# Summary

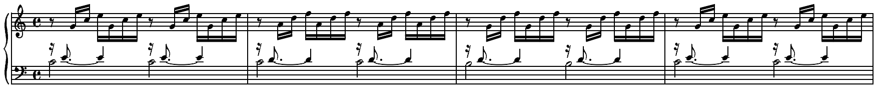
[JLS76] writes, "The Folklore is replete with stories of 'secure' protection systems being compromised in a matter of hours. This is quite astounding since one is not likely to claim that a system is secure without some sort of proof to support the claim. In practice, proof is not provided, and one reason for this is clear: although the protection *primitives* are apparently quite simple, they may potentially interact in extremely complex ways. Vague and informal arguments, therefore, often overlook subtleties that an adversary can exploit. Precision is not merely desirable for protection systems, it is mandatory."

In this thesis, we construct a formalism which provides the required precision.

We initially study the decidability of safety in specific protection systems using the common access-matrix model. We build upon this to the point where we can treat the systems themselves as computational elements and thus create a formalism which produces safety results for entire classes of protection systems.

Our study of the manipulation of rights encompasses a study of responsibility; we achieve a unification of the $\lambda_{\text{sec}}$-calculi for stack inspection with that for data inspection, and demonstrate an extremely close coupling between the calculi of responsibility and our formalism for decidability of safety in protection systems.

We use our formalism to model numerous existing systems and understand their strengths and their weaknesses; hence we are able to create powerful new systems without such weaknesses. Several such systems exist; we describe a sample implementation.



# 1 Introduction

Many of us are familiar with the mechanics of breaking computer security systems. But while we may understand some of these procedures at a purely mechanical level, do we really understand *why* we are able to crack systems? Where do these vulnerabilities come from? What aspects of protection system design allow so many trivial misfeatures to become such major problems?

Many of us also have some knowledge of how to create a relatively secure system. Such systems tend to be minimally permissive, frequently to an extent that they become unusable. Modification of such a system by an untrusted user is unthinkable. But even the less "secured" systems we use for day-to-day work tend to disallow almost entirely the transport or reassignment of permissions, because there is a general lack of understanding of the dynamics of permissions.

Our lack of understanding of all aspects of protection systems has left us unable to answer a great many questions, including:

- "*What undiscovered vulnerabilities exist in a system?*"
- "*How can we design a system in which minor implementation errors in application software do not affect overall system security?*"
- "*How do we balance security against permissiveness?*"

All of these questions and many others can be answered by a formal analysis of the design of protection systems.

## 1.1 What questions can we answer?

So what questions can we answer? The questions proposed above have arisen as problems through limited experience with our current repertoire of protection systems. There are many more open questions in the field of protection. There are even some questions at the asking of which eyes are rolled skywards. Yet many of these questions are answerable.

**We can build a system which automatically identifies vulnerabilities within itself.**

This is not entirely a new concept; intrusion detection systems such as Tripwire ([SK03]) already have limited functionality for identifying vulnerabilities within a system. However, Tripwire and its ilk are concerned only with identifying the occurrence of *particular* vulnerabilities. The utility of such a system is based on a previous static and largely experiential analysis of a *sample* Unix configuration, with conclusions such as "*if the password file is writable to any user, then the system may be compromised.*" If the system configuration is modified in any way, both the previous analysis and the existing product are liable to be inaccurate or useless.

If we can efficiently analyse any configuration of a particular protection system, then the system can accurately advise of the consequences of any proposed configuration change before it is enacted. This suggested analysis is generally impossible. However, there exist protection systems for which it is possible in linear time and space, and we exhibit a selection of these systems in section 7.

**We can build a system where mistakes in application code do not create security vulnerabilities.**

Writing a *secure* program frequently involves considerably more than simply implementing the functional specification. If untrusted input can influence the program behaviour, then this input could cause the program to attempt a protected operation with a malicious outcome. The source of any such data must be considered by the protection system when a protected operation is attempted.

If the underlying system does not provide and maintain security information for input data, then it becomes the responsibility of the application to compute and maintain this information. It is a consequence of such a system architecture that security must be considered in every line of application code that manipulates untrusted input, thus making the system prone to security errors.

We can design a system architecture such that security information is automatically maintained by the system. This information will be considered by the decision processes of the protection system when necessary. In such a system, secure programs may easily be constructed by a programmer who is entirely unaware of the concerns of security. Security then becomes solely the domain of the protection system.

**We can build a secure protection system yet still allow untrusted users to manage rights.**

It is impractical in any large organisation for all protection system administration to be performed by a central system administrator. The size of such an organisation requires that control be delegated. However, the trade-off between flexibility and security in protection system design has always been biased towards the side of security, and consequently against this kind of flexibility.

Giving arbitrary users any freedom to administrate the protection system has always been considered too great a risk to security: If our protection system does not adequately protect itself from malicious modification, then a user might exploit this to destroy the system. Such a system would, of course, be fundamentally insecure, but we may not have any way of proving that.

We can build a system within which it is safe to delegate the administration of the system to the users, to the extent that the role for the central administrator in handling permissions is almost abolished. We can even establish such a system over a cross-organisational network such that each local administrator can administrate rights within his own systems, yet the systems work securely together as a network under a single protection system.

**We can build practical, expressive systems with these properties.**

Many of these properties will appear surprising to the reader familiar with traditional protection systems, and it is natural to wonder what is lost in order to make these gains.

Nothing is lost. What is gained is simply an understanding of what properties are held by various types of protection system. Instead of increasing security by making a system increasingly restrictive, we create security by design, by analysis and by understanding the mechanisms of protection. There exist many systems with desirable properties, and we may choose freely to use any of these systems.

## 1.2 What questions do we not answer?

**We do not prove any implementation correct.**

It is important to make the distinction between correctness of design and correctness of implementation. Even given a correct design, any implementation containing an error may be insecure, but such an error may be corrected to create a secure system. A system with a flawed design may not necessarily be corrected; it must often be redesigned from scratch. Our work includes the design, example algorithms and complexity results for these algorithms, but does not include any technique for proving an implementation correct.

**There is security, and there is insecurity.**

A system which can be broken is not secure. A system which cannot be broken is secure. A system which we do not yet know how to break, but has not been formally proven secure is not to be considered secure. The common phrase, "*more secure*" must be interpreted as, "*more likely to be secure*". This is not a work on probability, and so we will not use such phraseology.

## 1.3 Our Approach

In this thesis we build an understanding of protection. We have learned many things about protection in the real world through thousands of years' experience. Many concepts with which we are familiar may be identified in our box and key example, and eventually will form the building blocks of a formal model. However, while we later provide a considerable amount of formal material, it is more important to understand the fundamental concepts and have an appreciation for how the models actually work.

### 1.3.1 Protection

We first introduce the concept of the protection system in section 2. We describe the shape and operation of a protection system in the real world, the example from which all computer protection systems are derived, that of a box with a lock and key, presented in section 2.1.2. It is from these roots that we discover the mechanisms of access control, or protection.

Many of us are already familiar with two stages of an access control decision: authentication and authorisation. It might come as a surprise to learn that every access control decision actually consists of *three* distinct computations, described more fully in section 2.1.3. First, we must identify the people around us, the authentication stage. Second, we must work out which of these people is responsible for making the request; this person is the "current principal". Third, we must decide whether that person is permitted access. The latter two stages form what has previously been called the authorisation stage, but they will rapidly become clear as distinct, but intertwined mechanisms. Of these stages, we leave authentication to the cryptographers, and study only the elements of authorisation, or as we call it, '*protection*'.

Having identified the mechanisms, we must have the naming of parts. Some of the basic definitions which follow in section 2.2 and the design choices of section 2.3 may appear obvious to the modern reader, but they are not automatic, and must be explicitly codified as axioms.

### 1.3.2 Policy

Our objective is to achieve security. Security is a funny thing: We might glance at a system and pronounce it apparently '*secure*' without any measure of objectivity, but when it is broken, we exclaim, "*Oh, it must have been insecure!*"[3] We do not yet have an absolute concept of security. By contrast, the term "*secure*", as we apply it to protection systems, must be an absolute. A protection system is either secure, or it is not. But there is as yet no yardstick by which we may classify a protection system as secure or insecure. What are we protecting, from whom, and against what?

Section 3 both introduces and completes our study of policy. A policy is an arbitrary specification for the protection system: any protection system which satisfies a policy is secure, or correct with respect to that policy. A policy may state who must be allowed access to objects, and who must be denied access. The protection system is responsible for enforcing the policy. But a policy makes a far stronger requirement than the specification of a state of a protection system. The state of a protection system indicates the accessibility of objects at a given time. A policy states what the accessibility of objects must be *at all times*, even into the infinite future of the protection system. It is this *at all times* which defines "vulnerability". If a protection system can, in the future, allow access, then it is considered to be vulnerable, and we call the system "*unsafe*". Otherwise, the system is "*safe*". Thus policy is the yardstick used for identifying vulnerabilities, and provides the questions we would ask of any analysis of our protection system.

[3] But we've fixed it and it's secure now, honest!

Having provided that important motivation and element of understanding, the policy steps into the background of the thesis. We will also prefer not to use the word "secure", since an absolute definition in terms of policy is available.

### 1.3.3 Formalisms

A basic formalism for protection systems has been around since 1976; we reproduce it in section 4. Part of the reason for the longevity of this model is its expressiveness; it can express formally the behaviour of any protection system. Such expressiveness is also its downfall: it can simulate a Turing machine, and therefore no useful positive results about protection systems may be constructed from this formalism. The major negative result of this formalism is reproduced in section 4.4; it is the undecidability of the "General Safety Problem", which states that it is not possible to analyse an arbitrary protection system to discover if it is safe.

The historical presentation of the formalism for protection systems is unfortunately lacking in at least one major respect, the consideration of the "current principal". While this does not affect the historical results, it does affect our ability to use the formalism in any practical analysis, and we must correct this omission in section 4.6 before we can apply the formalism to practical systems.

### 1.3.4 Mathematics

We may also construct many useful tools for manipulating protection systems as computational entities. Such tools allow us to classify a system, to describe the expressiveness of a system, and to make transformations of a system while retaining the properties of that system. It is these tools upon which we will rely to show that a result proven for one system transfers to another *equivalent* system or that a protection system is as *expressive* as another system. Our definition of equivalence is especially important, since it allows us to identify and study equivalence classes rather than individual systems, thus saving considerable effort.

Section 5 is a collection of this formal work, in which we introduce the full definitions of simulation, equivalence and expressiveness. It is frequently these definitions which allow us to make objective statements about the utility or behaviour of protection systems. Rarely will we use these definitions formally, although we frequently make implicit use of them by reference.

### 1.3.5 Classes

Given a protection system, any decidability result for that system will hold for the equivalence class of systems containing that system. However, we can also prove results about classes larger and more general than equivalence classes. A sufficient restriction of the general formalism for protection systems might produce a class of protection systems such that the "Class Safety Problem" is decidable. If any class safety problem for a particular system is shown to be decidable with sufficiently low complexity, then an algorithm for deciding the safety of particular system configurations may be constructed for the system. Thus we may identify protection systems which are correct, expressive, fast and decidably safe in low order polynomial time. Such systems may be of practical use.

In section 6, we study a number of classes of protection system strictly within the formalism. But the formalism itself does not provide any method for constructing classes worthy of study. The work in this section is arduous, and consists mostly of laborious stabs in the dark which fail to hit anything.

There is another method by which we may identify a class of practical protection systems. Section 7 shows that we may also construct a practical protection system from the ground up. We take first the simplest possible system and show that the safety problem for this system is decidable. We then extend it to satisfy many desirable functional requirements, and thus by a process of exploration, identify a useful protection system with a decidable safety problem. We study some instances of the class safety problem explicitly in this section, but we focus on the practical and administrative functional requirements for our '*ideal system*'.

It is indeed possible to produce an ideal system in this way, and produce elegant and fast algorithms to show that such a system is safe. However, in the development of this system, the business requirements have led us to create a far more powerful structure of permissions than we had previously anticipated, and these give some surprising results. We also discover towards the end of section 7 that the concept of the '*principal*' is far more complex and important than previously realised.

### 1.3.6 The Current Principal

Of our two initial problems of protection: identification of the current principal, and access control, we have traditionally considered computing the first correctly to be trivial, and deciding the safety of the second to be more difficult. After all, the current principal is just the person making the request. But section 7 inspires our interest in the principal as an abstract computational entity. Also, all the mechanisms that we have traditionally used for identifying the current principal appear to be flawed! We can not simply assume that the person making a request is '*responsible*' for the request because it isn't generally true.

We learn in section 8 that the principal is an abstract object not necessarily representing any person. We may represent any principal as a set of permissions and thus compute with principals to identify a current principal. By performing this computation correctly we may actually make the access control decision trivial; either the current principal has the required permission, or it does not. But now, identification of the current principal is the hard task. We show, in one of our more formal sections, the weaknesses of the various methods of computing a current principal. We can also show that there is really only one "right answer", only one correct way to compute the current principal, if we are to create protection.

### 1.3.7 Case Studies

The case studies are fairly informal with respect to much of what has gone before. It is sufficient to classify a system by the model it attempts to implement; usually this is clear, but sometimes more justification is needed. The case studies make two major illustrations. First, they show that the results from our formal study match experiential evidence from the real world. Where we predict a hole there exists a hole. Second, they show how little has been done to exploit the possibilities that a better protection system design has to offer. The author has twice implemented such a system in a medium scale environment, but of our case studies only two allow the transport of rights by unprivileged users. Of those, one is obsolete and one is the author's own implementation.

### 1.3.8 New Applications

If we use the optimal structure for permissions developed in section 7 and the mechanisms for the identification of the current principal from section 8, we can achieve a surprising system which retains all the positive results from our previous work. An informal sketch of such a system which builds on our previous work is presented in section 10, and we hope that this design can form a basis for future implementations of protection systems.

Section 11 contains our conclusions and section 12 describes possible further work.

### 1.3.9 Previous Work and Bibliography

A list of reference materials may be found in the bibliography on page 221. We reference work by other authors throughout the main text, rather than grouping all references in a single section or subsection dedicated to "Previous Work". In this way, references will be placed alongside the material to which they are relevant.

We also provide an index of concepts on page 229.

## 1.4 The Organisation of this Thesis

### 1.4.1 In Pictures

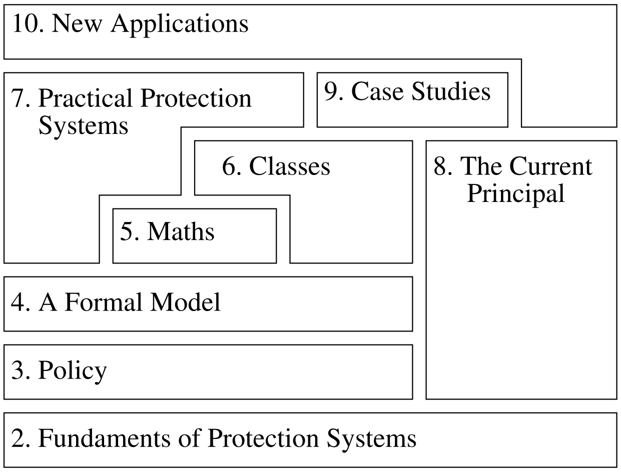


Figure 1: The structure of this thesis

Figure 1 is a dependency diagram of the thesis. We build upwards from the fundamental ideas to the new applications. Any section which rests on a lower section depends on the material within that section. Thus it is possible to study section 8 without understanding the material in section 5. However, it is recommended that the sections be read in order, since later sections solve problems inspired by earlier ones.

### 1.4.2 In Words

In section 2, we introduce the concepts of security and protection and introduce our first model of a protection system. We use this first model to help us identify the basic mechanisms of protection. We make some essential definitions, and establish much of the groundwork for a fuller formalisation of the mechanisms of protection.

Section 3 introduces the concept of a system policy: those requirements made of a protection system by the system administrator, and will motivate our concept of "*adequate*" protection by establishing a hard set of requirements for any protection system.

In section 4, we introduce the formal model and the "*General Safety Problem*". We introduce with these many concepts and tools with which we may work. Some major flaws are exhibited in the previous study of this problem, most notably in section 4.6, where we extend the formulation of the safety problem with the introduction of the concept of the current principal.

Section 5 demonstrates techniques for manipulating the definition of a protection system, including the important definitions of simulation, equivalence and expressiveness.

Section 6 uses these tools to identify subclasses of protection system for which the safety problem is decidable. We will endeavour to identify subclasses large enough to be generally useful and contain security models currently used but small enough to be decidably safe in low order polynomial time. This section includes both new and existing proofs.

Section 7 takes our theory back into the real world. We build a restricted taxonomy of protection systems designed to include many common real world systems. We compare the systems from this taxonomy using both quantitative and qualitative requirements to discover some extremely powerful and versatile systems with security decidable in linear time.

In section 8, we will show that we can reliably define and identify a current principal at any point in a computation. We formalise many models in a single $\lambda_{\mathrm{sec}}$-calculus, and show many equivalences and relationships between this model using the common formalism.

In section 9, we will use the knowledge gained from our formalisms to produce full case studies of various existing systems, showing how they fit into our model and where they are lacking. In this way, we will expose the vulnerabilities in these systems and show how they may be broken or *hacked*.

If we can understand why existing models are lacking, then we can design new systems without such faults. Section 10 sketches how we might extend traditional algorithms up to and including dealing with a network of untrusted hosts.

Our conclusions are in section 11, and some notes on further work are in section 12.

But first, we must meet the cast …