Appears in *Festschrift Randell*, C.B. Jones and J.L Lloyd (Eds.), Springer LNCS 6875, pp. 53–57, August 2011.

From DSS to MILS[*]
(Extended Abstract)

John Rushby

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025 USA

Abstract. I outline the principal ideas of the Distributed Secure System (DSS) on which Brian Randell and I collaborated in the early 1980s, its modern manifestation as MILS, and continuing research challenges posed by these architectures.

1 Introduction

I studied Computer Science at Newcastle, first as an undergraduate and then for my PhD, from 1968 to 1974. That makes my 1971 Bachelors degree among the earliest awarded in Computer Science (Newcastle first awarded the degree in 1969, and I believe Manchester was one year earlier). Brian Randell joined the department in 1970 but, although I profited from the stimulating impact of his presence, we did not work together until I returned to Newcastle as a Research Associate in 1980, following some interesting and enjoyable years programming at the Atlas Computer Laboratory and teaching at Manchester University.

The opportunity that motivated my return was a project in computer security led by Peter Henderson and funded by (what was then) the Royal Signals and Radar Establishment. Brian and I collaborated on a conceptual, and later real, security architecture that became known as the Distributed Secure System (DSS). We were recently invited to revisit this work for a presentation as a "Classic Paper" at ACSAC [1] and I will not repeat the historical recollections presented there. Instead, I want to focus on the key ideas from that work that seem to have some durability, and to point to their possible future evolution.

2 Policy and Sharing

The central idea underpinning our work was the observation that computer security is composed of two separate problems. One is the problem of enforcing

[*] This work was supported by National Science Foundation Grant CNS-0720908. The content is solely the responsibility of the author and does not necessarily represent the official views of NSF.

policy on the movement and processing of sensitive information (for example, information from a classified environment may not be released to the network without suitable review and authorization), and this problem is present no matter how the system is implemented. The second problem arises when entities storing, communicating, or processing information of different sensitivities share the same resources, and this problem is strictly a consequence of sharing: for example, files of different sensitivities may reside in a shared file system and, by error or malice, sensitive information may leak from the disk sectors of a highly classified file to one of lower classification.

Most prior (and, indeed, much later) work on computer security conflates these problems; the innovation of the DSS was to propose that they be handled separately and that this separation of concerns can lead to a simpler system design and implementation, and more convincing assurance that security is achieved. This proposal was largely stimulated by the work Brian was then leading on distributed systems [2], which made it feasible to contemplate implementations for simple secure systems in which there was no sharing of resources, thereby directing attention to the problem of enforcing policy.

A modern formulation of this approach is the MILS architecture employed in several US defense systems [3]. In its MILS formulation, secure system design begins with development of a *policy architecture*; this is an abstract architecture that can be represented as a "boxes and arrows" diagram whose focus, as its name suggests, is the *policy* to be enforced. The boxes (representing components such as processes or subsystems, depending on the granularity of the representation) are divided into trusted and untrusted components; to the extent possible, the *purpose* of the system is accomplished by untrusted (possibly legacy) components and policy is enforced by trusted components interposed in some of the arrows, which represent communications paths such as networks or interprocess communication.

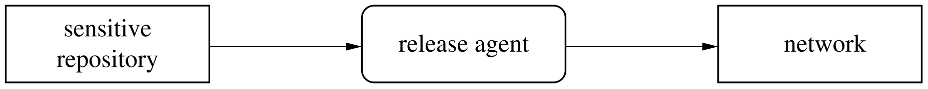

Fig. 1. A Simple Policy Architecture

Figure 1 displays a simple—in fact, just about the simplest—policy architecture: information may flow from an untrusted *sensitive repository* to an untrusted *network* only via a trusted *release agent* (we use a differently shaped box to indicate it is trusted).

The job of the release agent is to enforce some policy on the release of sensitive information (e.g., to redact certain items, or to reduce the accuracy of some numerical fields). It must do its job correctly, and we will later consider the difficulty of ensuring that it does so, but the point of the policy architecture is to create an environment in which it *can* do its job, for the architecture provides no path from the repository to the network save that through the release agent: the *absence* of arrows (in this case, of one directly from sensitive repository to network) is often the key decision in a policy architecture.

The boxes and arrows of a policy architecture are conceptually dedicated individual components and communications paths, but their implementation may share resources, provided this is done securely. For example, Figure 1 could be implemented in a single computer as shown in Figure 2.

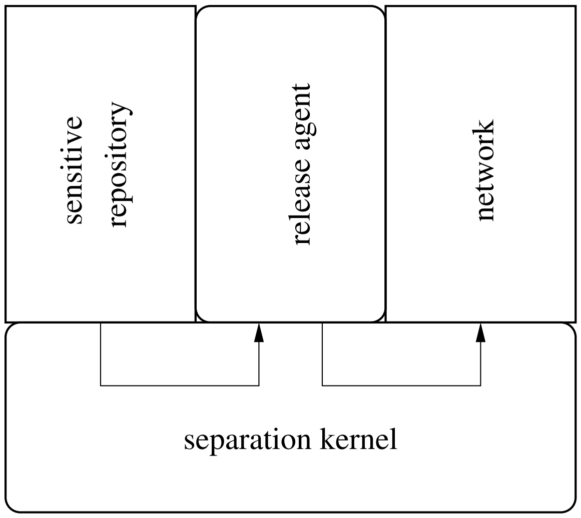

Fig. 2. Policy Architecture of Figure 1 Implemented using a Separation Kernel

The task of resource sharing components or mechanisms such as the separation kernel shown in Figure 2 is to *implement* a policy architecture: the sharing of resources must introduce no arrows or boxes other than those specified in the policy architecture (despite possibly nefarious activity by untrusted occupants of some boxes). The DSS identified four mechanisms for achieving this separation: logical (essentially separation kernels [4]), temporal (periods processing), cryptographic, and physical (no sharing).

Untrusted components that manipulate information of different sensitivities are a large source of complexity in many policy architectures, necessitating trusted "cross domain solutions" of the form illustrated in Figure 1 to mediate outgoing information flows. Guards and filters and other kinds of release agent are often needed only because the upstream information is a "blob" of different sensitivities managed by an untrusted entity, and the task of these agents is complex because essentially nothing can be assumed about the integrity of information leaving the blob. A major simplification can often be achieved by replicating the untrusted components and arranging matters so that each replica operates on information of a single sensitivity, thereby avoiding creation of the blob. This replication is feasible because technologies such as separation kernels (in modern terminology, these are essentially minimal and secure hypervisors) allow a single resource (such as a processor) to be virtualized and shared securely (and at essentially no cost) among multiple instances of an untrusted entity.

3 Formal Models for Policy Architectures

DSS left assurance as an exercise for the reader. There has been much progress since then, but some interesting research challenges remain. The top problem, in my view, is to develop a formal model or, rather, as I will explain, a class of formal models for policy architectures. These are needed to provide assurance that a given policy architecture, plus the properties ("local policies") assumed of its trusted components, delivers the required overall security policy, and also because the task of resource sharing components is to implement (parts of) a policy architecture—so a formal model for the architecture is needed to provide a requirements specification for the assurance of those resource sharing components.

The important attribute of a formal model for the policy architecture of Figure 1, for example, is to specify that there can be flows of data and information from the sensitive repository to the release agent and from there to the network, but that there must be no direct flow from the sensitive repository to the network. I tackled this problem of *intransitive flows* in [5], building on earlier work of Haigh and Young [6]. There have been many subsequent developments, reinterpretations, and misinterpretations of this work, but the most satisfactory is by van der Meyden [7], who demonstrates an exact correspondence between a revised formulation of "intransitive noninterference" and its natural "unwinding conditions." He and Chong [8] further show how properties of a policy architecture can be derived in this framework, given those of its trusted components, which are represented as functions.

The reason I believe there is still work to be done is that Figure 1 has a robust intuitive interpretation: it does not depend on the precise models of computation or communication to see that this figure requires the absence of unmediated flows from the sensitive repository to the network. The large body of work on variants of intransitive noninterference for different commputational models suggests it is not a simple task to formally model this apparently simple intuition. Much recent work casts the problem in terms of programming languages, and indeed it is necessary for the more abstract treatments to connect to this work, so that internal information flows in the program that implements a release agent, say, can be used in verifying that it enforces its local policy. In addition, the properties of the release agent can surely be relational, rather than purely functional as in the treatment of van der Meyden and Chong. Hence, what I seek is a "metaspecification" which, for any reasonable model of computation and communication, delivers the aproppriate interpretation of intransitive noninterference. And furthermore, given relational properties of the trusted components, this notion should allow the properties of the overall policy architecture to be calculated in a compositional way.

It is generally believed that system-level properties such as safety and security are not compositional, but I believe that information flow security properties can be developed compositionally using the approach that originated in DSS. So, in conclusion, ideas on computer security that began with Brian Randell thirty years ago (and not just those recounted here, but also those from his work with John Dobson [9]) continue to be relevant today, and continue to pose intriguing challenges for further research.

References

1. Randell, B., Rushby, J.: Distributed secure systems: Then and now. In: Proceedings of the Twenty-Third Annual Computer Security Applications Conference, Miami Beach, FL, IEEE Computer Society (2007) 177–198. Invited "Classic Paper" presentation. 1
2. Brownbridge, D.R., Marshall, L.F., Randell, B.: The Newcastle Connection, or UNIXes of the world unite! Software—Practice and Experience **12** (1982) 1147–1162 2
3. Boettcher, C., DeLong, R., Rushby, J., Sifre, W.: The MILS component integration approach to secure information sharing. In: 27th AIAA/IEEE Digital Avionics Systems Conference, St. Paul, MN, The Institute of Electrical and Electronics Engineers (2008) 2
4. Rushby, J.: The design and verification of secure systems. In: Eighth ACM Symposium on Operating System Principles, Asilomar, CA (1981) 12–21 (ACM *Operating Systems Review*, Vol. 15, No. 5). 3
5. Rushby, J.: Noninterference, transitivity, and channel-control security policies. Technical Report SRI-CSL-92-2, Computer Science Laboratory, SRI International, Menlo Park, CA (1992) 4
6. Haigh, J.T., Young, W.D.: Extending the noninterference version of MLS for SAT. IEEE Transactions on Software Engineering **SE-13** (1987) 141–150 4
7. van der Meyden, R.: What, indeed, is intransitive noninterference? (extended abstract). In: 12th European Symposium on Research in Computer Security (ESORICS). Volume 4734 of Lecture Notes in Computer Science., Dresden, Germany, Springer-Verlag (2007) 235–250 4
8. Chong, S., van der Meyden, R.: Using architecture to reason about information security. Technical report, University of New South Wales (2009) 4
9. Dobson, J., Randell, B.: Building reliable secure computing systems out of unreliable insecure components. In: Proceedings of the Seventeenth Annual Computer Security Applications Conference, New Orleans, LA, IEEE Computer Society (2001) 162–173. Invited "Classic Paper" presentation. 5