TRUSTING IN

COMPUTER SYSTEMS

William Samuel Harbison

Wolfson College

A dissertation submitted for the degree of
Doctor of Philosophy in the University of Cambridge

May 1997

Original Work

I hereby declare that this dissertation is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other university.

I further state that no part of this dissertation has already been or is being concurrently submitted for any such degree, diploma or other qualification.

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration.

"There's no need to give up a good theory, just because it isn't true." "Air America." US film 1990.

Preface

The thesis of this dissertation is that there is no such thing as a computer system. Or to be more specific, we are unable to agree with the concept of "a computer system" as being something monolithic that can be represented by a single conceptual model, which is internally and externally consistent, and which behaves as a whole in a uniform and predictable way, under all foreseen circumstances.

There may be little initial disagreement with a statement such as this put in this way, yet we find in practice that it is just such a view which is usually applied. The assumption that a system is a single entity about which global statements can be made (such as "the system is secure" or " the system works") is unfortunately all too common.

Many aspects of computer systems such as availability, reliability, functionality (sometimes too much as well as too little) and ease of use are the subject of criticism by both users and operators; and also by many sections of the public, who although not necessarily users of the systems directly, can be seriously affected by their operation.

This dissertation examines limitations of design in some current distributed computer systems, and proposes approaches that can help us that better reflect the true needs of all participants, including operators, users and other affected parties. In particular, we have looked at ways in which the design of systems could be improved by a systematic approach to the identification and reconciliation of the many different assumptions that are held by the separate parties with a stake in the system.

We observe that the goals, objectives, concepts and assumptions of the various parties involved in the system (such as designers, programmers, operators and users) seldom, if ever, seem to coincide.

Work in the areas of Software Engineering[1] and Requirements Analysis[2] has attempted to address the problem of the unambiguous

[1] *Software Engineering is a term that was introduced in the late 1960's in an attempt to focus attention on the inadequacies of software development at that time. It was an attempt to suggest ways of bringing engineering style disciplines to the software development process. A recent overview of methods and techniques can be found in [RT96].*

[2] *Requirements Analysis is a term that is applied to the process of establishing and documenting the user and functional needs of a system. A good introduction to Software Requirements Analysis can be found in [D90] and [S90].*

specification of software systems requirements. Several books on Software Engineering cover the topic briefly (see, for example, [GM86], [JT79], [P87], [S89]), and [IEEE85] provides a good overview of Software Requirements Engineering work in the US; but there are relatively few publications that are exclusively concerned with requirements specification in general. It appears to be a common characteristic of the techniques we have come across that they assume a centralised approach; with the system (and, in general, only the software component) being developed by a team with shared understanding and a common specification.

We believe that there is a need for a more detailed characterisation of the various participants in a computer system, and of their roles and underlying assumptions. This would lead to better identification and understanding of those areas where the assumptions are at odds with one other, rely on undefined capabilities or or are open to being understood (and therefore implemented) in more than one way.

We have sought a methodology that will allow us to better identify those areas of possible conflict or lack of knowledge, and we have looked for ways to improve the systems engineering approach to the design of computer-based systems in a practical manner that can be readily understood and easily applied.

We propose that systems are examined in a manner that analyses the conditions under which they have been designed to perform, examines the circumstances under which it has been implemented, and compares the two. We believe that such an approach is essential since in our experience we have (sadly) seldom found the two situations to be the same. Unfortunately, we find the application of designs for one context being applied inappropriately to another.

We are proposing that anyone planning to design a system, or part of one, should look at it from the point of view of each of the participants. This should encompass all of the components - including users[3] and implementers - to see what is being relying on by the various parties and to resolve differences in assumptions and approach[4] .

---

[3] *For examples of approaches to the problems involving the difficulties of accommodating human operators when designing interactive systems see [WA91].*

[4] *We have come across an approach that seeks to codify this ([M79], [L85]), though it appears to be poorly documented. Other conceptual modelling techniques are more commonly found; see, for example, [A77]. [BGM85], [DM88], [SR77], [YC79]. Though we should not forget that non-functional requirements tend to be so varied and complex that natural language must be used for their expression.*

We begin our analysis by looking at what is being trusted in a system, or for what a system is being trusted, and by what or whom. We examine some approaches developed in a (military) security context and in widespread use in commercial distributed systems. We demonstrate how the inappropriate application of these concepts can lead to unanticipated risks to the system and its users.

We show how the usual use of trust as a *system* property[5] can restrict our ability to reason about the security properties of a system. We introduce a new concept of trust that we show to be more useful for the analysis of the risk characteristics of a system. In particular, we show how our approach can be extended to the analysis of sub-systems and even to individual systems components.

We propose that trust be considered as a "relative" concept rather than the more usual usage as a system property; and that trust is a *substitute* for knowledge rather than the result of it. We show that although the concepts originate in a security domain, they are equally applicable to the analysis of risk throughout a commercial system, its components and its users.

References

[A77] M.W. Alford. A requirements engineering methodology for real time processing requirements. *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, 1977, pp. 60-69.

[BGM85] A. Borgida, S. Greenspan and J. Mylopoulos. Knowledge representation as a basis for requirements specification. *IEEE Computer*, Vol. 18, No. 4, 1985, pp. 82-101.

[D90] A.M.Davis. Software Requirements, Analysis and Specification. *Prentice-Hall* 1990.

[DM88] T. DeMarco. Structured Analysis and System Specification. *Yourdon Press*, New York 1988.

[IEEE85] *IEEE Computer*, Vol. 18, No. 4, April 1985.

[GM86] Narain Gehani and Andrew McGettrick (eds.). Software Specification Techniques. *Addison-Wesley* 1986.

[5] *Many approaches to requirements analysis adopt the principle: "It should only specify external system behaviour [H80].*

[H80] K.L.Heninger. Specifying software requirements for complex systems. New techniques and their applications. *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 1, 1980, pp. 2-13.

[JT79] Randall W. Jensen and Charles C. Tonies. Software Engineering. *Prentice-Hall*, Englewood Cliffs, NJ. 1979.

[L85] M. Looney. CORE - A Debrief Report. *NCC Publications*, Manchester 1985.

[M79] G. Mullery. CORE - a method for controlled requirements specification. *Proceedings of the 4th. International, Conference on Software Engineering*, Munich 1979.

[P87] Roger S. Pressman. Software Engineering, A Practitioner's Approach. *McGraw-Hill* 1987.

[RT96] C.V. Ramamoorthy and Wei-tek Tsai. Advances in Software Engineering. *IEEE Computer*, Vol. 29, No. 10, October 1996, pp.47-58.

[S89] Ian Sommerville. Software Engineering, Third Edition. *Addison-Wesley* 1989.

[S90] D.A.Stokes. Requirements Analysis. J.A. McDermid (ed), Software Engineer's Reference Book. Butterworth-Heinemann 1990.

[SR77] K. Schoman and D. T. Ross. Structured analysis for requirements definition. *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, 1977, pp. 6-15.

[WA91] George R.S. Weir and James L. Alty. Human-Computer Interaction and Complex Systems. Academic Press, London 1991.

[YC79] Edward Yourdon and Larry L. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. *Yourdon Press*, Englewood Cliffs, NJ. 1979.

"It is much easier to get forgiveness than to get permission" - (Attribution unknown).

Acknowledgements

I would like to thank all those who have helped and encouraged me to complete this dissertation under what turned out to be far from ideal circumstances.

My first thanks must go to Dr. Peter Wilby and David Talbot for their fulsome support of my initial application to Cambridge. Next, and by no means second, is Professor Roger Needham who was prepared to accept my application and also to become my supervisor. It was his encouragement that led me to investigate the (then) new area of trust and delegation in distributed systems.

I consider myself to be both lucky and privileged to have been at the Computer Laboratory at the same time as Rafi Yahalom, Li Gong and Mark Lomas; friends and colleagues. My thanks to them for the many enjoyable and inspiring times; especially those at The Anchor, The Clarendon Arms, The Eagle and occasionally the Cork and Bottle.

Throughout my time at the Computer Lab. Dr. Jean Bacon. has shown consistent interest in and support of my research; Jean, my many thanks. My thanks also to Professor David Wheeler for his insight, wit, and inability to let anyone get away with a sloppy thought.

My deep gratitude goes to Professor Bruce Christianson of the University of Hertfordshire; for hosting me there, and for taking the time and effort to visit me to discuss my research. His continuing interest, and efforts, in support of my work to finish this thesis have been truly inestimable.

Without the patience, perseverance, support and understanding of Roger Needham, this thesis would never have been completed and submitted. Roger, my sincerest thanks.

My research has not been supported by any external body; and has therefore, in reality, been supported by my family. My very deep appreciation goes to my daughters, Jennifer and Catriona, for their long-suffering patience and sacrifice over the years, and for their continuing encouragement. Above all, for their confidence in me and their undemanding love and support.

Contents

"Sed quis custodiet ipsos custodes?" - Decimus Junius Juvenalis. Satires, vi. 347

Summary

We need to be able to reason about large systems, and not just about their components. For this we would like to have conceptual tools that will help us to understand the behaviour of these systems, and to help us make sense of other, possibly conflicting, views.

In this dissertation we have sought to indicate the need for a new methodology that will allow us to better identify and understand those areas of possible conflict or lack of knowledge, and we have looked for ways to improve the design of computer-based systems in a practical manner that can be readily understood and applied.

In particular, we have taken the concept of trust and how this can help us understand some of the basic security aspects of a system. We have paid particular attention to the nature and type of assumptions that are made both within and between computer systems when they seek to communicate with each other.

The work contained in this dissertation has been motivated by a belief that the design and implementation of many computer-based systems in operation today do not meet the needs of users and operators; and by a strong desire to identify ways in which the design and engineering of such systems can be improved.

We note that many assumptions are frequently made on a *de facto* basis and which are frequently not acknowledged or even recognised for what they are. We show that an incomplete understanding of what is being assumed, relied upon and trusted can lead to an inadequate understanding of true vulnerabilities of systems. We examine various trust aspects of systems and introduce a definition of trust that we believe can help towards a greater understanding of system weaknesses.

We propose that systems are examined in a manner that analyses the conditions under which it has been designed to perform, examines the circumstances under which it has been implemented, and then compares the two. We believe such an approach to be essential since we have (sadly) seldom found in our experience the two situations to be

the same. It is unfortunately all too common to find the application of a design for one context being inappropriately implemented in another.

We are proposing that anyone planning the design of a system or part of a system should look at it from the point of view of each of the participants, and that this should include all of the components - including users and implementers - to see what they are relying on and to make sure that these assumptions are compatible.

We look at this problem from the approach of what is being trusted in a system, or what a system is being trusted for. We start from some approaches developed in a (military) security context and in widespread use in commercial distributed systems, and demonstrate how the inappropriate application of this concept can lead to unanticipated risks to the system.

We show how the usual use of trust as a system property can restrict the ability to reason about the security properties of a system; and we introduce a new notion of trust that we show is more fruitful for the analysis of the risk characteristics of systems. In particular, we show how, in contrast, our approach can be applied to the analysis of sub-systems and systems components.

We propose that trust be considered a "relative" concept, in contrast to the more usual usage, and that it is not the result of knowledge but a substitute for it. We show that although the concepts arose in a security domain, they are equally applicable to the analysis of assumption and risk throughout a system and its components.

In contrast to the standard use of trust as a property of a system, our notion of trust applies only within the context of a specific *viewpoint* from which to judge risks. We argue that it is only after the introduction of a specific context from which trust is to be judged, that we can understand many of the *intrinsic* vulnerabilities of a distributed system.

We have introduced the concept of there being more than one viewpoint from which to describe the behaviour of a system, and therefore the trust relationships that pertain. The utility of this concept lies in its ability to enable the nature of the risks associated with a specific participant to be measured, whether these are explicitly recognised and accepted by them, or not.

We propose a distinction between *trust* and *trustworthy*, and demonstrate that most current uses of the term *trust* are more appropriately to be viewed as statements of *trustworthiness*.

In particular we propose that trust is more properly understood and used as a *substitute* for knowledge; rather than the traditional "Orange Book" [DOD85] concept of it being the result of knowledge; where something is trusted if it exists within the security boundary of the system, and can violate the security policy of the system.

References

[DOD85] Department of Defence Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, US Department of Defence, Washington, DC, USA, December 1985.

"A computer is a machine that allows you to make more mistakes in a shorter period of time than any other human invention ..... except tequila." Unknown.

Chapter 1

Introduction

1.1 Background

Security in computer-based systems covers a number of diverse topics ranging from the physical protection of computers, peripherals, media and associated support services; to the design of specialised operating systems, communications systems, protocols and data encryption algorithms.

The discussion of security of information systems is slightly more restricted typically covering protection of availability, integrity and confidentiality of information systems and their associated data. This includes, for example:

- Access (or denial thereof) to the computer system, programmes or data;
- Modification to or running of programmes;
- Control and sequencing of programmes and data;
- Data and programme correctness;
- Audit trails and transaction logs.

These topics are often considered together because of their reliance on common mechanisms for their implementations within many computer systems.

While computers and their associated peripherals were confined to the computer room, issues of security were relatively straightforward and consisted primarily of physical checks and barriers. These could still be somewhat sophisticated and complex in some specialised areas; such as military, governmental, nuclear and other hazardous environments; but once systems and their controls moved outside a common physical area, with significant elements being distributed, the complexity of ensuring the security of the system as a whole increased in a major way.

The connection of computers to communications networks heralded a new era in information technology. However the advent of distributed computing created a host of new problems for those concerned with ensuring and assuring the correct operation of these systems.

Major conflicts of objectives now began to occur as the goal of making computers and their facilities more accessible to users started to conflict with requirements for the integrity of the systems and their data, and for combating unauthorised access.

Some consider that the implementation of sophisticated access control systems would provide for sufficient security in a system (c.f. "Orange Book" [DOD85]); and concentration on access control mechanisms can be seen in much of the literature.

Within military systems, the concept of "trust" is primarily associated with access to information and the associated controls for achieving this, and much work has been done on the design of "trusted" systems and components which implement these concepts.

In the "Orange Book", the concept of trust is treated essentially as being an attribute of a *system*. It is implicit in its usage that there is just one viewpoint from which trust, and the security of a system, is to be judged. We believe this usage to have limited utility when applied to commercial open distributed systems: and this is also possibly the case in some military applications.

In contrast to the use of trust as being a property of a system, we propose the notion that the concept of trust is best applied only within the context of the specific *viewpoints* from which risks are being evaluated. We argue that it is only after the introduction of such specific contexts from which to judge trust, that we can appreciate many of the intrinsic vulnerabilities of distributed systems.

A considerable amount of recent research has been devoted to the topic of cryptographic protocols, with particular emphasis being placed on public key cryptosystems. The concept of the public key cryptosystem was introduced in 1976 by Diffie and Hellman [DH76].

A public key cryptosystem uses *two* keys (a key pair) - one key for encryption and the other (different) one for decryption. Each key of the

pair act as a cryptographic inverse of the other, and knowledge of one will not of itself enable determination of the other[1] .

Such a focus is very understandable since it was the introduction of public key cryptography that for the first time made it possible to communicate securely over a non-secure channel without the need to use previously agreed keys. This was because it was no longer necessary to resort to the approach of having to communicate a secret key over an insecure communication channel, or by some other means of distribution, with the resulting risks involved.

Public key systems frequently rely on third party elements for the secure delivery and verification of important information concerning participants, for example, their identities and public keys. This will often be in addition to the use of these third parties in their primary roles as active components of the system that are providing services and facilities such as routing and message delivery.

Consideration of the security role of these third parties indicates that the military use of the concept of trust is not strictly appropriate to their activities. They are rightly referred to as "*trusted* third parties", but the nature of the trust involved is not easily captured using the "Orange Book" model.

Using the concept of "trust" we introduce in this dissertation, it becomes clear that they are trusted by many different parties and for many different things. It also becomes apparent that not all users of these services will understand just what it is they are trusting these third-parties for. This is especially likely to be the case when users are not aware that these third-parties are also undertaking other active roles within the system; roles that might sometimes be incompatible with the nature of the trust being placed in them.

Whatever we might feel about the effectiveness and security of current computer systems, public or private, it is becoming more difficult to avoid contact with them in our daily lives. For large numbers of people it would be difficult to lead a normal life without resort to these systems, encompassing as they do, local networks, Internet, WWW, electronic mail services, specialised private networks and databases, medical systems and cash dispensers, for example.

[1] *For more detailed coverage of cryptographic techniques the reader is referred to [K87], [P89], [W90], [S96].*

As computer-based systems become increasingly more widespread and pervasive, larger numbers of people are finding that they have a specific need or requirement to use them, or have great difficulty in avoiding them, if they are to go about their daily lives with a minimum of hassle and disruption.

Many systems which appear to have remained unaltered over many years, have probably been updated and are now relying on computer-based third-parties, unbeknown to the user; and sometimes also to the operator of the system.

It is, however, an inescapable consequence of the way distributed computer systems are implemented, that the easier and greater the access to them becomes, the more their vulnerability increases.

Some consider that we are moving towards an age of "information terrorists and information assassins"[2] ; where disruption to commerce is an easier and more effective way of targeting advanced industrial nations, than the more traditional and violent methods previously used. The target now becomes the information and associated communications systems that underpin modern industrial societies; the age of "information warfare" has already probably begun.

People and organisations are probably able to survive the destruction of most if not all of their physical assets and possessions, which can be repaired or replaced in most instances; and where this is not the case, substitutes can usually be found.

However, the irretrievable loss or corruption of data can prove to be a situation from which there is no practical means of recovery, with a resultant disastrous outcome. For many systems, stored information and data represent a form of active and living memory, the damage and loss of which can never be fully compensated for.

The increasing pervasiveness, even intrusion, of computer-based systems into the daily lives and activities of the population at large has meant that many more people have become aware of the reliance that organisations place on these systems. Many systems fail to live up to the claims made for them of making life easier for those using them, often the opposite being the case; and frequently neither the systems,

[2] *P. Strassman, former head of IT at the US Department of Defense, speaking at the Fifth World Congress of EDI Users.*

nor those who run them, appear to be accountable to those who are required to use them.

When computer-based systems go wrong in some way, especially when this is as a result of human error, the failures are often initially denied. This can be the result of the operator of the system not recognising the particular failure mode, and therefore being more likely to blame the ("untrusted") user rather than the ("trusted") system for any ensuing problems.

Examples of such problems abound, with reports of incorrect bills (sometimes ludicrously so), invoices for goods not ordered, "phantom" withdrawals from bank cash dispensers, and so on. Users can encounter great difficulties in trying to prove their case in the face of denials from the system operator and computer "experts", particularly when this is accompanied by a lack of knowledge of the workings of the system and with no access to it.

In addition to the computer-based systems that are encountered on a daily basis in dealings with industry and commerce, there are also large numbers of other computer systems that can have a major impact on our lives and on society as a whole, and which can have far reaching social consequences.

The growing use of information technology in both central and local government and their agencies has resulted in large amounts of computer-based data being accumulated on the population at large. Increasingly, these originally separate systems are being interconnected, with the result that many aspects of our lives can be affected by the use of computer-based systems in ways that are not always clearly understood.

Nor is this situation restricted to government and official computer systems. There are also many computer-based systems in the private domain that hold data on large sections of the population. The existence of many of these systems is not generally publicised.

These private systems also can have major effects on the populace, containing as they do data on most aspects of our interactions with the business world: mailing lists, telephone directories, purchasing preferences and history, credit details and so on. This data, which in

many instances is inaccurate as well as incomplete is increasingly being sold and traded on a commercial basis.

The numerous possibilities of things going wrong in interconnected systems that have been individually (and usually independently) designed - and in ways that cannot always be anticipated, let alone recognised - leads us to ask what is being trusted, how it is being trusted, and by whom.

Gladstone said that it was Parliament's great duty to hold to account those who run the country. This leads us to ask the question of who is ultimately responsible for the actions of the computer systems we must use, and to whom those responsible are accountable.

We see that many of today's computer-based systems are characterised by tension, and sometimes even conflict, between the users and the operators of the systems. This is frequently the result of differing expectations of what the systems should do, and how they should behave.

At even a simple level we can see examples of systems operators who do not trust their users, and conversely, users who do not trust the computer systems they are often compelled to use.

We believe that considerations such as these indicate that it is not adequate to consider there to be only one viewpoint from which the acceptability of a computer system is to be judged. Different participants in a system have different "stakes" in it and its correct operation, and therefore have different things at risk.

We therefore believe that there is benefit to be gained from first considering from which viewpoint a system is being judged, before evaluating whether the behaviour of the system is acceptable or not.

It could be argued that all viewpoints are essentially the same and will therefore result in broadly similar results. However, if we consider that there are at least three main groups with a direct interest in the working of the system: operators, users, and designers: then even in the case where all groups belong to the same organisation, it is probable that their respective starting points and assumptions will differ in some significant ways from those of the others.

We maintain that this implies that they each view the risks associated with the system in a different way, and that in turn this means that they each must be trusting the system for different things. We note that this is not the conventional way of treating trust in relationship to computer systems.

1.2 Aims of this research

Against the background above, this dissertation sets out to examine assumptions underlying the design, operation and use of computer-based systems, and in particular distributed systems, where reliability and integrity considerations are an essential part of their use and operation.

It is clear to us that once systems *and their controls* are moved outside of a common, physically secure area, with large parts of the system being (possibly geographically) distributed, then the issues surrounding the integrity of the system and its operation are magnified and changed in possibly unpredictable ways.

We look at issues involving systems design, security protocols and trust[3] and ask how we can specify the properties and behaviour of the system in such a way that all parties involved can be satisfied with its operation; and we should perhaps note that they may all be satisfied for different reasons.

We shall introduce a new notion of trust as it is applied to computer-based systems and show how its use can enable us to analyse a computer-based system from a number of different perspectives. We show how our approach can be used to identify and measure the risks associated with the system for any particular participant.

We examine issues such as:

> "What do we typically take on trust in current systems?"
> "How can we specify the security properties of a system so as to minimise the need for trust?"
> "Is it possible to do without trust altogether?"
> "To what extent can we make trust explicit rather than implicit?"

[3] *The Concise Oxford Dictionary defines trust as "Firm belief in reliability, honesty, veracity, justice, strength, etc. of person or thing".*

We look at what is trusted, by whom, and to what extent. We note that behaviour consistent with our notion of trust, (though not necessarily with that used as the basis of the "Orange Book"), is to be frequently observed in many users and operators in regard to their dealings with their computer systems.

We question whether people are always aware of the nature and exact amount of trust that they actually place in the systems they use. In such deliberations it is perhaps worth noting that some estimates[4] are that as much as 85% of computer crime is committed by insiders with validated access privileges.

With this as a consideration we would propose that any analysis of risk and trust in relation to the operation of a computer-based system should also include as necessary constituent parts, those people involved in the operation of that system, in addition to more traditional components such as servers and programmes.

In addition, we believe that in consideration of the nature of some insider attacks, the inclusion of the designers and implementers as constituent parts of the system should also be considered in any such analysis.

We wish to investigate the implications of trust as applied in the context of computer-based systems, their constituent parts, and the environments in which they operate; and use this as a way of determining critical assumptions and vulnerabilities.

The classical "Orange Book" approach to security of computer-based systems views the system as one that can be contained within a single boundary, with an "inside" and an "outside", and associates "trust" with being inside or outside this "secure" boundary.

We would not disagree with the concept of a boundary defining the extent of a computer-based system. However, if a system is sometimes taken to include those people involved in operating it, and sometimes even those involved in the design and implementation of the system, then we would argue that there is not necessarily a single and unique boundary of a system under all considerations. A system will have a different "inside" and "outside" for different principals and viewpoints.

[4] *P. Strassman, former head of IT at the US Department of Defense, speaking at the Fifth World Congress of EDI Users.*

We show how the introduction of the concept of "viewpoints" can be used to distinguish the different risks that different participants have with respect to the system.

We look at the terminology that is used to describe various formulations of the concepts of "trust" and "belief", and consider what useful distinctions can be made in these and many other of the terms that are commonly used in the specification and analysis of the security properties of computer-based systems.

1.3 Scope of this work

The initial motivation for this research came from a desire to examine what is meant by trusting a computer system to run correctly a particular piece of software (a payroll programme, for example); and the implications of such trust being misplaced.

We can be more specific and ask how we can assure ourselves that a given piece of software operating with known characteristics and behaviour running in a known environment over which we have direct control, will run identically in a similar system, over which we have no direct control.

For the purposes of this discussion we could consider a very simple programme such as one that on being presented with an integer value between specified limits would return the value of π to the number of significant digits specified. For such software, the correctness of the basic programme within its given environment would be known for a wide range of input values, even if it were not possible for the programme to be proved to be formally correct.

We wish to ask how we can trust software to behave in the same (known) manner as in our original system, *and in no other way*, in a system over which we have no control, access to or *directly* verifiable knowledge. We look at the value to us of trust as a concept, and how a proposed new usage can help us in our understanding of the behaviour of computer systems.

We examine the nature of trust and its relationship to other commonly used terms such as belief, reliance and delegation. We note in passing

that it is not the purpose of this dissertation to examine how to determine the correctness of a programme in the first instance.

We have sought to analyse why the actual ways in which systems behave differ from users' expectations: i.e. why users are surprised when and how the system they are using fails for other than a straightforward hardware breakdown; and have examined the role that trust plays in users' expectations.

In contrast to the common approach of "trust only what you can verify firsthand", we shall argue that it is preferable to consider trust to be a *substitute* for knowledge, and indicate how the concept of risk can be used to quantify the individual consequences of misplaced trust.

The initial vehicle we use to examine these ideas is that of a Trusted Message Service. We look at a very simple model of how a trusted message server might operate. We consider initially the simple case of a server with only one message to deliver, and with that message to be delivered to only one principal.

We examine what is involved in determining that the message delivery was successful, and we show that some assumptions and observations require additional, and sometimes conflicting, actions to be taken, if the security of the system as a whole is to be considered. We show that these additional actions lie outside the original basic design of the message server mechanism.

We subsequently apply the insights gained from this analysis to some specific areas, and identify previously unconsidered vulnerabilities, and identify the dangers of "hidden" delegation.

1.4 Basic considerations

Some feel that sophisticated access control mechanisms will suffice to ensure the integrity and security of a system, indeed a key concept of trusted systems standards is to control rather than facilitate access to the system [KE93]. However, as we have already noted, it is estimated that up to 85% of computer crime is committed by insiders with valid access.

This suggests that a concentration on access control will not of itself be sufficient to avoid incurring significant risks to the integrity of a computer system; though it is, of course, a necessary component. There is growing awareness that no computer system is fault free, and increasing attention has being paid to identifying the risks associated with such systems, and to methods of confining and controlling them.

The approach is often that it is now known that what has been implemented does not behave correctly under certain circumstances - representing a risk to the operation of the system; and that it is possible to retrospectively supplement or modify the design, in order to minimise or control the newly identified risks.

Many assumptions underlying the design of a computer system are often made on a *de facto* basis and are not always acknowledged, or even recognised, for what they are. We show how an incomplete assessment of all the assumptions that are actually being made can lead to an inadequate understanding of many of the vulnerabilities a system.

We examine various trust aspects of computer systems and introduce a definition of trust that we believe can help towards a greater understanding of system weaknesses. We propose that risk be considered as a measure of trust in a system; i.e. a measure of that which cannot be directly verified; rather than as a description of something that has bee be identified but not been provided for in the original design.

1.5 Constitution of this dissertation

In this chapter we have outlined the subject matter of this dissertation and provided some focus for the general topic areas covered.

In **Chapter 2** we outline some of the basic ideas, issues and concerns that can affect the operation and security of a distributed computer system, contrasted from a number of different viewpoints.

In **Chapter 3** we discuss the desired behaviour of a simple trusted message server and use this to introduce some of the design considerations and conflicts that can occur during implementation.

In **Chapter 4** we examine a well-known cryptographic protocol and look at some of the trust issues involved in its use.

In **Chapter 5** we investigate a number of existing trust models, and compare their assumptions and differences as a means for examining matters of trust in computer-based systems.

In **Chapter 6** we consider the conflicts that can occur in a distributed environment between the design of system integrity and the design of data integrity, by examining some of the implementation issues arising from the common practice of sharing data and resources.

In **Chapter 7** we summarise our findings and point to further areas of research that can be developed as a follow-up to the work done in this dissertation; and the implications of our research for other areas of computer science.

References

[DH76] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, November 1976, pp. 644-654.

[DOD85] Department of Defence Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, US Department of Defence, Washington, DC, USA, December 1985.

[K87] Neil Koblitz. A Course in Number Theory and Cryptography. *Graduate Texts in Mathematics 114*, Springer-Verlag 1987.

[KE93] R. Kuhn, P. Edfors, V. Howard, C. Caputo, T.S. Phillips. Improving Public Switched Network Security in an Open Environment. *IEE Computer* Vol. 26, No.8, August 1993, pp. 32-35.

[P89] C.P.Pfleeger. Security in Computing. *Prentice-Hall International, Inc.* 1989.

[S96] Bruce Schneier. Applied Cryptography. *John Wiley and Sons, Inc.*, New York 1996

[W90] Dominic Walsh. Codes and Cryptography. *Clarendon Press,* Oxford 1990.

"A shared secret is no longer a secret." W. S. Harbison[5]

Chapter 2

Security in Systems Design

2.1 Introduction

Traditionally, computer systems have been thought of as being comprised of hardware, software and data, and discussions of security issues have been confined mainly to these three components.

The OECD defines an "information system" in much wider terms: as, collectively "computers, communication facilities, computer and communication networks and data and information that may be stored, processed, retrieved or transmitted by them, including programmes, specifications and procedures for their operation, use and maintenance" [OECD92].

Security of information systems can be generally described as the protection of the availability, confidentiality and integrity of the systems and their constituent parts.

"Trusted Systems" were first discussed within the context of the security of military computer systems in the United States, and codified in the "Department of Defense Trusted Computer System Evaluation Criteria" [DOD85] - the "Orange Book".

This work originated in studies instituted by the US Defence Science Board in October 1967 into computer security safeguards that would protect classified information in remote-access, resource-sharing computer systems.

As such it seems reasonable to expect that the considerations resulting from this work should be of considerable relevance to many of today's computing environments. Indeed, the principles underlying the "Orange Book" have been widely used in the design of many commercial systems where security is considered to be an issue (a large

[5] *We have subsequently discovered that Benjamin Franklin had recorded an epigram [F 733] of a similar, if somewhat blacker, nature: "Three may keep a secret if two of them are dead.". This was based on a similar maxim from a contemporary of his, Edward Everett: "If you want your secret kept, keep it".*

number of distributed and on-line systems in current use).

It can be argued that military models of security are not always the most suitable basis for the design of a commercial open distributed system, given their predominant focus on secrecy and access controls. There is a concentration within military computer systems on the control of access to information: and these aspects are well- described in the "Orange Book". However, in the world at large this singular and specific concern is uncommon, and their are other and more general issues of security that arise and which are not particularly well addressed by such a restricted view[6] .

At the heart of the security considerations for a computer-based system is the concept of the existence of potential threats to the proper running and availability of the system and its data. That is, there is an assumption that some forms of adverse elements exist that could compromise the system in some manner, and that the security of the system could be vulnerable to such hostile action.

We will see that different security regimes will presuppose different kinds of threat, and will act to provide safeguards only for their perceived vulnerabilities. It is usually clear what the perceived threats are thought to be, although it will be seen that these threats are seldom completely stated explicitly. This sometimes results in situations where particular security regimes can be shown to be vulnerable in ways that they would probably find to be unacceptable, were all the assumptions to be known in a complete and explicit manner.

We will show that many security threats are made possible by an incomplete, and sometimes erroneous, understanding of the behaviour of the computer system by its designers, operators and users; and where the *explicitly* stated assumptions represent but a fraction of the total number of those that can affect the security of the system. This can result in the reliance on parts of the computer system that will be vulnerable forms of attack that otherwise would be known.

[6] *Some of the limitations of the "Orange Book", particularly the need for "interpretations" to extend its applicability, were subsequently addressed in the "Canadian Trusted Computer Product Evaluation Criteria" [CSSC93].*

2.2 Trust and Security

It is often the case that many of the assumptions that affect the security of a particular computer system can be deduced only from the context in which that system's security mechanisms operate. That is, from observations of the actual operation of the system, rather than from its specifications; though as we shall see later, this is not as simple as might first appear.

Many commercially developed UNIX systems have sought validation under the "Orange Book" criteria; and whilst this may have been motivated initially by a desire to supply "off the shelf" systems into sensitive government applications, such systems are also being offered in the general marketplace. We are of the opinion that this is not necessarily a beneficial development for commercial users.

We have looked at the implications of applying the military-derived model of trusted systems to the more commercial world of distributed computer systems. We found that such application can result in a misconceived view of the overall security of a system.

In particular, we note that the inter-connection of "Trusted Systems" does not of itself produce a system with the same security characteristics; and can lead to systems whose overall security is considerably less than that of its constituent components[7] .

Trust is considered by some to be a transitive property of a system[8] [e.g. CB94]. We believe that to take such a view is to confuse a number of separate and distinct concepts such as trust, belief and reliance. We will give definitions and examples of these concepts which we believe will illustrate the differences between them.

In particular we believe that there is confusion between the related concepts of trust and trustworthy. We propose that there is a distinction between these two concepts which we can summarise as: "trust is something *I do;* trustworthy is something *it is*".

In other words, "trust" could be considered to relate to a state of knowledge of a principal in the system, whereas "trustworthy" relates

[7] *We should point out that we are concerned here to highlight issues of bad design practice rather than with the notion of composability (see for example [CSFW8]).*

[8] *A relationship (¬) is said to be transitive when the following holds: if A¬B and B¬C then A¬C.*

to a property of some entity within the system (which could include a principal).

Another way of attempting to quantify the concept of trust is to look at the value *to us* of what is being trusted "letting us down". We can consider that to trust something is to "risk" the cost of what we are trusting going wrong. That is, trust can be quantified by the (negative) value recovering the situation.

From this viewpoint, the study of trust can be considered to include the study of those risks that are accepted, knowingly or not, within a given system or component part of a system.

Trust then becomes a statement about the subjective position of one party in a system to other parties and components. It is therefore quite possible for two or more principals to have different views on trusting something within a system.

With better knowledge and understanding of what is being trusted in this regard - and therefore what is being risked - we will be in a position to make more considered judgments on the balance of risks versus costs.

It is usual to think that we trust what we know - a good friend, our house not to collapse, etc[9] . What we are "trusting" here is consistency of behaviour: but consistency is not the same as certainty; and a prediction is not the same as fact. We suggest that "consistency of behaviour", which we often see used as a measure of trust, is better considered to be the (related) concept of "trustworthy".

We propose that, in contrast to this view, the concept of trust is better associated with the idea of what we *don't* know rather than what we do know. It can therefore be considered as a *substitute* for knowledge instead of a representation of it. Considered together with the notion of the existence of many different and valid viewpoints of a system, trust can be seen to be a *subjective* concept.

With this in mind, we suggest that we progress in our understanding of a system when we find ways of *reducing* our trust in it, of

[9] *In his recent book, Frances Fukuyama considers trust to be "the expectation that arises within a community of regular, honest, and cooperative behaviour, based on commonly shared norms, on the part of other members of that community [F95]".*

quantifying the costs of our (misplaced) trust and by seeking ways to replace trust with knowledge. In order for us to do this we first must understand the extent to which we are *actually* trusting the system.

It is not our aim to write a philosophical dissertation, and we therefore do not intend to enter into a detailed discussion concerning the nature of knowledge and truth. We rely, instead, on an intuitive understanding of these terms.

It is unlikely that we can ever have complete knowledge of anything in the real world - and we therefore have to use only as much knowledge and reasoning as is appropriate to the given situation. As an example, we usually "trust" a keyboard to generate the code we *expect* from our in-built assumptions. If we were to depress a "Q" key on a keyboard we would expect it to generate the appropriate code for the letter "Q". If, however, the keyboard had been configured to the French language it would generate the code for the letter "A".

When the keyboard is connected to a computer, it is usual for there to be some kind of visual feedback which would display the letter the code had generated; i.e. the letter "Q" or "A" would appear on a display; and we would then have a confirmation of our input.

However, it is worth noting that this feedback is usually only *logically* coupled to the key depression and is not necessarily coupled *physically*. The importance of this can be shown when considering what might happen in the case of a cash dispenser.

An intermediate (and unknown of) device could be interposed in such a way as to relay the responses we are expecting back to us, while having a different set of dialogues with the central computer to which we think we are directly connected.

It is also possible that the cash dispenser itself is entirely bogus (as shown in a recent court case [R95]) and be merely emulating the real device, even to the extent of giving out cash.

In both of the above cases the user will have "trusted" the device to do what was expected by the user, but it has actually done something entirely different to those expectations.

2.3 Summary

Being explicitly aware of what we are trusting can enable us to design strategies for coping with possible risks. In the case of the cash dispenser, a suitable challenge/response protocol, coupled to a smartcard incorporating its own keyboard, can provide a high level of protection against the risks described. Whether such strategies are economically justifiable will vary from case to case and with the cost of recovering the potential loss.

In summary, we propose that to understand trust we need to understand the limitations of our *knowledge*. In contrast to what is often believed, we are proposing that trust is *relative*; and rather than being the result of knowledge, is in fact, a substitute for knowledge.

References

[CSSC93] The Canadian Trusted Computer Product Evaluation Criteria. Version 3.0e. Canadian System Security Centre, Communications Security Establishment, Government of Canada, 1993.

[DOD85] Department of Defence Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, US Department of Defence, Washington, DC, USA, December 1985.

[F733] Benjamin Franklin. *Poor Richard's Almanac,* 1733.

[F95] Francis Fukuyama. *Trust.* Penguin Books 1996.

[OECD92] Organisation for Economic Co-operation and Development. *Guidelines for the Security of Information Systems., Paris 1992. OCDE/GD(92)190.*

[R95] Regina v. Hodges and Moore. Southwark Crown Court, September 1995.

"Lookers-on many times see more than gamesters" - Francis Bacon. Essays, 48.

Chapter 3

A Trusted Message Server

3.1 Introduction

To understand what fundamental requirements are necessary in the provision of a trusted message service, we begin by initially looking at a simple system that deals only with the retrieval of a message.

So as not to complicate our understanding of the various relationships involved, we start at the point where the message has already been made available to the server. For the purposes of this discussion we make the assumption that the message has been delivered to the server in a secure manner.

We wish to examine what is meant by the statement that the message server is *trusted* to deliver a message to the appropriate party[10] .

3.2 Single Message - Single Recipient

Let us consider the following situation. ALF is travelling away from home base and is not in a position to establish or to maintain contact on a reliable basis during these travels. It has therefore been agreed beforehand that if it is necessary to contact ALF during this time, then a message will be left at a predetermined location. ALF has undertaken to check the agreed location at regular intervals. We refer to this message delivery mechanism as the Trusted Message Server (TMS).

We would like to consider what steps are necessary in order to ensure the intended successful outcome of this process[11] .

[10] *We shall see very shortly that this question alone is not simply determined. We need to ask from which viewpoint this is being discussed.*

[11] *There is an implication here that only one of the possible outcomes is the correct one. We will return to this point later. [n.b.: The measure of a successful outcome depends upon secondary criteria - see later discussion on whether the message should be destroyed if the TMS is tampered with. There is a strong case to be made, I think, that a relationship exists between desired outcome under adverse conditions, and what is sometimes known as security policy. Clearly a number of differing outcomes can be considered. These can be related to a number of different security policies; whether overtly stated, or maybe just implied.]*

It should be noted that for the purposes of the following discussion, we make no assumptions on whether the message is encrypted or otherwise. At this stage our concern is with the correct delivery of the message to the designated recipient. We would therefore like to establish those criteria needed to characterise the successful outcome of this procedure.

In an ideal world, and with no adverse conditions, a successful outcome to the delivery of the message could be could be considered to have occurred when:

1. It is apparent to ALF that a message has arrived.
2. ALF's identity is accepted by the TMS.
3. The TMS (correctly)[12] delivers the message to ALF.

These are clearly necessary conditions, and by some they might be considered to be sufficient. However, in a non-ideal world, it is probable that not all circumstances are favourable and not all agents are benign. Therefore, even at this stage it is instructive to look at this situation in a little more detail.

It has been stated that we are looking for the successful outcome to message delivery. There are, however, a number of possible viewpoints from which to judge this outcome, namely:

i. The originator of the message.
ii. ALF.
iii. The TMS.
iv. An intruder.
v. An impartial observer, which we will refer to as DEM.[13]

We shall examine what points of commonality there exist for these viewpoints. If it transpires that they are not all equivalent then we will then be faced with the problem of from which viewpoint the outcome should be judged; and why.

Let us start by looking at the assumptions we have so far made:

[12] *The use of the term "correctly" here refers to the integrity of transcription of the message.*

[13] *It is assumed that the DEM has a complete view of all principals, systems and activities within this universe of discourse.*

A1. The existence of the Message Server.
A2. The existence of ALF at the time of delivery of the message.
A3. The existence of a non-hostile environment.

If we now examine in turn the consequences of each of these assumptions not being true:

nA1. The absence of the message server can mean

i. The message server does not exist[14] ;
ii. There is no message;
iii. The message server has been removed or destroyed.

nA2. The absence of ALF can mean

i. ALF has not arrived;
ii. ALF has arrived but not checked for some reason.[15]
iii. ALF arrived but has moved-on (or returned to base);

nA3. The existence of a possibly hostile environment could lead to a large number of possibilities[16] , including:

i. The message has been replaced by a substitute one;
ii. The message server has been removed or destroyed;
iii. ALF has been apprehended or destroyed.

Clearly, these are only indicative of the kind of events that could occur under various circumstances. Our original and relatively simple considerations seem now to have grown into something rather complex and somewhat obscure. What are we to make of all of these possibilities?

The problems seemed to proliferate when we started to consider events when things might possibly go wrong. Let us go back to our original problem.

We can see that if everything is assumed to be right and if all necessary conditions for success are fulfilled then the conditions for the

[14] *There are a number of reasons why this might be the case including deception.*
[15] *There are a large number of reasons that could be considered including a) ALF has got the place and/or time wrong; b) ALF has been apprehended; c) ALF has been destroyed.*
[16] *Some of these possibilities are similar to those of the case where ALF is absent. It should be noted that depending upon what is assumed to have happened then a different set of end conditions may be sought. Refer to subsequent discussion of security policies.*

required outcome are quite simple and few in number. This is a serious number of "ifs" and might not be what we really want to rely on.

We have already noted that there is not just one viewpoint from which to look at this situation. Therefore, in examining the issues involved from different possible viewpoints let us start by considering two possible characteristics of success. The successful delivery of a message to the server, and the successful retrieval of a message by the rightful addressee.

In looking at what constitutes successful delivery, we would like to start by examining this from the two viewpoints of the TMS and the message originator, respectively.

We could therefore consider some additional requirements, which might be thought desirable but have not yet been stated explicitly. We might like to ensure that only ALF be aware that a message has arrived, and that only ALF has access to the message.

Explicitly, then:

A4. There should be no externally verifiable evidence of a message being left (external, that is, to ALF and TMS).

A5. Only ALF should be able to retrieve the message.

These two conditions could be regarded as adding concepts of "privacy" and "secrecy".[17]

Thus far we have tried to outline the conditions for ALF to receive the message "discretely" and for it not to be given out to another party by "mistake". If we continue to develop this line of reasoning then we might envisage a hostile environment where others *actively* seek ALF and the message.

Attempting to counter such possible active threats might lead us to add the following conditions[18] :

[17] *By secret here we mean that the message is not readable by any party other than ALF.*

[18] *Note that we have assumed (cf. AC5) that the original message could not have been retrieved by an other party. However, it is possible that it may have been removed or replaced by another message.*

A6. ALF should be able to distinguish the situations of no message from that of a destroyed or removed message.

A7: ALF should be able to distinguish between a correct message and a counterfeit message.

These conditions could be considered to cover such concepts as "denial of service" and "message substitution".

The seven conditions listed above are not the only ones that we might have produced; they are, however, representative of those that need to be considered, particularly in environments whose characteristics may not be ideal, and which might also be considered to be hostile.

We have now moved some way from our initial simple problem and it is probably worthwhile summarising the discussion so far of our basic message system.

1. It appears that three simple conditions will satisfy the initial problem statement.

2. Four additional conditions were then introduced to counter possible threats that might arise if the environment in which the service operates, can be considered to be hostile to some degree.

3. In addition, we have seen the usefulness of adding to such frequently used ideas such as message, recipient, etc., the concepts of:

   C1: Outcome.
   C2: Viewpoint.
   C3: Environment.

It will be seen that these three concepts will come to play an increasingly important part in our subsequent discussions on the nature of trust and its relationship to the various parties involved.

3.3 The TMS' Viewpoint

In a manner similar to that of ALF's we need to look from the point of view of the Message Delivery Service at what constitutes the basic requirements for successful message transmission.

M1. The TMS must be able to indicate that a message has arrived.
M2. ALF (correctly) identifies himself.
M3. ALF (correctly) receives the message.

[n.b.: The TMS should also not allow its message to be extracted by any unauthorised means.]

These conditions mirror the first three conditions for ALF. In addition we might also like to consider the following situations:

M4. The TMS should be able to distinguish a situation where ALF is seeking to retrieve the message, but doing so under duress.

M5. The TMS must be able to distinguish ALF from an impostor.
M6. The TMS must ensure that following a successful retrieval by ALF the message is no longer available[19] .

It is also possible to consider that under certain circumstances either or both of the TMS and ALF may wish to prove to a third party that the message was delivered or received (both respectively - and not), if duplicitness is to be a consideration.

It should be pointed out at this stage that there are other viewpoints from which an examination of this example could be conducted.

1. Ours. The (dispassionate and) remote observer.
2. An involved observer (seeking to record for future use).
3. An Intruder (seeking to alter or destroy).

An examination of any particular exchange will not necessarily result in the same observations or conclusions[20].

[19] *It is worth noting that in some systems the continued storage of a message following delivery to the recipient is considered a benefit. It must now be obvious that this can lead to additional risks if secrecy is considered to be important.*
[20] *See discussion of known plaintext attack of Needham-Schroeder.*

3.4 Rules For Successful Delivery

The following are a set of rules that could be applied to the Message Delivery Server (TMS) to achieve the major objective:

1. The procedure will be used once only[21].

2. The message is to be delivered to ALF and to no one else.

3. Upon correct delivery the TMS will destroy itself and any message.

4. Any message not retrieved within a given time period will cause the TMS to destroy itself and any message.

5. Any attempt to retrieve the message by persons unrecognised by the server will cause the TMS to destroy itself and any message.

6. Any attempt to tamper with the TMS will cause it to destroy itself and any message.

We can reasonably deduce from the above that this set of rules embodies the consideration that the *secrecy* of the message is more important than the *delivery* of the message when there could be a conflict between the two.

A system based on these rules will seek to enforce the principle that if a threat to the integrity of the system is perceived, then it will be preferable that no one - and that includes ALF - should retrieve the message, than that the possibility is allowed for the message to fall into unauthorised hands.

Clearly a different set of rules would need to be applied if it were deemed to be more important that ALF should receive the message than to protect it against any form of unauthorised disclosure.

It should be possible to reverse this thinking and to deduce the particular requirements pertaining to a specific protocol implementation from an analysis of its workings.

[21] *It has been suggested that these procedures be considered on a "per message" basis rather than once only. This will lead to a significantly different and more complex protocol than that considered here.*

Such a set of rules can be likened to the concept of a Security Policy[22], which consists of a set of rules within which the applicable process or procedure should operate, but which do not of themselves form part of the process or procedure.

We can see this in the example above where we can change the security policy (rules) without changing the procedure for message delivery. This could clearly result in a different possible overall outcome in the event of an attack on the system or some form of system malfunction.

3.5 Summary

The above analysis was undertaken on what is a relatively simple security model compared to many in actual operation. What we see, however, is that in common with other models there are a significant number of *undocumented* assumptions that must be made in order for the model to perform at all.

We have also seen that in even such a simple model as this, these assumptions can have a major effect both on the operation of the system, and also on the determination of the outcome; successful, or otherwise.

There are many security protocols in regular use in large numbers of systems and networks that are of considerably more complexity than the ones discussed above. These protocols are all purported to provide secure communications of one kind or another.

We believe that a systematic analysis of these protocols in a similar manner to that illustrated above is likely to reveal several assumptions that have not been explicitly documented, and which are critically necessary to the correct running of the protocol.

The lack of understanding of these fundamental assumptions when the protocol is being implemented is, in turn, likely to leave the systems in which they have been implemented vulnerable to (previously undocumented) attacks.

[22] *The existence of an explicit and well-defined security policy enforced by the system is a requirement for a secure computer system [DOD85]. See also [S91] for a discussion of some of the issues concerning "Security Policies".*

We give an illustration of this in the analysis of the Needham-Schroeder protocol in a later chapter.

Some of the uncertainties encountered by the use of an extremely simple protocol in the previous discussion on the message server can be ameliorated by making changes to the protocol. For instance, we could extend the protocol by adding a message from the sender that states that a message has not been been sent, if that is the case. The question then arises as to whether this changes the trust relationships.[23]

Although this would clearly be a different situation, it would look *the same* using the traditional notion of trust, being an "objective" system property; and the trust relationships would remain unaltered.

We contend, however, that the answer to the question above *must* be yes; since new messages will introduce new uncertainties into the system, as well as new information.

The consideration of trust as a property of a system can therefore be seen to provide a less powerful concept for the analysis of security weaknesses in computer-based systems than the concept that we have proposed in this dissertation.

References

[DOD85] Department of Defence Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, US Department of Defence, Washington, DC, USA, December 1985.

[S91] Daniel F. Sterne. On the Buzzword "Security Policy". IEEE 1991.

[23] *This example was suggested by Roger Needham.*

"He was a gentleman on whom I built an absolute trust." Shakespeare. Macbeth, I. iv

Chapter 4

Analysis of Security Protocols

4.1 Introduction

We would like to start by looking at the Needham-Schroeder Protocol [NS78], which is one of the first, simplest and most successful of cryptographic security protocols. It is a protocol that has been adopted as the basis for many operational security systems. This includes the Kerberos Authentication Server, created in the Athena Project at MIT [T88], utilised in what is perhaps the most prominent strong authentication service in wide use today [MNSS87, KN93].

We show how the inappropriate application of the Needham-Schroeder protocol can allow novel attacks on the protocol to be mounted. We have not found the two attacks discussed below to have been considered elsewhere.

4.2 Needham - Schroeder Protocol

Needham and Schroeder [NS78] were among the first to consider the use of cryptographic protocols for the authentication of communicating parties where secure computer communication in large networks is desired. Their paper states that within the context of secure computer communications, the term "authentication" means verifying the identity of the communicating principals to one another[24] .

We note that this definition of authentication does not encompass the issue of the verification of the transactions between principals. The linked issues of when and how sessions end are similarly outside the scope of the publication. (We would like to return to these issues later, as we believe them to be of crucial importance in the overall context of secure communications in computer networks.)

[24] *cf. COD definitions:*
Secure means safe against attack, impregnable, reliable, certain not to fail or give way.
Verify means to establish the truth or correctness (of) by examination or demonstration.

Three functions are discussed in the Needham - Schroeder paper:

1. Establishing an authenticated interactive communication.
2. Authenticated one-way communication (such as mail).
3. Signed communication for authentication of origin and integrity by a third party.

It is stated that secure communications in physically vulnerable networks depend upon encryption of material. Assumptions made are that an intruder can alter, copy, replay parts of, or all, messages; and can also emit false messages.

The protocol implicitly assumes that the principals have a secure environment in which to operate, and is designed only for those situations where a *mutual* choice of secure communications has been made, and no forms of compulsion are involved. Mechanisms that may be required to enforce compliance or to restrict information flows are not within the scope of the paper.

Needham and Schroeder have therefore consciously restricted themselves to looking at what could be considered to be a small and relatively simple subset of all of the possible functions that could be considered within the context of secure communications in large networks of computers.

It should be noted that Needham and Schroeder quite clearly state that their protocols should be regarded as *examples* that expose the issues of authentication in large networks and should not be looked on as fully engineered solutions to the overall security problems of particular applications.

As we shall see, even within such a restricted domain the issues involving the security of the communications between the principals involved can get quite complicated very quickly.

Needham & Schroeder presuppose, not unreasonably, that the use of authentication servers is *necessary* for the provision of a source of authoritative information on the keys belonging to the principals using large networks. We shall see that the introduction of a third party into the communication between principals can lead to unforeseen consequences for the security of that communication.

Keys are shared between the principals and the server respectively. New session keys are generated by the server for each session. These keys "*must* be unpredictable and *should* never have been used before".

It is not stated how this is to be achieved. The server could precompute keys and "remember" which key it has issued to whom; or an algorithm could be devised which accomplishes the requirement automatically and reliably. Professor Needham has suggested (in a private communication) that a physical random number generator would suffice. The adequacy of such a system would depend upon the specific implementation and key space, since a birthday attack could be possible [S92].

It is also not stated as to whether the session key requirements are meant to apply only in the context of a specific pair of principals, or whether they apply globally to all of the principals of a particular authentication server.

4.3 Needham-Schroeder Symmetric Key Protocol

In the symmetric-key protocol, where the same key is used for both the encryption and the decryption, authentication *depends* upon the two principals being the only two who know the key that is being used. Apart from, that is, possibly also the ("trusted") servers.

In the following protocol descriptions the notation used will be that described in the original paper: encryption is indicated by braces that are superscripted with the key used.

Protocol 1.

1.	A	$\rightarrow$	AS :	A,B,I_{A1}
2.	AS	$\rightarrow$	A:	$\{I_{A1},B,CK,\{CK,A\}^{KB}\}^{KA}$
3.	A	$\rightarrow$	B :	$\{CK,A\}^{KB}$

It is stated that the recipient's name *must* appear in message 2. The reason given is that otherwise an intruder could intercept message 1. and replace B's address with that of a different addressee (say) X. As an exercise in understanding some of the complex issues involved in the design and analysis of security protocols we shall examine this in a little more detail.

In order to better understand the dynamics of the intrusion we have found it useful to introduce a new notation: square brackets, [ ], denote the presence and activities of an intruder in the system. This notation is used within the following message to illustrate how the intruder, denoted by *X*, might operate.

Using the notation referred to above, the substitution attack that is possible where the recipient's name has not been included in message 2, is represented as follows:

4.	A	$\rightarrow$	[X: A,B,I_{A1} ; X Æ]	AS:	A,X,I_{A1}
5.	AS	$\rightarrow$		A:	$\{I_{A1},CK,\{CK,A\}_{KX}\}_{KA}$
6.	A	$\rightarrow$	[X: $\{CK,A\}_{KX}$]		(which X can decrypt of course)
7.	X	$\rightarrow$	$\{message\}_{CK}$	A :	

In message 4 the intruder *X* is monitoring *A*'s communications. It intercepts *A*'s message, substitutes its own identity for that of *B*, and forwards the modified message to the authentication server.

In message 5 the authentication server sends the session key to be used, encrypted with *X*'s key. Without a specific reference to the recipient in this message from the authentication server *A* has no way of knowing that the session is being set up with *X* and not with *B*.

In message 6 *A* sends the session key to what *A* believes to be *B*; but *X* intercepts once again, and is of course able to decode the message since it has been encrypted with *X*'s own key.

In message 7 *X* then begins transmission directly with *A* using the session key requested by *A* for communication with *B*.

In fact, what we see has happened is that communication has been established between *A* and *X*. A believes it is talking to *B*, and *B* knows nothing about the proposed communication that *A* wished to establish with *B*.

A could now find itself in a very difficult position. There will be no independent record that the original request was made for communication with *B*.

If the authentication server keeps records of session key requests then these records will show (c.f. message 4) that *A* (apparently) *requested*

communication with X despite what A might claim and A's own records might (purport to) show.

This demonstrates the importance in the analysis of protocols such as this of considerations such as whether the authentication server does or does not possess state. In the example of the attack discussed above we can see that if the server has state then the initial damage done by the intrusion can be compounded by the server seemingly being able to verify a bogus transaction.

With the addition of the recipient's address in the second message of the protocol, a substitution of the recipient (as shown in message 4) can be detected by A on receipt of that message.

4.4 Denial of Service Attack

There are other types of attack, however, to which the Needham-Schroeder protocol may still be susceptible. With the protocol as specified we observe that the first message is in the clear. This can give rise to the possibility of a denial of service attack.

Let us consider the first message of the protocol; and using the same notation as before:

8. $A \rightarrow [X: A,B,I_{A1}\,;\ X\ \text{Æ}]\ AS: A,X,I_{A1}$

The second message of the protocol will now become:

9. $AS \rightarrow A: \{I_{A1},X,CK,\{CK,A\}_{KX}\}_{KA}$

Upon decrypting this, A will be able to see that the recipient is now stated to be X, i.e. a different principal to that of the original request, namely B; and will be able to deduce that something has happened to the original request.

However, it is still not necessarily possible at this stage, for A to conclude that the original request has been intercepted and that the identity of X has been substituted for that of B.

A might conceive of the possibility that the original message has been corrupted in some manner before being received by *AS*, and that this might have resulted in the mistaken identity of the originator. *A* might also conceive (perhaps more seriously) of the possibility that the substitution has been made by the authentication server itself; either accidentally or even deliberately.

Whatever the interpretation assumed by *A* of why it received a wrong message, it is clear that if message 9 is the only response that *A* receives then the only possibility for communication available to *A* will be with *X*.

If *X* has control of the network it is clearly always possible for the intruder to deny *A* completely the possibility of communications. However, in the attack we have just described, we can see that it is possible for an intruder to use the *properties* of a cryptographic protocol (in this case the Needham-Schroeder Protocol) to deny *A* the possibility of communication with *B*, *without needing to have control of the communications network.*

There are clearly a number of variations on this attack, including that where an intruder substitutes for *A* itself (or simply blocks the first message from *A*) thereby ensuring that *A* is totally denied service of any kind. An intruder can also selectively target principals with whom *A* wishes to communicate, and thus deny access to specific principals or groups of principals by *A* (and only *A*, if this is what is desired).

These attacks could conceivably be avoided by the complete encryption of message 1. (However, it is clear that if *A*'s messages are blocked completely then nothing can prevent denial of service to *A*. Though it is likely to be obvious to *A* if this were to be the case.)

If message 1 were to be encrypted then clearly there is a penalty to paid for this in the time taken to do the encryption; and an even greater burden placed on the authentication server by the requirement for the larger number of decryptions necessary to handle all of the principals it serves.

In addition, complete encryption can cause difficulties for the authentication server in identifying the message originator. This could be done on the basis of a known transmission channel or transmission

time; or even by exhaustive search, though this is clearly not very practical if a large number of principals are involved.

It is sometimes assumed that all encrypted messages are accompanied by enough cleartext information to enable the authentication server to make the appropriate key selection, and that this could apply in the case of the complete encryption of message 1. It is clear from the above discussion of denial of service that *any* information in the clear, and especially that which could potentially identify the sender is also capable of being subverted by an intruder.

Of perhaps more concern to the users of systems employing cryptographic means of secure communications are those networks where the network itself adds information identifying the originator of the message *without the knowledge* of the users. In this case it is possible for the user to erroneously believe that it can protect its identity from intruders by such means as the complete encryption of all messages as discussed above. If the system now adds identifying information to the messages then there is again another opportunity for an intruder to confound the desired communications.

In many systems it is also common for a situation to occur where the user might know that such identifying information is added, but is not on a position to stop such information being added. This topic is one which is worthy of further discussion but is not within the immediate scope of subject of this thesis.

If we return to the discussion of the consequences of an authentication server possessing state, then it is easy to see from discussions similar to those outlined above for the case of an intruder, that it would be possible for the *server itself* to masquerade as any one of the principals.

4.5 Known-plaintext Attack

Whilst Needham & Schroeder caution against regarding their protocols as fully engineered solutions to the overall security problems of particular applications, they also mention that the protocols provide an adequate solution to the authentication problems specified by them.

They go on to note that their protocols would need elaboration to meet a number of other security goals and go on to specifically mention that of "withholding all matching cleartext-ciphertext pairs from an eavesdropper".

It is clearly of importance to understand the problems that can occur from attacks that can be made by intruders in a system. We feel that (especially) within the context of this thesis, it is of considerable importance to understand that attacks can also be made by *legitimate* participants in the system. These participants have the potential to subvert the system in a very serious manner - and at a very different level of trust - using the protocol's own capabilities to achieve this, and often with no possibility of detection.

Within this context, the Needham-Schroeder protocol is vulnerable to a known-plaintext attack that can be made by one principal to discover the key of another principal within the system.

Looking at the first two messages of the protocol we see that A is in a position to make a known-plaintext attack on B's key at the point where the second message is decrypted. The protocol could allow A to accumulate a large number of known plaintext/ciphertext pairs for subsequent analysis of B's key, by requesting many sessions from the authentication server. If A discontinues each session at message 2 then B is never in a position to know what A is attempting.

If the authentication server possessed state then it could be possible to determine that such an attempt had been made by a subsequent analysis of the authentication server's logs. In this case state can help to disclose the attack.

It is clear from this that the various participants in - as well, of course, as the designers and implementers of - this system believe that a large number of elements of the system will behave in quite specific ways in specific circumstances[25] . The correct operation of the system relies on these beliefs being true.

[25] *It is perhaps more correct to say that the participants behave in ways that are consistent with holding these beliefs. In general it is not possible to know what is actually believed, but only to deduce this from what is stated or from observed behaviour. It is always possible, of course, for a participant to believe one thing but to behave as if other beliefs were true, for the purposes of deception.*

Assumptions would appear to be made by participating parties that systems elements also possess specific properties, and to trust them in these regards. Most of these assumptions and beliefs are specified neither by the protocol nor by explicit description of the system.

Within the context of our concept of trust we could state the vulnerability to a known-plaintext attack as described above as: "a principal in the system trusts all other principals in the system not to mount a known-plaintext attack on their key".

4.6 Summary

Discussions like these suggest that when we see a protocol statement such as:

P1. $A \rightarrow B$: {message},

we should perhaps ask the question: "what is the meaning of this statement?"; for we have already observed that it can mean different things to different people. In particular, it is not clear that it has a common and unambiguous meaning to:

i. the protocol designer,
ii. the protocol implementer,
iii. *A*,
iv. *B*,
v. an intruder (*X*)
vi. a (disinterested) third-party observer.

As a simple example:

i. the protocol designer might have intended that the statements should read as "*A* sends a message to *B* which *B* receives";

ii. the protocol implementer may implement this as "*A* sends message to *B*";

iii. *A* may understand the statement as "*A* sends a message to *B* which no one else reads";

iv. *B* might understand the statement to mean that the message sent to *B* is guaranteed to have come from *A*";

v. An intruder such as *X* might believe that the message is encrypted with *A*'s key shared with *B*.

vi. A disinterested third-party might observe that all that is intended is that *A* originates a message which is intended to be transmitted to *B* by some unspecified communication channel.

References

[DS81] D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM* Vol. 24, No. 8, August 1981, pp. 533-536.

[KN93] J.T. Kohl and B. Clifford Neuman. *"The Kerberos Network Authentication Service"*, Internet RFC 1510, M.I.T. Project Athena, Cambridge, Massachusetts, September, 1993.

[LGSN89] T.M.A. Lomas, L. Gong, J.H. Saltzer and R.M. Needham. Reducing Risks from Poorly Chosen Keys. *Proceedings of the 12th. ACM Symposium on Operating Systems Principles*, Litchfield Park, Arizona, December 1989. Published as *ACM Operating Systems Review* Vol. 23, No. 5, pp. 14-18.

[MNSS87] S.P. Miller, B.C. Neuman, J.I.Schiller, J.H. Saltzer. *Section E.2.1: Kerberos Authentication and Authorisation System*, M.I.T. Project Athena, Cambridge, Massachusetts, December 21, 1987.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM* Vol 21, No. 12, December 1978, pp. 993-999.

[OR87] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review* Vol.21, No. 1, January 1987, pp. 8-10.

[S92] G.J. Simmons (ED). Contemporary Cryptology. IEEE Press, New York, 1992, Ch. 4, Appendix F, pp. 257-258.

Chapter 5

Trust and Computer Systems

5.1 Introduction

We have chosen to discuss the concept of trust as it relates to an analysis of the security of computer-based systems. One of our contentions, which we develop below, is that the term is not always used in a consistent manner. We refer to Appendix I for illustration of some of the ways in which the term "trust" is used[26].

We believe that there is sometimes merit in noting the distinction between a formal definition of a term and its somewhat more colloquial usage. It is often the case that the (different) usages relate to different views of the system: for example, the manner in which the system is *defined* to behave by the designer, and the manner in which it is *observed* to behave by the user. In our experience, these viewpoints are seldom co-incident.

We would maintain that a definition seeks to embody a viewpoint. It's importance is not that all must necessarily agree with the particular viewpoint, but that (significantly) different viewpoints should require distinctly different definitions.

The adoption of a particular (and related) set of definitions gives rise to a specific *model* of a system whose description it is sought to provide. It is the relationship of any such model (that is, its value) to perceived experience (that is, perceived "reality"), that is used to judge the ultimate usefulness of the model (that is, the consistency between experience and belief).

We present below two basic and different models of trust as applied to computer-based systems. Although we have observed a recent increase in the discussion of trust and computer systems [e.g. BFL96], the overwhelming deliberations over the past few years have concentrated on the application of the "Orange Book" model to commercial open

[26] *This particular collection is courtesy of Michael Roe.*

distributed computer systems.

We think that the application of the military model of computer security as presented in the "Orange Book" is seldom appropriate to commercial open distributed systems, and we present our arguments below.

5.2 Trusted Computer Systems

Trusted systems were first discussed within the context of the security of military computer systems in the United States; as described in the "*Department of Defense Trusted Computer System Evaluation Criteria*" [DOD85] - the "Orange Book".

This work originated in studies instituted by the US Defence Science Board in October 1967 into computer security safeguards that would protect classified information in remote-access, resource-sharing computer systems. As such it would be reasonable to expect that the considerations resulting from this work should be of considerable relevance to many of today's computing environments.

There is, however, a concentration within military systems on the control of access to information. Within the world at large this singular focus is uncommon, and more general issues of security arise which we note later and which are not particularly well addressed by such a restricted view of what constitutes security in a computer-based system.

We find that the presentation of trust in the "Orange Book", treats trust as being essentially a system-based concept. We also observe that it is implicit in the way the concept is used in the context of the "Orange Book" that there is considered to be just one viewpoint from which trust, and the security of a system, is to be judged.

Within military systems adhering to the "Orange Book" precepts, *trust* is directly associated with access to information, and the associated controls for achieving this. Much work has been done on the design of "trusted" systems and components which implement these concepts, and many feel that the implementation of sophisticated access control systems provides for sufficient security of a computer-based system.

The principles underlying the'Orange Book' have also been widely used in the design of many commercial systems where security is considered to be an issue. We believe, however, that military models of security, with their predominant emphasis on secrecy and access controls, are not always the most suitable basis for the design of an open distributed commercial system.

The 'Orange Book' sets out to deal with security issues relating to "remote-access, resource-sharing computer systems". However, in comparison to commercial systems of similar nature, the military systems are more likely to be closed systems in the way that they are implemented. They will derive from a common design; implementing similar security policies, design rules and control structures.

The "Orange Book" model relies upon there being a well-defined security boundary around the system; with a clear differentiation between what is trusted being inside the security boundary, and what is not trusted being on outside of it.

In a commercial environment, it is often the case that considerations of integrity, reproducibility, verifiability and availability of information are frequently as important as considerations of secrecy and control of access to information that are the paramount concerns to be found in the military environment.

As a case in point, we can observe that concentration on secrecy may result in an environment where concealment of behaviour that can threaten the security of the system can be quite easy to accomplish. It might be argued that in many circumstances, a concentration on openness and verifiability, rather than secrecy, will result in systems that are intrinsically more secure against many of the threats that are found to occur within typical commercial environments.

One of the fundamental motivating factors behind the implementation of open distributed computing systems is the desire to share common resources between different users. It is inevitable that different users will have different requirements and priorities, and it is therefore unavoidable that at the outset of the design of such systems there will be a conflict of interests. The idea of a *common* security boundary in this context could therefore be of only limited value.

It seems to us that the idea that it is the system boundary that defines what is trusted and what is not has little match with the realities of the requirements of the different users, principals and systems components in open distributed computer systems.

Systems which are designed, implemented and operated by separate organisations, are unlikely to have a common, unique boundary, with a single "inside" and "outside". It is therefore likely that the primary source of threat to such systems will arise from the activities of a participant who is, for certain specific purposes, an "insider".

The "enemy within", a participant within the system - though not necessarily inside any specific component of the system, with knowledge of how the system works and where the absence of uniformity of interpretation and control has resulted in weaknesses in the security of the overall system, will be in a strong position to threaten the security of the system.

It is worth examining what is typically taken on trust in current systems: servers of various kinds, shared libraries and software components, shared data bases, communications networks, etc. We observe that different participants will consider different things to be trusted in a distributed system; and most, if not all, of these components will outside the direct control of most users.

Trust, by definition, is not a guarantee. Therefore an approach to understanding trust is also one of assessing risk. This leads us to question whether trust should be a local or a global consideration; and whether it is an objective or a subjective concept.

The "Orange Book" approach is to seek to put all of the security-related aspects of a computer system inside a "Trusted Computer Base' (TCB) whose features are considered to be "operative, correct, tamperproof *under all circumstances*". This is stated to be achieved variously through rigorous analysis, the design and implementation structures, and testing. Systems conforming to the "Orange Book" are considered "trusted".

We question the utility of these concepts being applied to commercial open distributed computer systems. We believe that the contexts to which the "Orange Book" precepts apply differ significantly from those

in which commercial distributed systems are implemented.

The TCB idea of placing all security related components inside a common boundary and then of verifying the "trustworthiness" of this construct does not seem to us to be particularly relevant to the design of open distributed systems; indeed it might well be considered to be the very antithesis of the open distributed concept.

In a distributed system, it is very likely that different states of knowledge will exist in different parts of the system. There will be distinct designs, operators and users, and different assumptions will be made in the the use of protocols and (where it is used) cryptography.

We observe that it is a specific characteristic of an open distributed system is that it will exhibit *independent* failure modes. Different parts of the system can fail in ways that not only are not related to each other, but are not necessarily detectable, for what they are, by other components.

The appropriateness of applying a system designed for one context to a different context, without very careful analysis, must be open to serious objections. There can be large differences in the assumptions underlying the applicability and operation of the two systems; and this can introduce elements of risk, and also give rise to operational behaviour and failure modes, that are not properly understood by the system's implementers, operators and users, either collectively or individually.

This is particularly so in the design of open distributed systems, where distributing what was originally a "closed" system operation can dramatically increase the complexity and risks involved. For example, replicating Trusted Computing Bases (TCBs), and linking them by networks (even ones that are encrypted) does not of necessity result in a system as secure as the original, closed TCB.

This can result in an insecure system being constructed from individually secure components. An illustration of this was given at the 1994 Cambridge Workshop on Security Protocols by Mark Lomas (not yet published) on how the back-to-back use of two different and individually secure protocols can introduce weaknesses that were not to be found in either of the original protocols.

in which commercial distributed systems are implemented.

The TCB idea of placing all security related components inside a common boundary and then of verifying the "trustworthiness" of this construct does not seem to us to be particularly relevant to the design of open distributed systems; indeed it might well be considered to be the very antithesis of the open distributed concept.

In a distributed system, it is very likely that different states of knowledge will exist in different parts of the system. There will be distinct designs, operators and users, and different assumptions will be made in the the use of protocols and (where it is used) cryptography.

We observe that it is a specific characteristic of an open distributed system is that it will exhibit *independent* failure modes. Different parts of the system can fail in ways that not only are not related to each other, but are not necessarily detectable, for what they are, by other components.

The appropriateness of applying a system designed for one context to a different context, without very careful analysis, must be open to serious objections. There can be large differences in the assumptions underlying the applicability and operation of the two systems; and this can introduce elements of risk, and also give rise to operational behaviour and failure modes, that are not properly understood by the system's implementers, operators and users, either collectively or individually.

This is particularly so in the design of open distributed systems, where distributing what was originally a "closed" system operation can dramatically increase the complexity and risks involved. For example, replicating Trusted Computing Bases (TCBs), and linking them by networks (even ones that are encrypted) does not of necessity result in a system as secure as the original, closed TCB.

This can result in an insecure system being constructed from individually secure components. An illustration of this was given at the 1994 Cambridge Workshop on Security Protocols by Mark Lomas (not yet published) on how the back-to-back use of two different and individually secure protocols can introduce weaknesses that were not to be found in either of the original protocols.

We believe that in much of the analysis of the security of distributed systems, the role of the medium - usually a communications network - utilised for interconnection and the interchange of messages between component parts of the system has been largely ignored. The role of the network could be considered to be that of an active system component, or a set of such components; or as an *intermediary* (trusted third party, even), or a set of intermediaries; or even some mixture of the two.

We maintain that whichever consideration is taken, including that of ignoring the role of the network entirely, there will be major, and different, implications in the analysis of the trust relationships of those involved in the message interchanges in the system, and these will change between the various cases.

In contrast to the "Orange Book" we believe that there is value in considering security issues, and trust in particular, from a local, subjective viewpoint rather than from an intrinsic and global system-based viewpoint.

The importance of considering a specific context and viewpoint to the understanding of trust can be illustrated by observing that two participants may trust an implementation of a system, but for different reasons, and because of these differences they will each have different vulnerabilities with respect to the system.

What one participant has to trust the system for can be very different to that which another participant has to trust it for; for example, a service user compared to a service provider. As an example, let us consider the interaction between a bank and an account holder in a transaction involving the withdrawal of cash from an Automatic Teller Machine (ATM).

The objectives from a customer's viewpoint could be stated to be that they do not get charged for the withdrawals of others, and that they do get the cash they ask for when they ask for it.

The objectives from the bank's point of view can be stated as to ensure that it is not defrauded, and that wrong amounts of cash are not given out nor cash given to a wrong person.

We believe that it is therefore inappropriate to use one's trust assessment for the purposes of judging the other's level of risk - as bank customers who have been victims of "phantom" withdrawals from their bank accounts will have found out to their cost.

The inappropriateness of considering trust to be a global property in distributed computer systems was nicely demonstrated in [CL95]. This paper concludes "In real life there is no global trust, and protocols should not be designed which unnecessarily require principals to trust their certification authorities."

The above considerations lead us to conclude that there is merit in considering a different notion of trust to that used in the "Orange Book" and the various systems that seek to implement it.

In the following section we propose a new way of looking at trust which we believe has considerable advantages over the "Orange Book" approach, particularly in the consideration of security issues in open distributed computer systems.

Our approach allows for individual participants to assess their own level of risk. They are in a much stronger position to choose the level of risk that they are prepared to bear.

We show how our approach can be used to examine the vulnerabilities of different participants in these systems, and how this can lead to a greater understanding of the risks that are being assumed.

We also show how these concepts can be applied to sub-systems and component parts of distributed systems, and to stand-alone systems in a way that can allow for an understanding of individual failure modes. We do not believe that this can be done using the "Orange Book" model because the TCB concept of system-wide trust sidesteps the basic ideas required to conduct an analysis at the sub-system level.

5.3 Trust and Knowledge

"Truth" could be considered to be a constant and not subject to change, it is only how it is perceived - or perhaps we should say, how it is *mis*perceived - that changes. To (mis)quote Shakespeare: "Truth is not truth that alters when it alteration finds": though there is a

dependency on the passage of time, since something may once have been true but have now ceased, or will in the future cease, so to be.

We could say that almost the exact opposite might be said of *trust*. It can be argued that the more information that becomes available in a trusted environment, then the more the perception of the original trust could be expected to change[27] . This would mean that the trust will be strengthened, confirmed, weakened or destroyed; but that almost certainly it will be changed in some way.

As we have previously noted, trust is, by definition, not the same as certainty. It has been observed[28] that trust is usually used in situations where "facts" are missing, but are needed in order to complete a process or procedure. Trust can be seen to change as knowledge of the situation changes.

Our considerations lead us to the conclusion that trust is essentially an *epistemic* concept; that it relates to a state of knowledge and *not* to a state of a system [CH96]. We contrast this approach to the more traditional one found in the "Orange Book"[29] .

The concept of trust being "knowledge-based" rather than being "system defined" leads us to believe that it would be useful to consider the idea of trust being *relative* to a base of knowledge.

To illustrate how this could be acceptable at an intuitive level, let us consider the following situation, using a general understanding of the meaning of trust.

I have a financial officer of whom I have had personal knowledge for very many years. I have no doubts about his honesty. I am however aware that he has what is euphemistically called a "drinking problem"; and it is not unknown for the numbers coming from his department following a long lunch hour to need subsequent "updates".

I am therefore be in a position where I do not necessarily question a person's honesty (i.e. I "trust" them, the *person*), yet I do not believe that the numbers I have been given by them are correct (i.e. I do not "trust" them, the *numbers*).

[27] *See later discussion of trustworthiness.*

[28] *This was suggested during a private conversation with Professor R.M. Needham.*

[29] *This is also different from the concept of trust as extolled by Francis Fukuyama in his recent book, "Trust" [F95].*

This example illustrates that to trust someone for one thing - e.g. not *deliberately* falsifying the accounts - is not the same as trusting them for something else, albeit related: in this case, delivering the correct accounts.

We believe that it is important to our understanding of trust that many *different* things may be being trusted in any given circumstance - in the above example these are honesty and competence, respectively. This leads us to consider trust in a different way to that usually presented in discussions of security of computer-based systems; where trust is considered to be a property of the system *per se.*

As a consequence of our line of argument we believe that there is considerable merit in the consideration of trust as being a statement about *relative*[30] , and local, knowledge; in contrast to the more usual usage of trust as an *objective* concept applied on a system-wide basis.

When we examine how the concept of trust has traditionally been applied to computer-based systems, we see that it has not always been used in a common, or even a consistent manner.

Probably the most widespread use of the concept of trust has been in the context of "Trusted Systems" (c.f. "Orange Book"). It has been stated that within the "Orange Book" context, a "trusted" component of a system is regarded as one that is *capable* of violating the security policy of that system.

Trust is thus perceived as a predetermined property relating to the system as such, and discussion of trust is therefore limited to a discussion of the security policy of the system, and the relationship of individual system components to that security policy.

We believe it to be a significant shortcoming in the "Orange Book" that the secure operation of a compliant system is dependent upon the "correct input by administrative personnel of parameters related to security policy"; and where it is not stated what is the relationship of these personnel to the security policy itself, nor to how their actions affect whether a system is to be trusted or not.

[30] *By relative here, we mean relative to a specific local knowledge base - however it may be stated, and whether explicit or only implied.*

What we have is a situation where system-defined criteria for the security of the system are administered by *individuals* who are neither included as components in the definition of the system, nor in the criteria for determining whether the system is to be trusted or not.

We find this concept of trust to be of only limited utility in the analysis of the security vulnerabilities of open distributed computer systems. We contend that it is essential to an understanding of the security properties of a distributed system to include the administrators of the system as an essential part of the system itself.

We would, in fact, go even further than this. We believe that it is essential to also include the designers, implementers and users of any computer-based system in the analysis of its security properties.

We have already shown that there are a number of possible viewpoints from which to judge the security of a system, and we would ask what are the considerations that should determine which viewpoint should be used for the analysis of the system.

It is not at all obvious to us why, of all the viewpoints that are available, the military criteria should choose that of the *system itself* for what is to be trusted and what is not.

Such a consideration would seem to embody the view that it is only the behaviour of the (abstract) hardware and software components of a computer-based system that can in some way be made constant and therefore used as a basis for a (fixed) determination of trust. We do not believe that such a fixed focus is particularly useful in understanding the security weaknesses of commercial open distributed computer systems.

In particular, we note that an *abstract* viewpoint, namely that of the system, has been chosen; rather than, say, that of the security policy formulator, or the policy's administrator. (We use the singular because it is our contention that if more than one administrator is involved, then the application of the security policy will differ between them because they will have different trust sets.)

It is tempting to speculate that the choice of an abstract concept such as that of the computer system to form the base from which trust is to

be judged was made in the belief that it was possible to abstract the concept of trust, embody this concept in the system design, and then (figuratively) to put a wall around this embodiment. Such an approach would seem to imply that a man-made artifice, such as a computer system is to be trusted more than its specifiers, designers, implementers, operators, administrators.

We observe that in many (non-military) situations where the concept of trust is used, it is most commonly used as a substitute for knowledge. As we have previously noted, trust is invoked instead of knowledge in situations where there is the need for a particular piece of knowledge to complete a transaction within the system, but for some reason this knowledge is lacking.

We can obviously conceive of a number of possibilities why such essential knowledge might be lacking. These can include the difficulty and cost of obtaining the knowledge, the length of time it might take to obtain it, the availability of the knowledge, or even its very existence.

We propose, therefore that trust be considered essentially as a statement about the position of one party in a system to other parties or system components. It is quite conceivable for different principals to have different views on the matter of the trust of the same aspect of their common system.

It seems to us, therefore, that the "Orange Book" usage of the terms "trust" and "trusted" are better understood when associated with the related, but different, concepts of *reliance* and trustworthy.

We do not believe the concept of trust as we have presented it, to be an atomic notion, to be treated as a discrete entity which either exists or does not. We show how it is possible to construct a notion of trust from a combination of statements of belief, that follows the intuitive use of the concept.

We demonstrate this by introducing the following little 'calculus'[31] we have formulated, and using it to examine the concepts of trust and reliance.

[31] *With apologies and thanks to Burrows, Abadi and Needham [BAN89], as appropriate.*

It is our intention only take this discussion far enough to demonstrate our conjecture that trust is not an atomic notion, and also how it differs from other, similar, concepts such as reliance.

5.4 A Simple Definition of Trust

We start by presenting a working definition of trust, as it pertains to the principals of a system. In the following, $\mathbb{A}$, $\mathbb{B}$ are principals, and S is a statement of some knowledge. This could be a cryptographic key or the state of a system, part of a system, or a system variable, for example.

Defn: **Trust**

$\mathbb{A}$ *trusts* $\mathbb{B}$ about S means

If

$\mathbb{B}$ says S

then

$\mathbb{A}$ *believes*

$\mathbb{B}$ *believes* S

We write this as :- $\mathbb{A}\ t\ \mathbb{B}$.

We propose that this is different to a statement about belief involving principals, which we see as follows:

Defn: **Belief**

$\mathbb{A}$ *believes* $\mathbb{B}$ about S means

If

$\mathbb{B}$ says S

then

$\mathbb{A}$ *believes* S .

We would like to emphasise that what we are concerned with here are statements that can be made about the principals in a system, and not with statements about knowledge or belief, *per se*.

We would like to demonstrate the use of this calculus by applying it to to an specific example[32]. Let us consider the earlier tale of the accountant and his boss.

We start by noting that there are two different things that are stated as being trusted in this example: the *person*; and the person's *output*. These could be characterised as the person's *honesty* and their *competence*, respectively.

Let us try to formulate what we mean when we say that we "trust the person". This is usually taken to mean that we believe that they are not lying; that is, that *they* believe what they are saying:

$\mathbb{A}$ **trusts** $\mathbb{B}$ **'s *Honesty*:**

If

$\mathbb{B}$ says $\boldsymbol{S}$

then

$\mathbb{A}$ *believes*

$\mathbb{B}$ *believes* $\boldsymbol{S}$.

Which we note is our proposed definition of trust.

In a similar way, a formulation of "trust of output" could be said to mean that we believe *what* they give us, that is:

$\mathbb{A}$ **trusts** $\mathbb{B}$ **'s *Competence*:**

If

$\mathbb{B}$ says $\boldsymbol{S}$

then

$\mathbb{A}$ *believes* $\boldsymbol{S}$.

Which is our definition of "belief".

We can see that what was called "trust" in the two cases can be shown to be two separate and distinct concepts.

[32] *We are indebted to Bruce Christianson for suggesting this approach.*

"Honesty" is related to a state of mind, and it can therefore change; although the state of the system can remain unchanged. "Competence" occurs where it is believed that might be a *fact* involved.

We believe that, without resorting to the realms of philosophy, a meaningful distinction can be made between different usages of the term "trust" and to what that term has been applied. These distinctions can be useful in the analysis of the security properties of a computer-based system as they can be used to illustrate different states of knowledge and perceptions of the principals in the system.

Treatments of trust as a fixed property of a system, such as those contained in the "Orange Book", cannot allow such distinctions to be made.

We believe that the simple concepts presented above are capable of being extended and developed further, however this has been left for further work.

5.5 Further Considerations

One of the ways in which this calculus can be extended is to consider to what extent deductions can be made from observations of the behaviour of principals using these and other, formalisms.

As an example, let us consider our earlier definition of belief, but examine the behaviour from another viewpoint. Observations from this viewpoint could be presented as follows:

If (it is observed that)

$\mathbb{B}$ says $\mathbf{S}$ (to $\mathbb{A}$)

and ($\mathbb{A}$ behaves in a way consistent with)

$\mathbb{A}$ *believes* $\mathbf{S}$

then (to what extent can we say that)

$\mathbb{A}$ *believes* $\mathbb{B}$ (for $\mathbf{S}$).

Further work will be undertaken to expand on this approach; again with the focus of different viewpoints and local knowledge. To the extent that belief is an *internal* state of a principal, then the question arises of to what degree it can ever be deduced from a principal's behaviour.

It is our contention that an analysis of the security properties of a system, that ignores *deceit*; which will include bluffing, lying and misrepresentation, for example; will be unable to identify all of the major vulnerabilities of the system.

The ability to handle deceit is central to the risks facing a computer-based system, its users, and proprietors. It is this capability and perspective that is missing from the "Orange Book", and which we believe to be so important in our approach.

In the case just given it is possible, for example, that $\mathbb{A}$ may know $\boldsymbol{S}$ by some other means than having just been told by $\mathbb{B}$; $\mathbb{A}$ may have told $\mathbb{B}$ $\boldsymbol{S}$, in some, other, unobserved, manner. Indeed, it could be possible that $\mathbb{A}$ and $\mathbb{B}$ are in collusion with each other to establish an *alibi* of some nature.

Clearly, what can be deduced from observations alone may be very restricted and flawed. However, for many of the principals - and others with a stake in a system - *subjective* observation may be the only means available to them to examine the security characteristics of the system as these characteristics affect them.

It is a major difference in our approach to those incorporating the "Orange Book" principles, that such a relativity of view is not only readily accommodated, but is at the very basis of considerations.

We would reiterate that the calculus presented above is intended at this stage to be applied only to relationships between principals. It cannot be applied, as it stands, to relationships between principals and statements *per se*.

By way of illustration we would point out that although it might look to some as if our definition of belief between principals could be considered to be transitive in nature, this could only be the case if

statements such as A **believes** S and A **says** S were *equivalent*. Clearly, such a proposition does not allow for the analysis and understanding of duplicity and misrepresentation by a principal.

We therefore maintain that careful consideration is given to the distinctions already made concerning the application of these various concepts to "active" components of a system (e.g., principals), and "passive" components of a system such as cryptographic keys.

It is just as obvious from our presentation of our concept of trust that as proposed, it is not transitive; and we believe that the precise nature of security attributes that are professed to exhibit the property of transitivity need very careful examination.

Intuitively, notions of trust are usually concerned with actions, i.e., "behaviour", and notions of belief are more normally associated with information, i.e., "facts".

We believe that there has been some confusion between the concepts of trust and "reliance", just as there has been between the concepts of trust and belief.

Reliance is not the same as trust. Something is relied on if it is *necessary* for the completion of an activity. Reliance can involve both "facts" (=belief) - the value of a particular piece of information may be necessary; and "behaviour" (=trust) - the particular actions of a principal may be necessary.

We note that the concept of reliance, although apparently composite, exhibits the property of *transitivity*. For if A relies on B for S and B relies on C for S, then A relies on C for S. This is true irrespective of whether A agrees to B's reliance on C, or is even aware of it. It seems to us that this is the notion being considered when transitivity of trust is proposed.

Other related terms are also to be found in the literature, such as "trustworthy", "authority", "speaks for" and "jurisdiction", though not all of these terms are well-defined.

The BAN logic does not use the concept of trust; instead it uses the concept of "jurisdiction", defined as follows:

Jurisdiction:

if

$\mathbb{A}$ *believes* $\mathbb{B}$ has ***jurisdiction*** over $\boldsymbol{S}$

then

if

$\mathbb{A}$ believes ($\mathbb{B}$ *believes* $\boldsymbol{S}$)

then

$\mathbb{A}$ *believes* $\boldsymbol{S}$.

We observe that this definition contains a mixture of statements about "competence" ($\mathbb{B}$ has **jurisdiction** over $\boldsymbol{S}$) and "honesty" ($\mathbb{B}$ *believes* $\boldsymbol{S}$).

It is still not clear to us the precise manner in which these different contexts are compounded. We would also note the similarity of the second clause ($\mathbb{A}$ believes ($\mathbb{B}$ *believes* $\boldsymbol{S}$)) to our definition of trust.

We would point out that the concept of jurisdiction as defined is not sufficient to replace the concept of trust as we have presented it, since of itself the concept of jurisdiction does not allow for the analysis of corrupt, malign, *or incompetent*, principals within a system.

We observe again that many of the established methods of analysing the security properties of a system assume that the system has been installed, and is being operated, according to overall design principles that will *require* some level of internal system integrity. It will therefore be the case that any failure of the system to adhere to these integrity assumptions, *for whatever reasons*, are not capable of being detected within this framework.

We would like to return for one final reflection on the relationship of transitivity and trust.

In a recent publication ([CB94], p. 54), Cheswick and Bellovin make the statement that "Transitive trust may also be an issue" when referring to a situation of interconnected computers *independently* extending their (so called) trust relationships unbeknown to other computers with which it is interconnected.

They present a situation where $\mathbb{A}$ trusts $\mathbb{B}$ for something and unbeknown to $\mathbb{A}$, $\mathbb{B}$ trusts $\mathbb{C}$ for (part of)this. $\mathbb{A}$ is then said to be trusting $\mathbb{C}$, without $\mathbb{A}$'s approval or knowledge.

As we have covered above we believe that this is to confuse trust with other, and distinct, concepts: either reliance, or, perhaps in this case, delegation; depending upon the extent of specific knowledge and consent. There are clearly major issues regarding trust, and the evaluation of risk, in those situations where delegation occurs.

There is an ostensible relationship between delegation and trust, and in many situations it is clear that we delegate to those we trust; but it is also clear that the converse is sometimes also true, and we "place trust" in something or somebody is a statement of effective delegation.

What is not always so clear is the nature of what is being delegated. Sometimes it is the role (i.e., Departmental Manager), and sometimes it is the authority (i.e., approve invoices up to £xxx amount).

We believe that the topic of delegation deserves a significant study in its own right, and in particular the relationship of the roles of the delegator and, for want of a better word, the delegate. The person doing the delegating and the person receiving the delegated powers will have different trust sets, that is they will trust different things in different ways.

The situation surrounding delegation will be compounded where there is any form of delegation occurring that is not known to the (putative) delegator. *Hidden* delegation can present major problems and risks for the principals in a system; not the least of which is the determination of when it is occurring.

We would point out at this stage that we believe there are implications of this research for ongoing research in other areas, for example we think that analogies can be made to the case of conventional computer systems and to the relative situations of system designer and system user, who could be said to have "delegated" the design of the system Our experience shows that the two parties rarely have identical sets of belief and trust.

In a similar way a user running of a piece of software could be considered to be delegating a task to the system when the programme is run. There a clearly many opportunities for hidden delegation to be occurring within the confines of an computer system and its system software.

A similar argument can be made for the process of communicating data from one system to another, where the nature of the communications network can clearly add some complexity to the analysis of what is being trusted and by whom.

Another topic that can be linked to the concepts of trust and delegation is that of "trusted third parties". We think that the discussions of trust and delegation above indicate the topic of trusted third parties as being a more complicated subject than is sometimes presented.

We believe that the examination of what is involved in behaviour such as "trusting a key-server to deliver a unique and secure key", using our approach, will point to many areas where the underlying assumptions have not previously been considered relevant to the security analysis.

There are many situations where third parties are being used, and inevitably trusted, without the knowledge or conscious awareness of the user. We would argue that in many distributed systems the network itself is often viewed as "invisible", and we believe that it should be treated as a trusted third party in any security analysis. In general, we have not found this to be the case.

5.6 Summary

We think that our approach to trust as representing a statement of relative knowledge, compared to the more usual use of the concept as a property of a system, allows for the analysis of the security properties of participants and components *within* the systems boundaries.

Our approach to associating risk with knowledge, rather than with the design of the system, allows us to handle in a consistent manner the actual operational characteristics of a system, and to allow for different principals having different perspectives of the system.

The "Orange Book" approach to the concept of trust imputes responsibilities to participants in the system that are necessary for the correct functioning of the system. In contrast it might be said that our approach could be likened to "every one for them self". We believe that our view is particularly important for the analysis of corrupt, malign, or even incompetent, principals or servers, for example.

Systematic comparisons of a number of different viewpoints can lead to a much better understanding of the system vulnerabilities, and can allow different principals to choose their own levels of trust in relationship to their knowledge base; and therefore to have some control over the levels of risk that they are prepared to tolerate.

References

[BAN89] Michael Burrows, Martín Abadi, and Roger Needham. *A Logic of Authentication*. DEC SRC Research Report 39, February 28, 1989

[BFL96] Matt Blaze, Joan Feigenbaum and Jack Lacey. Decentralised Trust Management. *Proceedings of the IEEE Conference on Security and Privacy*, Oakland, Ca., May 1996.

[CB94] W.R. Cheswick and S.M. Bellovin. *Firewalls and Internet Security*. Addison-Wesley 1994, p. 54.

[CH96] B. Christianson and W.S. Harbison. Why Isn't Trust Transitive? *Security Protocols*, Lecture Notes in Computer Science 1189. Springer 1996, pp. 171-176.

[CL95] B. Christianson and M.R. Low. Key-spoofing attacks on nested signature blocks. *IEEE Electronics Letters* Vol.31, No. 13, 1995, pp. 1043.

[DOD85] Department of Defence Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, US Department of Defence, Washington, DC, USA, December 1985.

[F95] Francis Fukuyama. *Trust*. Penguin Books 1996.

"For now we see through a glass, darkly; but then face to face.". I Corinthians ch.13, v. 11

Chapter 6

DISTRIBUTED SYSTEMS, SHARED DATA AND DELEGATION

6.0 Introduction

Distributed systems combine potentially large numbers of independent components comprising hardware, software and communications into an apparently single operating unit. Services are provided by mechanisms whose internal workings are usually (deliberately) hidden from the end users of the system, by means of abstract interfaces, protocols and procedures. Indeed, it is commonly held that one of the key design objectives of a distributed system is to provide what is referred to as *"transparency"*[33] [ANSA89,ISO92]. We shall see that the implementation of this concept can lead to serious and unforeseen consequences for users of the system at all levels.

Transparency is a consequence - though by no means an inevitable consequence - of the *separation*[34] of the component parts of a distributed system. Separation of components is an inherent attribute of a distributed system [CDK94], and it is this separation that provides for the parallel execution of programmes and for the multiple concurrent access to resources and data.

It is the separation of components that allows a distributed system to be expanded and contracted as required - both physically and functionally - without disruption to the operation of the system as a whole. This is also often associated with the concept of "openness"[35].

While not being a prerequisite for a distributed system, openness allows for a more general sharing of resources in a way that can be

[33] *Transparency is defined as the the "concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components". [CDK94]*

[34] *We will restrict our discussion here to physical separation, but the arguments can easily be extended to include logical and functional separation.*

[35] *We use the term open here in the International Standards Organisation's (ISO) reference model for Open Systems Interconnection (OSI) sense [ISO81]. The term is also used in the sense of an open operating system[LS79] which deal mainly with the concept of minimal operating system functions (e.g. lightweight kernels), which are not specifically the subject of this dissertation.*

independent of a particular operating system or specific computer hardware.

The ISO Reference Model for Open Systems Interconnection [ISO81, ISO92] has been developed as a standard conceptual framework for the definition of communication protocols that would meet the requirements of an open system; and it has become increasingly common for distributed systems to be designed specifically in an hierarchical and layered way that provides for both the logical and physical separation of functionality.

A consequence of this design approach is that a number of common interfaces will occur at different levels within a system, from that of low-level physical and logical components to that of large-scale application programmes running on multiple (and often different) hardware platforms.

These interfaces are provided to allow *transparent* access to specified levels of the system without the need to have knowledge of other parts of the system. We shall see that major, and unforeseen, consequences can result from the separation of the components of a system, and this can happen at many, if not all, of the levels within the system.

A distributed system may include components (operating at the same or at different levels) that have been provided by, or are operated by, disparate organisational entities. These entities may be part of the user's organisation, but also they may be completely separate legally and organisationally.

In either case, responsibility for the correct working of the system is likely to be split between different organisations, and this can lead to conflicts in the management and control of the overall system.

The separation of the components of an open distributed system coupled with possibly different ownership, implementation, and control can lead to inconsistent results in operation, testing and fault-finding.

Fragmentation of supply can also give rise to inconsistencies which can result from incomplete or inaccurate interfacing between disparately provided services. There is also the possibility of conflicting overlaps in the provision of services, with ensuing unpredictable of outcome.

The chances of this occurring with a small, well-managed in-house system, under single system management control, is likely to be quite small. However, within large-scale systems where functional and operational capabilities span many separate areas of management, geography and control, attempts to maintain consistency at one level of the system can conflict with parallel activities at other levels and by other, overlapping, functions.

Some form of overall system management is needed in order to avoid these types of problem and to furnish a common set of user interfaces and services while at the same time maintaining consistency of the system's *behaviour* as seen by the user.

The provision of facilities expected in distributed system such as fault tolerance and isolation, dynamic reconfiguration, component recovery and the enforcement of security and protection regimes will require the system management functions to have capabilities for the control of communications channels and for the selective isolation of users and components throughout the system .

For system management controls to operate effectively, they will need autonomous communications capabilities that are separate to normal user service functions, and which can provide independent communications channels (both physical and logical) that are not accessible, or known, to other parts of the system.

We can see how in an open system, the requirements for maintaining overall system integrity and fault tolerance can be used to justify an approach that hides the existence and detailed operation of sub-systems and components from other sub-systems, components, end-users, and even the systems managers of other operational domains.

Unfortunately, the ability to subvert such a system, with its multiple levels and interfaces, is greatly enhanced by being able to hide what is going on within the system, thereby constraining the knowledge of users of what activities are occurring across the system.

In the following sections of this chapter we look at the consequences of using a distributed system for resource sharing. We use the example of a commercially available system to illustrate the potential risks to users in systems of this type, and how the principles of transparency

and openness can undermine system integrity and lead to users being vulnerable to many forms of attack without their knowledge or ability to avoid.

6.1 Distributed Systems and Shared Resources

In a distributed system with a client-server architecture, a server is generally considered to be a component of the system that manages a particular type of resource [CDK94], be it a physical resource such as a printer, or a logical resource such as a file. A physical server may also sometimes have more than one logical function, such as being both a file server and a boot server.

The concept of a resource is abstract and somewhat arbitrary but can be considered to be any object which can be allocated within the system [TB74]. Such objects can encompass a wide range of computer components including hardware items such as printers, processors and disc drives as well as software and system components such as programmes and processes, files and data bases, and mixed hardware (firmware) / software functions such as system and remote booting.

Services provided by such a distributed system must be able to reconcile the different requirements imposed by the need to operate with multiple clients "simultaneously". There will be a requirement that there be a minimum of interference between the separate operations performed on behalf of different clients (or even multiple invocations from the same client), whilst at the same time being able to retain the necessary underlying ability to share the resource or data; and also to maintain overall integrity over the resource.

With multiple clients accessing a common server, the possibilities of conflicts and inconsistencies are many. Different techniques have been devised in order to minimise or eliminate such conflicts as they occur with different types of resource.

In the following discussion we will concentrate on techniques that are in widespread use for the avoidance of inconsistencies of data items shared by clients on a common server: though we note that these techniques do share many attributes with those used for other types of resource.

6.2 Shared Resources and Concurrency Control

A server performs a series of operations on behalf of a number of clients, and does so in a manner that seeks to preserve the integrity of the data involved, whilst still allowing the maximum concurrency of access to data items consistent with this.

The mechanisms most usually found in controlling concurrent access to data by different client operations involve the application of read and write locks to the data items. These locks can usually be applied at a number of different levels, from the highest logical and physical levels, to that of an individual record or field, depending on the type of application and data involved.

The use of different levels (the *granularity*) of locking, can give rise to significant differences in the observed performance characteristics of a system, both at the individual server and also the client application; and the manner in which conflicting demands from client processes are balanced, can result in unequal treatment of different processes, with the possibility of certain users being advantaged or disadvantaged compared to others.

The server will usually balance conflicting locking demands using a number of means directly available to it, such as suspending processes, queuing processes in one of a number of ways, or even by terminating them and requiring that they be restarted.

The undetected manipulation of locking mechanisms in a distributed system is possible as a consequence of the implementation of the *transparency* design principle. Undetected tampering can also take place as a consequence of layering in open systems, where each layer represents a specified level of abstraction, and uses services provided by lower layers utilising this abstract interface (see for example [B93]).

Hardware resources are seldom shared in the sense that several processes are simultaneously accessing the same device. Rather, the resource is said to be *shared* when exclusive control is given to each process for only a particular interval of time.

The allocation of a resource in this way will clearly slow down the service for any given process, but it does allows the *utilisation* of the

resource to be that much greater than a totally dedicated resource.

In order to allow this sharing of resource whilst at the same time meeting some overall system goals for performance, consistency and integrity there must be some form of overall system control of the resources involved.

The synchronisation of concurrent access to shared resources in a client-server system is performed by processes that run on the servers. These processes control the management of locking schemes and mechanisms and are referred to collectively as *lock management* services.

In the following section we look at a specific example of the implementation of a lock manager in an existing and well-documented distributed system. We examine some of the ramifications of this particular implementation, and the potential impact the design decisions can have on the operation and integrity of the system as a whole.

6.3 A Distributed Lock Manager

Digital Equipment Corporation's VAXcluster™ environment incorporating the VMS operating system is regarded by many as an exemplary implementation of a distributed system. Its design objectives include, as well as the generic capabilities of a distributed system, the characteristic that members of a cluster can boot and fail in an manner which is independent of each other component of the system.

This environment consists of a highly integrated organisation of computer systems whose members share resources, queues and disk storage under a single security and management domain. Although most cluster resources may be shared, user processes and system memory are node specific, and failures to the node will require the processes to be recreated by the user on possibly another node in the cluster.

This description, and those following come from the VMS System Management Manuals [DEC88]. These documents are intended to

guide VMS systems managers through the processes and procedures necessary to set up and tailor a cluster operating environment. There are a number of software components used to implement the cluster functions, but we restrict our initial attention to the operation of the VMS Lock Management System Services.

The *Distributed Lock Manager* provides facilities enabling the support of system wide synchronisation functions for control of access to shared resources. This is accomplished by the use of support services that implement the locking and unlocking of resource *names*, and the provision of queuing mechanisms.

The resources controlled by the lock management system can be any entity recognised by the VMS operating system (for example, files, data structures, databases, and executable routines). We shall concentrate our discussions on data resources, and in particular file locking.

The first point of note is that the lock manager is only effective between *cooperating* processes. The name that is specified by a process represents the resource that is being locked. "Other processes that need to access the resource must refer to it using the same name."

It is stated in the VMS System Management Manual covering Lock Management Services that "The correlation between the name and the resource is a convention agreed upon by the cooperating processes".

From this it would appear that the system is relying on agreements between user processes rather than on formal and enforceable policy regimes. If two processes were to refer to the same (or part of the same) data structure using different names, then the lock management processes are likely to be ineffective with regard to that data.

No system-wide mechanism for enforcing names is presented, nor any system-wide facility for data management that might provide for or support the provision of unique names for data items. Even the use of the lock manager itself would appear to be entirely at the discretion of the user processes themselves.

It is possible for individual System Managers to seek to enforce a set of standards and conventions when setting up processes for which they are directly responsible or over which they have some form of direct

control. It is unlikely however, that all accesses to shared data by processes within a distributed system can be foreseen, and enforcement of access controls to data will not be possible without the explicit and precise cooperation of all processes involved[36].

Even allowing for the goodwill and cooperation of the various users that share specific systems resources, it is still quite possible that different processes will have been established at different times and by possibly different parties, and that different models and assumptions will have been used which could result in the inadvertent bypassing of important system conventions and controls, and thereby affect shared data in ways that are both unknown and unanticipated by other users.

The accidental and unplanned bypassing of systems conventions and controls is always a potential problem in systems that have more than one author and whose operations have been constructed and modified over a period of time.

The lack of mandatory enforcement mechanisms is a serious omission, however, for the moment let us assume the case of a well ordered and centrally managed system, where we might feel fairly confident in the presumption that all processes are well-behaved and will always use agreed systems conventions and services.

We might initially feel more confident that such an implementation could be considered secure against the problems associated with deliberate manipulation of the system to the benefit of one or other of the parties involved. With tampering unable to take place, or at least not in an undetected manner.

In the next section we look at ways in which typical locking mechanisms[37] are implemented. We shall see that not only are there opportunities for deception and manipulation of data at a number of levels within a distributed system; but tampering can take place in ways which can be concealed from users.

Such tampering can be done in ways that are unverifiable by users, and even made to appear to have been done by parties other than the perpetrator, including users themselves.

---

[36] *"Many database administrators know that application security can be bypassed, but keep quit for an easy life". Neil Hutton [CS97].*

[37] *A comprehensive account of file locking techniques can be found in [CDK94].*

6.4 Distributed Systems and Delegation

One observation we can already make from what we have seen above is that significant portions of the operation of a distributed system lie quite specifically outside of the ability of a user to control, monitor or even verify.

Of particular interest to us in these considerations is the *integrity*[38] of the data within a distributed system. In practice, this data is held on behalf of a client by a server, and the only way usually open for the client to access and verify the data is to invoke one or more of the server's operations; operations over which the client has no direct control.

The client has effectively handed over whatever independent capabilities it might have had regarding the data, and has, in essence, *delegated* its responsibilities to the server.

It is therefore a consequence of using shared resources in a distributed system that control is being delegated, often unintentionally, and this will result in users and their applications having to *rely* on the servers within the system for the availability and integrity of those resources, whether they wish to or not.

If we consider the implementation of locking mechanisms for shared data, as discussed earlier, then we might conclude that these processes will operate only in favour of the server.

This should not be taken to imply that any ultimate benefit to be gained from the specific application of locking and queuing mechanisms will necessarily accrue to the server; but rather that the server is in a unique position to enable such benefit, however that may be measured.

We can envisage situations (for example, in the areas of currency and securities trading) where the selective operation of locks could be used to disadvantage one user over another. There could be serious financial implications for a trader unable to access the current, real time, value of a share or currency exchange rate, or who is given as a current value one which has been subsequently updated.

[38] *Integrity is used in the meaning that only authorised modifications can be made, and only by parties authorised to make them [P89]. There is also the issue of the modifications being made correctly.*

If, in addition to the user being given inaccurate or untimely data, the situation is deliberately hidden by the server and is unknown to the user, then the impact of any unfavourable effects could be greatly magnified. Even in those situations where a user is aware of something being wrong with the system, as could happen in the extreme case of denial of access, their ability to protect against possible adverse effects will be very limited.

The above considerations lead us to ask the question: *"On whose behalf does a server operate?"*. We have seen that it is not difficult for the selective use of read and write locks within a server to advantage or disadvantage different clients of that server. This could especially be the case when users are in direct competition with each other, while at the same time they are ("cooperatively") sharing a common data base.

This raises the issue of precisely how a client should treat a server. In looking back at our discussion of a trusted message server in Chapter 3 we can see that there are a number of roles that could be considered:

1. An *extension*[39] of the client and its associated processes;
2. An *independent* [40] executor of instructions.
3. A *hired*[41] third party.
4. An *agent*[42] of a second party.
5. An *independent intermediary*[43].

We have not specifically referred to the notion of a "trusted third party" in our examples. This is because we do not find there to be an exclusive meaning to this concept. Examination of the above shows that they are all in effect "third parties", and they each will be "trusted" to some extent or another, depending on the way in which they are used.

---

[39] *This is the easiest case to consider, since the objectives and controls of the server should be coincident with those of the client(s), and therefore verifiable as such.*

[40] *It is always important to ask of what or whom something is independent. In establishing independence we need to know who or what is defining or guaranteeing such an attribute, and how it might be verified and monitored. Is an independent entity to be considered a part of the system or to stand outside of it. Without complete and unambiguous specification, any discretionary capabilities, whether exercised or not, will be unclear* and unknown.

[41] *We have in mind here a facility that is specifically available to the user to undertake a given set of tasks or responsibilities directly for a specific user, and to an agreed performance level as a consequence of a commercial agreement between the two. In such an arrangement it is important that there is a clear understanding of what is being contracted for: the server or the service, what is individual and what is shared, etc. Without this, a true understanding of what is being trusted and relied on cannot be obtained.*

[42] *An agent can be considered to be a representative of a party. It usually has responsibility for conducting negotiations and agreements on behalf of its principal. However, the exact nature of the legal relationship between the two is usually secret, and therefore the precise relationship with an agent, and the capacity in which it is operating can be obscure.*

[43] *This is used to denote an entity that, unlike those above, has an equal relationship to both the user and server.*

We have seen that the operation of a client-server relationship requires the client to delegate responsibilities, *de facto*, to the server. In order to understand more fully the implications of delegation, whether it has been implemented knowingly or otherwise, we feel that it is important to establish more detail about certain aspects of the relationship[44], for example:

i) how a delegation relationship is established;
ii) how long it is to last;
iii) how operation of the relationship is monitored;
iv) how the relationship is made known to others.

As a deeper understanding is gained of the role delegation plays within a distributed system, more detailed abstractions that are more closely fitted to specific situations will be proposed.

All we need say at this stage is that there is clearly a relationship between delegation and trust; and that in many situations delegation can be considered to be the "reverse" side of trust.

As of writing we are considering the following working form for the representation of delegation:

I [give/transfer/allow use of], [uniquely/irrevocably/until/unless/ for a period/for a task],
my [authority/power/knowledge]
to [someone/something/(ones/things)]
under [these conditions: .....]

We believe that there is considerable scope for development of the concepts introduced above. Further work will yield additional research results, and a deeper understanding of the workings and failings of distributed systems designs. This work is, however, beyond the scope of this dissertation.

[44] *For an example of how some authors have treated delegation refer to [GD90].*

6.5 Transparency and Trust

As already noted, distributed systems are motivated by a requirement to share resources of one kind or another. Different operational and design models will be used for the various operations of these systems, such as communications, data storage and processing capabilities. Also, different trust models will apply to different parts of a system as viewed by other, dependent, parts of the system.

We have seen that within a client-server system, the only way for a client to access data items stored on a server, is by invoking one of the server's operations. We note that, in general, a server will be operating on behalf of a number of client processes, each possibly unaware of the others' existence, and whose requests for service will probably be interleaved.

To maximise concurrency, the server will have to *serialise* [P79, BG81] access to the data items, and will do this using its available mechanisms of resource locking and transaction queuing. It will even terminate process transactions in some instances, such as when deadlocks [H72] occur.

It is clear that the server has primary control over the shared data, and is in a position to effect its contents at any one time, as well as being able to control the order in which operations are allowed to change the data. The manner in which a server has been configured, and many of the algorithms by which it operates will not be visible to users of the system.

We have seen that transparency is used as a guiding principle in the design of distributed systems to provide a consistent and unified view of the system to its users. The implications of implementing this concept of transparency are quite far reaching, and from some viewpoints could be considered to have the potential for an extremely harmful influence on the design and behaviour of a distributed system.

In the context of our discussions on servers we might conclude that the application of the principle of transparency is likely to result in mechanisms that hide how the system has been implemented, and the way in which it has been designed to work.

Hiding what a system actually does is probably the exact opposite of what a typical user would expect transparency to mean. It is much more likely that they will take it to denote complete visibility of what and how something is happening.

It is understandable that an application programmer, wishing to concentrate on the functional details of an application, might welcome a common and "transparent" system interface with which to communicate. However, unforeseen consequences can result from lack of knowledge of how a system is configured and behaving at a given point in time.

In many systems, especially those which are transaction based, applications, as well as data, will be shared. Ignorance of the details of a system's operations can have a major impact on the understanding of timing and performance issues of an application.

Many assumptions will be made about the environment within which an application is presumed to operate. Some of these assumptions will be correct, others could be completely unwarranted, yet determination of which is which can be (made to be) very difficult.

We have seen how the implementation of transparency can provide an environment whereby designers are able to conceal possibly *hostile* components and activities from users and applications. Such activities can include, for example: eavesdropping, replay of messages, impersonation of selected users and selective denial of service.

These risks, which result from the decision to implement a distributed system in a transparent manner, are not always clearly understood by users and application programmers (amongst others). The perceived relationship to the system and its components can be quite different to the actual one, resulting in undeterminable effects particularly with regard to inconsistencies and failures in the operation of an application.

Undetected manipulation of the system by legitimate members of the shared community can occur, in a manner similar to that described in Chapter 4. Unfortunately, as we have already seen, the ability to conceal the nature of such attacks is also an integral attribute of the implementation of the principle of transparency.

It is in such unforeseen ways that designs whose primary aim is to provide integrity at the level of the overall system can be in conflict with the integrity at the level of the individual user. Flexibility at the system level can override a user's requirements for independent and local reassurance, verification and validation.

6.6 Summary

In previous chapters we have shown that by introducing a number of different conceptual *viewpoints* from which to view the operation of a system, we can highlight dependencies of one part of a system on other (possibly unknown) parts.

The process of establishing what is actually known and what is being "taken on *trust*" from a particular viewpoint, allows for qualitative judgments to be made about the system that can represent an "individualisation" of risk as seen from that point.

Application of the concept of transparency in the design of distributed systems, as discussed earlier, makes qualitative assessment of individual risk difficult to achieve. Only one viewpoint can be presented, apart from the "standard" user defined by the interfaces, namely that of the central system as assumed to be implemented. The correctness of this viewpoint cannot be checked, even *post hoc*, since this is exactly what transparency prevents[45].

We recall that transparency allows, or more correctly requires, the actual workings of the implementation of the system to be concealed from users, and we can conclude that transparency is not the same as integrity or accountability.

In summary, given the way in which distributed systems are designed, we conclude that in the presence of transparency:

1. There is no defence against insider attack:
   the system may be corrupt and the user cannot prove it;
   the user cannot differentiate between accident and malice.

[45] *"It is desirable that the global infrastructure and other higher level infrastructures (the supporting physical and logical environment layers of a system) should conceal the details of transparency mechanisms from objects operating within the service environment layers". From the ANSA reference manual [ANSA87].*

2. The converse is also true:
   the basic system can be innocent of bad behaviour, but a user may be corrupt and the system cannot prove its own innocence.

3. Logging and audit rely on locking services that can be subverted and therefore cannot be used as a reliable record of events, no matter how good the locking service specification.

4. The use of replication relies on the correct implementation of even more complicated protocols (such as Byzantine agreement [LSP82]), and therefore cannot be verified either.

We have seen when analysing different operational possibilities in the comparatively simple case of the message server discussed in Chapter 5, that the complexity of situations and outcomes can escalate very quickly. Our conclusions establish the necessity for analysing specific distributed system implementations from various viewpoints, with representative combinations of systems components, users, designers, public interest, etc. This is, however, beyond the scope of this dissertation.

References

[ANSA87] ANSA Reference Manual, Release 00.03. Advanced Network Systems Architecture, Hills Road, Cambridge, United Kingdom, *ANSA Project*, 1987.

[ANSA89] ANSA (1989). The Advanced Network System Architecture (ANSA) Reference Manual. Castle Hill, Cambridge, England: *Architecture Project Management*, 1989.

[BG81] P.A. Bernstein and N. Goodman. Concurrency Control in Distributed Database Systems. *ACM Computing Surveys*, Vol 13, No.2, 1981, pp. 185-222.

[CDK94] G. Coulouris, J. Dollimore and T. Kindberg. Distributed Systems Concepts and Designs. *Addison-Wesley* 1994.

[CS97] Neil Hutton. Security? What Security? *Client/Server Magazine*. Reed Business Publishing. Jan/Feb 1997.

[DEC88] VMS System Management Manuals, AA-LAxxA-TE. *Digital Equipment Corporation*, Maynard, Massachusetts April 1988.

[GD90] Morrie Gasser and Ellen McDermott. An Architecture for Practical Delegation in a Distributed System. IEEE 1990.

[H72] Richard C. Holt. Some Deadlock Properties of Computer Systems. *Computing Surveys*, Vol. 4, No. 3, September 1972.

[ISO81] ISO-7498 (1981). ISO Open Systems Interconnection, Basic Reference Model *International Standards Organisation*, 1981.

[ISO92] International Standards Organisation (1992). Basic Reference Model of Open Distributed Processing, Part 1: Overview and guide to use. ISO/IEC JTC1/SC212/WG7 CD10746-1, *International Standards Organisation*, 1992.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, Vol. 4, July 1982, pp. 382-401.

[LS79] B.Lampson and R.F.Sproull. An open operating system for a single user machine. In *ACM 7th. Symposium on Operating Systems Principles*, December 1979.

[P79]] C.H. Papadimitriou. The Serialisability of Concurrent Database Updates. *Journal of the ACM*, Vol. 26, N0. 4, 1979, pp. 631-635.

[P89] C.P.Pfleeger. Security in Computing. *Prentice-Hall International, Inc.*, 1989.

[TB74] D.C. Tsichritzis and P.A. Bernstein. Operating Systems. *Academic Press* 1974.

"Never ascribe to malice that which can adequately be explained by stupidity". Unknown

Chapter 7

Conclusions

7.0 Synopsis

The thesis of this dissertation is that there is no such thing as a computer system. Or to be more specific, we find that we are unable to agree with the concept of "a computer system" as being something monolithic that can be represented by a single conceptual model, which is internally and externally consistent, and which behaves as a whole in a uniform and predictable way, under all foreseen circumstances.

There may be little disagreement with a statement such as this put in this way, yet we find in practice that it is just such a view which is usually applied. The assumption that a system is a single entity about which global statements can be made (such as "the system is secure or " the system works") is unfortunately all too common.

The treatment of a collection of (usually) co-operating entities as a single, uniform and consistent whole is misleading. It is to imply that there is, or can be, a *single valid* point of observation from which all activity can be seen to be deterministic.

Different components of a system - which should be taken to include designers, users, programmers and system managers, as well as clients, servers, operating systems, applications programmes, software and hardware sub-systems and networks will all need to be considered.

They will all have different views of what they perceive to be the system and its associated behaviour. In a very real sense, the behaviour of a system will be relative to the viewpoint of a particular component *at a particular time.*

We need to be able to reason about large systems, and not just about their components. For this we would like to have conceptual tools that will help us to understand the behaviour of these systems, and to help us make sense of other, possibly conflicting, views.

We have sought to indicate the need for a new methodology that will allow us to better identify and understand those areas of possible conflict or lack of knowledge, and we have looked for ways to improve the design of computer-based systems in a practical manner that can be readily understood and applied.

We propose that anyone planning the design of a system, or part of a system, should look at it from the point of view of *each* of the participants, and that this should include all of the components - including users and implementers - to see what they are relying on and to make sure that these assumptions are compatible.

We believe that a more detailed characterisation of the various participants in a computer system, and of their roles and underlying assumptions, will lead us to a better understanding and identification of those areas where these assumptions rely on undefined capabilities, or are at odds with one other or are capable of being understood (and therefore implemented) in more than one way.

To this end we have introduced the concept of "trust", and defined it in a way that enables *qualitative* judgments to be made. The additional concepts of "reliance" and "delegation" are introduced, and highlight the dependency of one part of a system on other parts of the system, including those that are unknown and possibly even unknowable.

We conclude that trust is best considered as a *localised* concept used as a substitute for knowledge, and that considering a computer system to be a *discrete* entity with global characteristics can be dangerous and damaging to the "health" of computer operations and their users.

7.1 Summary

The principal goal of our work has been to identify new systems engineering approaches that can help in the design and construction of computer systems that more closely reflect the expectations of users. In particular, we have examined ways in which the design of computer systems could be improved by a systematic approach to the identification and reconciliation of the many assumptions that are held by the different parties with a stake in the system.

Experience leaves us with the abiding impression that the goals, objectives, concepts and assumptions of the various parties who are involved in a computer system (i.e., systems designer, programmer, operator and user) are seldom, if ever, coincident[46].

We frequently hear today, criticism of many computer-based systems that the needs of users in particular are inadequately realised, or even ignored. Many aspects of computer systems such as reliability, availability, functionality - sometimes too much, as well as too little - and ease of use have been under attack by both users and operators; and also by members of the public who, although not necessarily directly involved with the system can often be seriously affected by its operation [CWC].

We have sought to identify and examine ways in which the design and engineering of distributed computer systems can be improved, particularly with regard to their users. The predictability of operating characteristics from the point of view of a user has been a particular concern.

We have looked at ways in which *users* of a system can survive system faults. Significant effort has gone into the design of "fault tolerant" systems (see for example, [C91], [CDK94], [M93] and [A97]), but exactly what is being tolerated, and by whom or what, is not always apparent.

We would like to be able to understand the behaviour of large systems in totality and not just their components, and we have sought to establish a conceptual framework for reasoning about the integrity of computer systems and their component parts. We have used this conceptualisation to gain insight into the nature of possible computer systems failure modes and to examine ways for systems to survive certain classes of failures.

We do not believe, however, that there is a unique point from which it is universally valid to judge the adequacy of a system. Whilst it is probably most common to find that the system *itself* has been used as just such a point (c.f.H80), the behaviour and security of a system depends upon the point of view from which it is being judged.

[46] *John Shore's article "Why I never met a programmer I could trust" [S88] highlights many of the issues in a thought-provoking, somewhat amusing, way.*

We have looked at ways to identify and evaluate risks from specific and different perspectives rather than from that of an abstract and idealised concept of a system as a whole. We have been looking for ways to change our view of what is happening from that of the system "looking out" to that of the user " looking in".

We have sought to analyse why the actual ways in which systems behave differ from users' expectations: i.e. why users are surprised when and how the system they are using fails for other than a straightforward hardware breakdown; and have examined the role that trust plays in users' expectations.

We have shown that what can be deduced from observation of the behaviour of a system depends upon the the point of view of the observer; and we have developed a concept of *viewpoints* that assists in the understanding of expectations of different participants. We have shown that the implementation of large systems will almost inevitably result in conflicts between different viewpoints.

Attempts to conceal the workings of a system from users - such as transparency and layering - do not allow for the development of different viewpoints. The diversity of the different participants is effectively collapsed into the single position of an idealised system as conceived by the designers.

To give focus to our investigations, we began by considering the security aspects of a distributed system and its components. In the context of a distributed computer system, security is not just about protection from invasive elements or the divulging of secrets: it is at least as much about correctness of operation and consistency of observed behaviour.

We have found that a lack of clear and agreed definitions of many security concepts has hindered coherent and consistent discourse of many aspects of system failure. We have introduced some new concepts in an attempt to clarify some of the important issues.

Traditionally, security of computer systems has been based on considerations that have originated in military models of thinking [e.g. BL73]. This model is primarily concerned with *secrecy* of information and the prevention of unauthorised disclosure.

This is clearly, in itself, not sufficient to maintain the security of a system, and a counterpart model dealing with the integrity of information was subsequently proposed by Biba [B77]. Together these two models have provided the basis of the U.S. Department of Defense Trusted Computer System Evaluation standard [DOD85]; also known as the "Orange Book" because of the colour of its covers.

The principles underlying the "Orange Book" have been widely used in the design of many commercial systems where security is considered to be an issue. However, military models of security with their predominant emphasis on secrecy and access controls, are not always the most suitable basis for the design of an open distributed commercial system.

Attempts have been made to separate the issues involved between military and commercial requirements, and an example of this can be found in [CW87], where Clark and Wilson set out to present a security policy for data integrity that was based on commercial data processing practices.

In a commercial environment we observe that integrity, reproducibility, verifiability and availability of information are frequently more important considerations than the major concerns of secrecy and access control found in military settings.

In particular, we note that concentration on secrecy could result in an environment where concealment of behaviour that threatens the security of the system is quite easy to accomplish. It can be argued that in many circumstances, openness and verifiability result in systems that are intrinsically more secure against many of the typical threats to be found within commercial environments.

The "Orange Book" sets out to deal with "remote-access, resource-sharing computer systems". However, in comparison to commercial systems, military systems are most likely to be closed systems rather than open in their implementation. The model relies upon there being a security boundary around the system. with a clear differentiation between what is inside and therefore trusted, and what outside and therefore not trusted[47].

[47] *We have previously noted the Canadian effort to extend security criteria to a wider range of products [CSSC93].*

One of the fundamental motivating factors behind the concept of open distributed computing systems is the desire to share common resources between different users. It is inevitable that different users will have different requirements and priorities, and it is therefore unavoidable that at the outset of the design of such systems there will be a conflict of interests.

When we consider the design of commercial open distributed computer systems, we are considering systems whose commonality lies in their starting point of some standard or definition of the system to be produced. It seems inevitable that different designers and implementers will produce systems that will sometimes behave differently in similar circumstances as a result of their *individual* interpretations of a system's specifications[48].

Systems which are designed, implemented and operated by separate organisations, are unlikely to have a common and unique boundary, with a simple "inside" and "outside". It is therefore understandable that the primary source of threat to such systems can arise from the activities of a participant who is in some respect an "insider".

The "enemy within", a participant of the system with knowledge of how it works, and where absence of uniform and consistent interpretation and control has resulted in weaknesses in the overall security of the system, will be in a strong position to threaten it.

In addition, absence of common control structures can result in failures of independent, component parts of the system going unrecognised or being misinterpreted by other parts of the system. Although this is not the same class of threat as the putative "enemy within", the rogue activities of such "insiders" can have effects on the system that are arguably of even more serious consequence.

We have already noted that systems designed in one context can be implemented in another. Whenever this is done, careful evaluation of an appropriate threat model is necessary to ensure that the design assumptions of the system being implemented match those of the system to which they are being applied.

[48] *The task involved is not to be underestimated. Bell et all [BBD77] reported that the requirements document for a ballistic missile defence system contained over 8000 distinct requirements and was 2500 pages long.*

We have also seen the importance of ensuring that all of the assumptions underlying the design of a system are clearly documented. Without this the implementers are unlikely to realise that such assumptions have been made; which can result in unpredictable behaviour of the system that is difficult to trace and analyse.

The appropriateness of applying a system designed for one context to a different context, without very careful analysis, must be open to serious questioning. There can be large differences in the assumptions underlying the applicability and operation of the two systems. This can introduce unforeseen elements of risk and give rise to operational behaviour and failure modes that are not properly understood by the system's designers, implementers, operators and users, collectively or individually[49].

This can be particularly the case in the design of distributed systems, where incorporating what was originally a "closed" system into an open and distributed operation can dramatically increase the complexity of the system and the associated risks. For example, replicating trusted computing bases (TCBs), and linking them by networks (even ones that are encrypted) is likely to result in a system that is less secure than the original (see Figure 1)[50].

When we look at Figure 1 we note that in the open system implementation of this model that the TCB has dual roles, sometimes acting as a TCB and sometimes acting as a user. Indeed, we can see from this diagram that it could be acting in both roles simultaneously. We can also see that a TCB could be acting in one role towards one TCB, and in an entirely different role towards another.

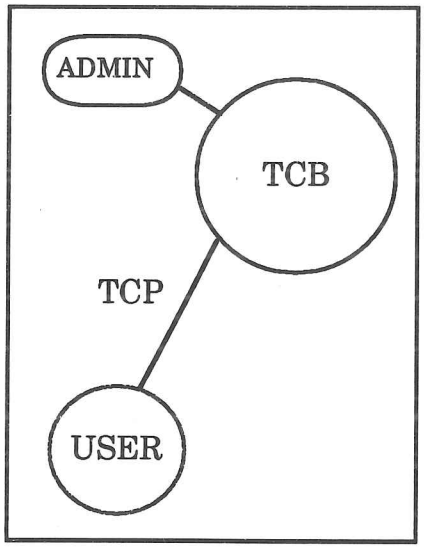
We note that the concept of a *Trusted Path* is an essential component of the "Orange Book" trusted system. The interconnections between replicated TCB's in an open system version of a trusted system will not be *Trusted Path*'s as understood in the "Orange Book" context, and the integrity of the originally conceived secure system will already have been compromised.

---

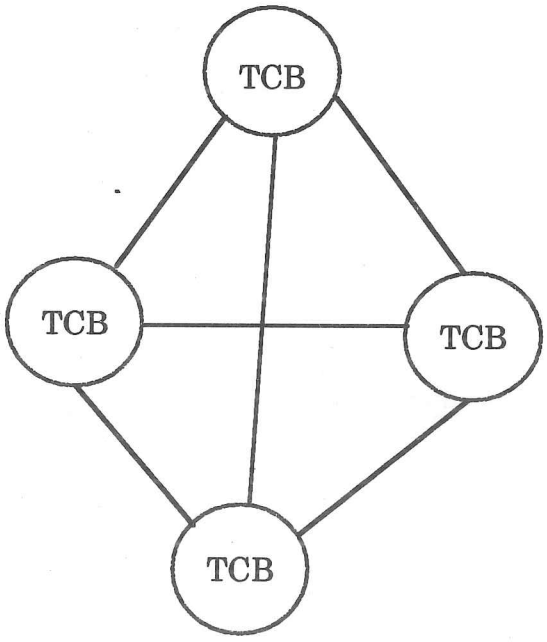
[49] *Sommerville notes in his book [S89] that "It is very difficult to formulate a definitive specification for large software systems. Thus, it should be assumed that initial system requirements will be both incomplete and inconsistent."" How much more so, then, will this be the case in an open distributed system.*

[50] *In this example the secure operation of the trusted system depends upon the link between the Trusted Computing Base (TCB) and the User being a Trusted Path (TCP). A trusted Path is defined in the "Orange Book" as "A mechanism by which a person at a terminal can communicate directly with the Trusted Computing Base and cannot be imitated by untrusted software".*

"ORANGE BOOK"



Trusted System



General Open Distributed System

Figure 1.

We have deliberately not complicated this diagram by trying to represent the "real" users of such a distributed system.

To help us to focus our discussion we propose the following definitions:

> A *distributed* system consists of a collection of processors, together with their ancillary peripherals, systems and applications software, connected by a communications network[51] .
>
> An *open* distributed system is one where the interconnection between the different levels and components of the system is achieved through implementations that are not necessarily identical, but which are designed to a common set of specified interfaces.

Let us consider, as an example, the complexity of the security problems that might be involved in an open distributed computer systems concerned with the management of the storage and retrieval of multi-media data. Where text documents, images, video sequences, maps and other complex objects are stored in independent databases, that may be physically separated, and which are interconnected by a network of communications links.

In a typical open distributed system such as this, it is very likely that different states of knowledge will exist in different parts of the system. There will be distinct designs, operators and users, and different assumptions will be made in the the use of protocols and (where it is used) cryptography.

It is not at all clear to us that a direct analogy can be made in cases such as these to the more simple model presented in the "Orange Book". Going even further, we would maintain that representing complex systems such as these by the simple replication of the "Orange Book" model will only mislead in any analysis of the systems security properties.

We have separately demonstrated that the proper operation of cryptographic protocols depends upon their context and environment. The use of a protocol outside of its specified (or implicitly specified) environment can lead to unanticipated and undesirable behaviour.

[51] *In some discussions the network is included as part of the distributed system.*

Some protocols are specific about the environment in which they are designed or presumed to operate (for example, Needham-Shroeder's trusted principals), and these assumptions are explicitly stated. Attacks on these protocols often are successful because the protocols have been implemented in in an environment that in some way violates or differs from that presumed by the designers of the protocol.

It is sometimes obvious what *some* of the designers' assumptions are: some are made quite explicit, while others have been made implicitly obvious, for example, by reference to another protocol they are seeking to improve. However, there frequently are others that are not stated, or even acknowledged, and in these cases the discovery of various attacks on the protocols illustrates the constraints, (and also some of the *hidden* assumptions) that need to be considered if the protocol is to be successfully implemented.

We have also shown that the outcome of a particular protocol (successful or not) will usually depend upon information or criteria that are not part of the protocol itself.

This raises the issue of how far it is desirable to go, outside of the specific description of the message interchange itself, in order to ensure that the protocol can terminate in a *known* outcome or state. It seems to us that many protocols appear to be designed on the basis that they *always* work; and that if something adverse happens - a wrong key, wrong message or reply, for example - then the protocol may just terminate in some indeterminate state. It is not clear to us that this is a reasonable approach to take.

All of which can result in an insecure system being constructed from individually secure components. An illustration of this was given at the 1994 Cambridge Workshop on Security Protocols by Mark Lomas (not yet published) on how the back-to-back use of two different and individually secure protocols can introduce weaknesses that were not to be found in either of the original protocol.

We should point out that we are not here seeking to introduce the notion of composability, and its application to the security properties of a system - for a discussion of this topic and some of the issues involved, we refer you to the session on Composition at 8th. IEEE Computer Security Foundations Workshop [CSFW8].

Our deliberations could be seen as being orthogonal to those being worked on in the area of Composition, where attention is more focussed on issues of properties that may be *invariant* between systems. We are concerned with considerations of a *subjective* nature: it is a consequence of our line of argument that there is considerable merit in looking at trust as being associated with *relative*, and local, knowledge; in contrast to the more usual usage of trust as an objective concept applied on a global basis.

Clearly the considerations of the two approaches are related, but what we seek to illustrate here are the local consequences of using a system component, considered to be secure in one environment, but which is not secure when implemented in another, different, environment.

We observe that a specific characteristic of an open distributed system is that it will exhibit *independent* failure modes[52]. Different parts of the system can fail in ways that not only are not related to each other, but are not necessarily *detectable*, for what they are, by other components.

The importance of *context* and *viewpoint* to the understanding of trust can be illustrated by observing that two participants may trust an implementation of a system, but for different reasons, and because of these differences they will each have different vulnerabilities with respect to the system.

What participant A has to trust the system for can be very different to that which participant B has to trust it for; for example, a service user compared to a service provider. It is therefore inappropriate to use one's trust assessment for the purposes of judging the other's level of risk - as bank customers who have been victims of "phantom" withdrawals from their bank accounts will have found out to their cost.

This can be considered another example of where from one viewpoint it is the system that is trusted but not the user, and from the other viewpoint it is the user who is trusted but not the system. Other viewpoints exist where, for example, neither is trusted.

[52] *We are also aware that the occurrence of "common mode" failures in complex systems can also cause unpredictable failures that are difficult to isolate [C97].*

Military systems attempt to define security boundaries where those components that are inside are considered "trusted" and those outside the boundary are not to be considered "trusted". Using the "Orange Book" concept of trust something is *trusted* if it can violate the security policy of the system; that is, if it exists within the security boundary of the system. We have noted that in the commercial environment, it is much more likely that a threat to the security of the system will come from within rather than from outside.

In the "Orange Book", trust is treated essentially as an intrinsic, system-based concept. It is also implicit in its usage that there is just one viewpoint from which trust, and the security of a system, is to be judged. We have shown that this usage has somewhat limited utility when applied to commercial open distributed systems.

In contrast to the standard use of trust as a property of a system, our notion of trust applies only within the context of a specific *viewpoint* from which to judge risks. We argue that it is only after the introduction of a specific context from which trust is to be judged, that we can understand many of the *intrinsic* vulnerabilities of a system.

We have introduced a concept of trust that is directly related to *individual participants* in the system and to their lack of knowledge about it. In this regard, then, we propose that trust is therefore not to be treated as a property of the system at all, but as an attribute of individual participants. We have shown how this approach can be used to identify and measure the risks associated with the system for any particular participant.

We have proposed that we consider trust to be related to what we have not directly verified and therefore what we do not know. Our understanding of trust is as a measure of the current limitations of our knowledge. Trust can thus be seen as a substitute for *knowledge.*

When this has not been explicitly realised, then this can lead to things being trusted that perhaps should not be so treated. Also we note that within our considerations, trust is not to be seen as a constant, but something that can change as knowledge changes.

We have introduced the concept of there being more than one viewpoint from which to describe the behaviour of a system, and therefore the trust relationships that pertain. The utility of this concept lies in its ability to enable the nature of the risks associated with a specific participant to be measured, whether these are explicitly recognised and accepted by them, or not.

All systems are of necessity, trusted in some way. We propose that our *goal* should be not to have to trust a system; we argue that we should attempt to systematically replace *trust* with *knowledge* in those areas that are of a critical nature to us. Our goal should be that we should try to reach the position where "we know (or can verify that) it works" replaces "we trust it".

In those situations where we are consciously aware of trusting something, we should endeavour to identify mechanisms that will *validate* its behaviour.

The extent to which we will be able to pursue this goal, and the associated costs of this approach, will vary with the anticipated costs of failure in the system. The level of trust we are prepared to assume is thus directly related to the level of risk that we are prepared to accept, using our notion of trust.

Our approach is not restricted to particular participants - for example, system and user - but can be extended to any component, module and sub-system that has well-defined interfaces. This allows trust models for individual parts of the system to be considered, along with those of the suppliers and users of the system.

This enables us to analyse a system from a number of different perspectives, and therefore to build a more complete understanding of the nature of the assumptions - many of which may never have been explicitly stated, recognised, or even understood - which underlie its design and operational behaviour. Different assumptions result from different levels of trust. If trust models are not identical then different risks will be being assumed[53].

[53] *As software engineers, we tend to neglect models. In other scientific disciplines, models act to unify and explain, placing apparently disjoint events in a larger, more understandable framework. The lack of models in software engineering is symptomatic of a much larger problem: a lack of systems focus. Few software engineers understand the need to define a system boundary or explain how one system interacts with another [PJCK97].*

Our treatment of trust as being associated with specific viewpoints, to be seen as not a system-wide property but rather as an attribute of a participant in a system (whether this be user, operator, designer or even a sub-system or component part), and which can differ for different participants and also at different times for the same participant, enables us to examine, for example, the vulnerabilities of one part of a system to failures in another part. This could be then be extended to the *detection* of suspicious behaviour of either a system component or system user.

We believe our analysis to have shown the limitations of applying even relatively simple security models outside of a well-understood and pre-defined environment; and the risks that can occur in applying them outside of the context in which they were designed.

This has led us to propose a method for codifying and comparing the risks inherent in any computer-based system, and how this can enable the level of risk *at any particular level* in the system to be understood and managed *at that level.*

We have shown that it is insufficient to take the viewpoint of just one of the involved parties as the basis for defining a system's behaviour, and to judge the system from only that particular viewpoint (the one usually taken being that of the designers of the system). It is essential to any comprehensive understanding of the system that the separate roles of the implementer and the user are also included; with the assumptions about the system from their viewpoints being documented and taken fully into account in any analysis of the system's behaviour. These assumptions can then be examined, and verified, in detail, and compared for ambiguities and conflicts.

We think that far too many assumptions are made that are *implicit*, and understood by only one of the parties. Examples of this are: "I expect the ATM to give me my money, and not to give it to anyone else"; "I expect the system not to give out money in a way that would make the bank liable to lose money"; "I expect the compiler to compile my code accurately and efficiently, and not to add any code of its own" (c.f. Ken Thompson's Turing Award Lecture [T84]).

Each role, and each participant in that role, can be expected to have assumptions about the operation and behaviour of the system from their own viewpoint. Our experience has shown that these sets of assumptions are unlikely to coincide, and are more likely to conflict in major ways in many areas of critical importance to the participants.

We argue that each party should understand what it is that it relies on from other parties and parts of the system, and that its "trust set" should be explicitly stated and compared with that of the other relevant participants at whatever particular level the analysis is being undertaken.

We maintain that not only will this highlight the "hidden" assumptions of one party on other parts of the system, but it also enables the identification of those areas where the assumptions do not match, or where there are major conflicts. We believe that it is only from a study of the behaviour of each of the relevant parties that a thorough understanding can be attained of what each party may be *implicitly* assuming.

We believe that the adoption of our approach will allow for the identification of the extent and type of assumptions that each component in the system is making and relying on for the correct operation of the system from its viewpoint, and for the detection of incompatible and conflicting goals.

We are proposing that *trust* be considered as a relative concept, and that it is not the result of knowledge but a substitute for it, and that adopting our viewpoint is fruitful for the analysis of security risks in computer-based systems.

We hope that our approach will give a new meaning to the words "trusting the computer" - and will result in fewer examples of the inappropriate application of a design from one context being applied to a different and unsuited context. We would like to see computer systems "trusted" less and "known" more.

7.2 Future Work

Our investigations have pointed to some exciting areas of future research. We believe that the further development of conceptual models based upon our ideas can lead to better understanding of the reasons why computer systems fail in the way that they do[54]. This can lead to improvements in the design of distributed systems that will allow the viewpoints of the users and operators of these systems to assume a more prominent place in the design considerations.

The concepts of trust, reliance and delegation are capable of further development and applicability. The development of a rigorous method by which the behaviour of a computer system can be evaluated from different perspectives and at different levels within the system, as well as from without, will provide a powerful tool for systems engineers in the future.

The application of the ideas we have developed to specific computer-based services such as trusted and untrusted third parties, electronic notary services and even networks themselves could lead to greater understanding of the local component of computer operations This in turn can help us to understand the relationship of the computer system to the environment in which it is required to operate.

We would like to see the tying together of the computer system itself, the operational environment and the policies by which the system including the users are to conform.

References

[A97] Algirdas Avizienis. Toward Systematic Design of Fault-Tolerant Systems. *IEEE Computer* Vol. 30, N0. 4, April 1997, pp. 51-58.

[B77] K. Biba. Integrity considerations for Secure Computer Systems. *US Air Force Electronic Systems Division*, 1977.

[54] *We note established interest in related areas such as the analysis of software design faults [G86].*

[BBD77] T.E. Bell, D.C. Bixler and M.E. Dyer. An extendible approach to computer aided software requirements engineering. *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, 1977, pp. 49-60.

[BL73] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. *MITRE Report MTR 2547*, v2, Nov. 1973.

[C91] F. Cristian. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM* Vol. 34, No. 2, February 1991.

[C97] Richard I. Cook. Observations on RISKS and Risks. *Communications of the ACM* Vol. 40, No. 3, March 1997, p. 122.

[CDK94] G. Coulouris, J. Dollimore and T. Kindberg. Distributed Systems Concepts and Designs. *Addison-Wesley* 1994.

[CSSC93] The Canadian Trusted Computer Product Evaluation Criteria. Version 3.0e. Canadian System Security Centre, Communications Security Establishment, Government of Canada, 1993.

[CW87] D. D. Clark and D. R. Wilson. A comparison of Commercial and Military Computer Security Policies. Proceedings of the 1987*IEEE Symposium on Security and Privacy* Oakland, Ca., 1987, pp. 184-194.

[CWC] Almost any issue of Computer Weekly or Computing.

[DOD85] Department of Defence Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, US Department of Defence, Washington, DC, USA, December 1985.

[G86] J. Gray. Why Do Computers Stop and What Can be Done About it? *Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems*, Computer Society Press, Los Alamitos, Ca., 1986, pp. 13-17.

[H80] K.L.Heninger. Specifying software requirements for complex systems. New techniques and their applications. *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 1, 1980, pp. 2-13.

[M93] S. Mullender (Ed). Distributed Systems, Second Edition. *Addison-Wesley* 1993.

[PJCK97] S.L. Pfleeger, R. Jeffery, B. Curtis and B. Kitchenham. Status Report on Software Measurement. *IEEE Software* Vol. 14, No. 2, March/April 1997, pp. 33-43.

[S88] John Shore. Why I never met a programmer I could Trust. *Communications of the ACM* Vol 31, No. 4, April 1988, pp. 372-375.

[S89] Ian Sommerville. Software Engineering, Third Edition. *Addison-Wesley* 1989.

[T84] Ken Thompson. Reflections on Trusting Trust. *Communications of the ACM* Vol 27, No. 8, August 1984, pp. 761-763.

"A definition is the enclosing a wilderness of idea within a wall of words." Samuel Butler. 1912

Appendix I

Trust

From the Pocket Oxford Dictionary:

> *"trust.* 1. n. Firm belief that a person or thing may be relied upon, state of being relied upon."
>
> *"take on* trust: accept as true &c without testing"
>
> "in *a position of* trust: having duties that can be neglected without immediate detection"

From ISO 9594-8:1993:

> "trust: Generally, an entity can be said to 'trust' a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects. This trust may apply only for some specific function. The key role of trust in the authentication framework is to describe the relationship between an authenticating entity and a certification authority; an authenticating entity shall be certain that it can trust the certification authority to create only valid and reliable certificates."

From CD 10181-1.2:

> *"trust:* a relationship between two elements, a set of activities and a security policy in which element x trusts element y if and only if x has confidence that y will behave in a well defined way (with respect to the activities) that does not violate the given security policy."
>
> "A security sub domain element can be told by a security super domain security authority to trust elements of other security domains."

"Trust is based on assurance which can be obtained either by something elements of security domains are told or by something they know."

From section 26.24, part I of the ODP Reference Model:

"Trust is a relationship between agents where agent A delegates agent B to carry out certain roles under rules determined by A and agreed by B."

From the Orange Book:

"*Trusted Computing System*: A system that employs sufficient hardware and software integrity measures to allow its use for processing simultaneously a range of sensitive or classified information"

"*Trusted Computing Base:* The totality of protection mechanisms within a computer system - including hardware, firmware and software - the combination of which is responsible for enforcing a security policy."

From the ITSEC:

The term "trust" is conspicuous by its absence in the ITSEC.

"Some word that teems with hidden meaning - like Basingstoke." W. S. Gilbert. Ruddigore, 1887

GLOSSARY

We briefly define some of the terms that are used in this dissertation.

Authenticate. "1. To invest (a thing) with authority; to render authoritative." [OED71]

Authority. "4. Power to influence the conduct and actions of others; personal or practical influence." *ibid.*

Confidence. "6. The confiding of private or secret matters to another." *ibid.*

Confidential. "4. Enjoying the confidence of another person; entrusted with secrets; charged with secret service." *ibid.*

Confidentiality. "Confidential quality; state of being confidential." *ibid.*

Covert Channels. A programme that transmits information about itself, the data it operates on or other aspects of the computer system to unauthorised individuals.

Cryptography. The writing of things in a secret or disguised manner. This is usually done in order to hide data from unauthorised view. It is also used in the provision of 'digital signatures' and message authentication.

Denial of Service. The prevention of legitimate access to a computer system.

DES. The *Data Encryption Standard* (DES) was developed for the government of the USA for use by the general public for sensitive information. It is based on an algorithm developed by IBM known as *Lucifer* and now more properly as *DEA* (Data Encryption Algorithm). It uses a combination of substitutions and permutations applied for a repeated number of cycles, usually 16, to encrypt plaintext using a 64-bit key into 64-bit blocks.

Digital Signature. The digital analogue(!) of a written signature.

Integrity. "**2**. The condition of not being marred or violated; unimpaired or uncorrupted condition; original perfect state; soundness." *ibid.*

Non-repudiation. The attribute that enables a message to be proved to have originated from the sender it purports to come from.

Privacy. The property of preventing the unauthorised extraction of data.

Public Key System. A cryptographic system which relies on a separate key for encipherment and decipherment respectively: one of which is publicly disclosed; the other of which is held secret by the user.

RSA. RSA encryption is named after its three inventors: Rivest, Shamir and Adelman. It is a public key algorithm which uses two keys, one for encryption, the other for decryption, that work in symmetric pairs. It is based on the difficulty of factoring the product of two large prime numbers.

Secrecy. "**1**. The quality of being secret or of not revealing secrets; the action, practice or habit of keeping things secret.." [OED71].

Secret. "**A.** *adj.* **1.** Kept from knowledge or observation; hidden, concealed. **a**. Predicatively (esp. in *to keep secret*): Kept from public knowledge, or from the knowledge of persons specified; not allowed to be known, or only by selected persons." *ibid.*

Security. It is perhaps interesting to note that the the term *secure* originally had a different meaning that that in common use today. The origin of the word is from the Latin and means 'without care'. The Oxford English Dictionary [OED71] notes the original usage and meaning:

"**A.** *adj.*

I. Feeling no care or apprehension.

1. Without care, careless; free from care apprehension or anxiety, or alarm; over-confident.

In early instances often contrasted with *safe*.

1641 QUARLES Enchir, iv. lxiii, (1654) T 1, ... The way to be safe is never to bee secure."

The current meaning is:

"**II.** Having or affording ground for confidence; safe; (objectively) certain." *ibid.*

It is perhaps ironic that many of todays systems are probably secure in this original sense rather that the more modern meaning.

Symmetric Key System. One in which the participants share a secret key known only to themselves Also known as a ***shared-key*** or ***private key***.

Trap Door. A concealed entry-point to software in a computer system.

Trojan Horse. Software that does one thing (usually destructive) while appearing to do another.

Virus. A segment of self-replicating code that attaches itself to application programmes or to other executable system components. These code segments move from programme to programme and machine to machine. They can replicate an indefinite number of times. [MH89].

References

[LBMC94] A Taxonomy of Computer Program Security Flaws. *ACM Computing Surveys* Vol. 26, No. 3, September 1994, pp.211-254.

[MH89] J. McAfee and C. Haynes. Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System, *St. Martin's Press*, New York 1989.

[OECD92] Guidelines for the Security of Information Systems. *Organisation for Economic Co-operation and Development. Paris 1992. OCDE/GD(92)190.*

[OED71] Oxford English Dictionary, *Oxford University Press* 1971.

Bibliography

[A77] M.W. Alford. A requirements engineering methodology for real time processing requirements. *IEEETransactions on Software Engineering*, Vol. SE-3, No. 1, 1977, pp. 60-69.

[A97] Algirdas Avizienis. Toward Systematic Design of Fault-Tolerant Systems. *IEEE Computer* Vol. 30, N0. 4, April 1997, pp. 51-58.

[ANSA87] ANSA Reference Manual, Release 00.03. Advanced Network Systems Architecture, Hills Road, Cambridge, United Kingdom, *ANSA Project*, 1987.

[ANSA89] ANSA (1989). The Advanced Network System Architecture (ANSA) Reference Manual. CastleHill, Cambridge, England: *Architecture Project Management*.

[B77] K.Biba. Integrity considerations for Secure Computer Systems. *US Air Force Electronic Systems Division*, 1977.

[BAN89] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. *DEC SRC Research Report 39*, February 28, 1989

[BBD77] T.E. Bell, D.C. Bixler and M.E. Dyer. An extendible approach to computer aided software requirements engineering. *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, 1977, pp. 49-60.

[BFL96] Matt Blaze, Joan Feigenbaum and Jack Lacey. Decentralised Trust Management. *Proceedings of the IEEE Conference on Security and Privacy*, Oakland, Ca., May 1996.

[BG81] P.A. Bernstein and N. Goodman. Concurrency Control in Distributed Database Systems. *ACM Computing Surveys*, Vol 13, No.2, 1981, pp. 185-222.

[BGM85] A. Borgida, S. Greenspan and J. Mylopoulos. Knowledge representation as a basis for requirements specification. *IEEE Computer*, Vol. 18, No. 4, 1985, pp. 82-101.

[BL73] D. Bell and L. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. *MITRE Report MTR 2547*, v2, Nov. 1973.

[C91] F. Cristian. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM*, Vol. 34, No. 2, February 1991.

[C97] Richard I. Cook. Observations on RISKS and Risks. *Communications of the ACM* Vol. 40, No. 3, March 1997, p. 122.

[CB94] W.R. Cheswick and S.M. Bellovin. Firewalls and Internet Security. *Addison-Wesley* 1994, p. 54.

[CDK94] G. Coulouris, J. Dollimore and T. Kindberg. Distributed Systems Concepts and Designs. *Addison-Wesley* 1994.

[CH96] B. Christianson and W.S. Harbison. Why Isn't Trust Transitive? *Security Protocols*, Lecture Notes in Computer Science 1189. Springer 1996, pp. 171-176.

[CL95] B. Christianson and M.R. Low. Key-spoofing attacks on nested signature blocks. *IEEE Electronics Letters* Vol.31, No. 13, 1995, pp. 1043.

[CS97] Neil Hutton. Security? What Security? *Client/Server Magazine*. Reed Business Publishing. Jan/Feb 1997.

[CSFW8] *8th. IEEE Computer Security Foundations Workshop.*

[CSSC93] The Canadian Trusted Computer Product Evaluation Criteria. Version 3.0e. Canadian System Security Centre, Communications Security Establishment, Government of Canada, 1993.

[CW87] D. D. Clark and D. R. Wilson. A comparison of Commercial and Military Computer Security Policies. Proceedings of the 1987 *IEEE Symposium on Security and Privacy,* Oakland, Ca., 1987, pp. 184-194.

[D90] A.M.Davis. Software Requirements, Analysis and Specification. *Prentice-Hall* 1990.

[DEC88] VMS System Management Manuals, AA-LAxxA-TE. *Digital Equipment Corporation*, Maynard, Massachusetts, April 1988.

[DH76] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, November 1976, pp. 644-654.

[DM88] T. DeMarco. Structured Analysis and System Specification. *Yourdon Press*, New York 1988.

[DOD85] Department of Defence Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, US Department of Defence, Washington, DC, USA, December 1985.

[DS81] D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM* Vol. 24, No. 8, August 1981, pp. 533-536.

[H72] Richard C. Holt. Some Deadlock Properties of Computer Systems. *Computing Surveys*, Vol. 4, No. 3, September 1972.

[F733] Benjamin Franklin . *Poor Richard's Almanac,* 1733.

[F95] Francis Fukuyama. *Trust.* Penguin Books 1996.

[G86] J. Gray. Why Do Computers Stop and What Can be Done About it? *Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems,* Computer Society Press, Los Alamitos, Ca., 1986, pp. 13-17.

[GD90] Morrie Gasser and Ellen McDermott. An Architecture for Practical Delegation in a Distributed System. IEEE 1990.

[GM86] Narain Gehani and Andrew McGettrick (eds.). Software Specification Techniques. *Addison-Wesley* 1986.

[H80] K.L.Heninger. Specifying software requirements for complex systems. New techniques and their applications. *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 1, 1980, pp. 2-13.

[IEEE85] *IEEE Computer*, Vol. 18, No. 4, April 1985.

[ISO81] ISO-7498 (1981). ISO Open Systems Interconnection, Basic Reference Model *International Standards Organisation*, 1981.

[ISO92] International Standards Organisation (1992). Basic Reference Model of Open Distributed Processing, Part 1: Overview and guide to use. ISO/IEC JTC1/SC212/WG7 CD 10746-1, *International Standards Organisation*, 1992.

[JT79] Randall W. Jensen and Charles C. Tonies. Software Engineering. *Prentice-Hall*, Englewood Cliffs, NJ. 1979.

[K87] Neil Koblitz. A Course in Number Theory and Cryptography. *Graduate Texts in Mathematics 114*, Springer-Verlag 1987.

[KE93] R. Kuhn, P. Edfors, V. Howard, C. Caputo, T.S. Phillips. Improving Public Switched Network Security in an Open Environment. *IEE Computer* Vol. 26, No.8, August 1993, pp. 32-35.

[KN93] J.T. Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service, *Internet RFC 1510, M.I.T. Project Athena*, Cambridge, Massachusetts, September, 1993.

[L85] M. Looney. CORE - A Debrief Report. *NCC Publications*, Manchester 1985.

[LBMC94] C.E. Landwehr, A.R. Bull, J.P. McDermott and W.S. Choi. A Taxonomy of Computer Program Security Flaws. *ACM Computing Surveys* Vol. 26, No. 3, September 1994, pp. 211-254.

[LS79] B.Lampson and R.F.Sproull. An open operating system for a single user machine. In *ACM 7th. Symposium on Operating Systems Principles*, December 1979.

[LSP82] L.Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, Vol. 4, July 1982, pp.382-401.

[M79] G. Mullery. CORE - a method for controlled requirements specification. *Proceedings of the 4th. International Conference on Software Engineering*, Munich 1979.

[M93] S. Mullender (Ed). Distributed Systems, Second Edition. *Addison-Wesley* 1993.

[MH89] J. McAfee and C. Haynes. Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System, *St. Martin's Press*, New York 1989.

[MNSS87] S.P. Miller, B.C. Neuman, J.I.Schiller, J.H. Saltzer. Section E.2.1: Kerberos Authentication and Authorisation System, *M.I.T. Project Athena*, Cambridge, Massachusetts, December 21, 1987.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM* , Vol. 21, No. 12, December 1978, pp. 993-999.

[OECD92] Guidelines for the Security of Information Systems. *Organisation for Economic Co-operation and Development. Paris 1992. OCDE/GD(92)190.*

[OED71] Oxford English Dictionary, *Oxford University Press* 1971.

[OR87] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review* , Vol.21, No. 1, January 1987, pp. 8-10.

[P79]] C.H. Papadimitriou. The Serializability of Concurrent Database Updates. *Journal of the ACM*, Vol. 26, N0. 4, 1979, pp.631-635.

[P87] Roger S. Pressman. Software Engineering, A Practitioner's Approach. *McGraw-Hill* 1987.

[P89] C.P.Pfleeger. Security in Computing. *Prentice-Hall International, Inc.* 1989.

[PJCK97] S.L. Pfleeger, R. Jeffery, B. Curtis and B. Kitchenham. Status Report on Software Measurement. *IEEE Software* Vol. 14, No. 2, March/April 1997, pp. 33-43.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, Vol. 27, No. 2, April 1980, pp. 228-234.

[R95] Regina v. Hodges and Moore. Southwark Crown Court, September 1995.

[RT96] C.V. Ramamoorthy and Wei-tek Tsai. Advances in Software Engineering. *IEEE Computer*, Vol. 29, No. 10, Oct. 1996, pp. 47-58.

[S88] John Shore. Why I never met a programmer I could Trust. *Communications of the ACM* Vol 31, No. 4, April 1988, pp. 372-375.

[S89] Ian Sommerville. Software Engineering, Third Edition. *Addison-Wesley* 1989.

[S90] D.A.Stokes. Requirements Analysis. J.A. McDermid, (ed), Software Engineer's Reference Book. Butterworth-Heinemann 1990.

[S91] Daniel F. Sterne. On the Buzzword "Security Policy". IEEE 1991.

[S92] G.J. Simmons (ED). Contemporary Cryptology. *IEEE Press*, New York 1992, Ch. 4, Appendix F, pp. 257-258.

[S96] Bruce Schneier. Applied Cryptography. *John Wiley and Sons, Inc.*, New York 1996

[SR77] K. Schoman and D. T. Ross. Structured analysis for requirements definition. *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, 1977, pp. 6-15.

[T84] Ken Thompson. Reflections on Trusting Trust. *Communications of the ACM* , Vol. 27, No. 8, August 1984, pp. 761-763.

[TB74] D.C. Tsichritzis and P.A. Berstein. Operating Systems. *Academic Press* 1974.

[T88] G.W.Treese. Berkeley UNIX on 1000 Workstations, ATHENA changes to 4.3BSD. *Proc USENIX*, Winter 1988.

[W90] Dominic Walsh. Codes and Cryptography. *Clarendon Press,* Oxford 1990.

[WA91] George R.S. Weir and James L. Alty. Human-Computer Interaction and Complex Systems. Academic Press, London 1991.

YC79] Edward Yourdon and Larry L. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. *Yourdon Press*, Englewood Cliffs, NJ. 1979.

FINIS