The Formal Design of Protection Systems

Ben Mankin
University of Bath
ben.mankin@anarres.org

Introduction

Security is increasingly a fundamental requirement of business and industry, for practical, legal and competitive reasons. However, security is not a fundamental requirement in the design of the current generation of computer systems. Vulnerabilities appear and are exploited, and even the introduction of excessively restrictive security systems has done little to reduce the impact of such exploits.

The object of our work is to produce designs for correct, flexible security systems to underpin the next generation of systems such that security is handled in a consistent, robust and provably correct manner throughout any computer system. Our work is proof-oriented, and we give absolute security and computational results for any systems we produce. We construct systems having all of the following properties:

- Guaranteed conformance to a security policy.
- Secure delegation of security system administration.
- Distribution over a network without central authority.
- Human-oriented justification of all access decisions.
- Security is maintained outside application code.
- Accounting and auditing information is available on demand.

Our increased understanding of protection systems also allows us to perform a limited security analysis of many legacy systems, and previously unknown vulnerabilities can sometimes be identified.

Overview

Many of us understand some of the mechanics of identifying vulnerabilities in a computer security system. These holes tend to be implementation bugs, and are frequently found in application software, rather than in the operating system itself. But we do not understand *why* trivial bugs can become such major vulnerabilities. Where do vulnerabilities come from? What aspects of protection system design allow so many trivial misfeatures to become 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 impractical. Modification of such a system by an untrusted user would be 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.

- "*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?*"

The questions proposed above have arisen in our limited experience with protection systems. There are many more open questions which have not yet arisen in practice, and all these questions can be answered as a consequence of a formal analysis of the design of protection systems.

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

This is not *entirely* new; intrusion detection systems have limited functionality for identifying vulnerabilities. However, they are concerned only with identifying the occurrence of *particular* vulnerabilities based on a previous experiential analysis of a *sample* system configuration.

If we can efficiently analyse any configuration of a particular protection system, then the system can accurately report 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.

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

Security is an implicit requirement above and beyond any functional specification. If the underlying system does not maintain security information for input data, then the application must maintain this information lest untrusted input cause a program to attempt a protected operation with a malicious outcome. Overall system security must be considered in every line of application code, 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, distributed protection system without central administration.

It is a requirement in any large organisation that protection system administration be delegated. However, the apparent tradeoff between flexibility and security in protection system design has always been biased towards the side of security, and consequently against this kind of flexibility.

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 abolished. We can even establish such a system over a cross-organisational network such that each local administrator or user 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.

Nothing is lost in order to make these surprising gains. We simply achieve 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. Unsurprisingly, most of the protection systems currently in use do not fall into *any* of these categories.

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 secure, and there is insecure. There is no *"probably"*.

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*". We are not statisticians and so we will not use such phraseology.

Applications

Practical applications exist for our work in many industries, including but not limited to those where data integrity, privacy or data protection are paramount.

Financial Services: Security information maintained at operating system and network level would make correct accounting information available to any financial services organisation processing electronic transactions. It would be impossible for an attacker to cause a fraudulent transfer of funds since his actions would be accounted for, and hence detected.

Data Protection: An organisation dealing with patient records or customer data would benefit from a system which guaranteed compliance with the data protection act and related legislation. It would be possible for the applications developers to write arbitrarily permissive groupware applications in the knowledge that the underlying operating system and network protocols would prevent any undesirable leakage of information.

Agents and Grid Computing: The legal status of software agents can be established only when their provenance and integrity is assured. As basic access control enabled timesharing, a network oriented security system design can enable network sharing, creating an environment in which agents can safely operate.

Total accountability does not immediately cause loss of privacy. Our work identifies many circumstances where security information should be partially or fully discarded. In addition to our applications, we look at the relationship between security, privacy, anonymity and repudiation, and show that these concerns are not mutually exclusive.

Conclusions

The designs proposed by our research solve a large part of what has been termed the *security problem*. This problem has arisen largely through a lack of understanding of the nature of vulnerability, responsibility and policy. As a consequence of this, the current generation of security systems are after-market additions which attempt to compensate for poor design with excessive restriction on the user.

With our improved understanding, we have designed a system to satisfy not only the functional requirements of security, but also the computational requirements of protection system analysis, the social requirements of privacy and many practical requirements proposed in this document and elsewhere. Any system which correctly answers the security problem for a given set of requirements may enable a vast range of new technologies. We are confident that further research will extend our ability to construct and use protection systems to satisfy new requirements as they arise.

Ben Mankin
23rd October 2003