2008

Least privilege in separation kernels

Levin, Timothy E.; Irvine, Cynthia E.; Nguyen, Thuy D.

IEEE Design and Test of Computers

IEEE Design and Test of Computers, Vol 25, No. 6, pp 590-598.
http://hdl.handle.net/10945/7160

Least Privilege in Separation Kernels

Timothy E. Levin, Cynthia E. Irvine, and Thuy D. Nguyen

Department of Computer Science, Naval Postgraduate School
833 Dyer Rd., Monterey, CA USA
{Levin,irvine,tdnguyen}@nps.edu

Abstract. We extend the separation kernel abstraction to represent the enforcement of the principle of least privilege. In addition to the inter-block flow control policy prescribed by the traditional separation kernel paradigm, we describe an orthogonal, finer-grained flow control policy by extending the protection of elements to subjects and resources, as well as blocks, within a partitioned system. We show how least privilege applied to the actions of subjects provides enhanced protection for secure systems.

Keywords: Assurance, Computer Security, Least Privilege, Separation Kernel.

1 Introduction

The Sisyphean purgatory of penetrate and patch to which users of commodity systems are currently subjected has lead to increasing recognition that platforms with assurance of penetration resistance and non-bypassability are required for certain critical functions. This need for high assurance calls for a layered system architecture where enforcement mechanisms of the most critical policies themselves depend upon layers of no less assurance. For many high assurance systems currently being planned or developed, a general-purpose security kernel may provide more functionality than necessary, which has resulted in increased interest in the use of separation kernels to support real-time embedded systems and virtual machine monitors (VMM). Many of these separation kernels are minimized to have both static policies and static allocation of resources, such as is suitable for certain fixed-configuration or embedded environments.

Despite a resurgence of interest in the separation kernel approach, the principle of least privilege (PoLP) [19] is often overlooked in the design of traditional separation kernels due to the belief that a separation kernel should only be concerned with resource isolation. A principal consequence of this omission is that problems relating to all-or-nothing security and over-privileged programs are left for application designers (and security evaluators) to resolve. For systems that must protect highly sensitive or highly valuable resources, formal verification of the ability of the system to enforce its security policy is required. Recent advances in the assurance requirements for high assurance systems [14] have included verification of the target system's conformance to the principle of least privilege. To provide vendors and integrators with tools to formally describe least privilege in separation kernels, a least privilege separation model is presented.

J. Filipe and M.S. Obaidat (Eds.): ICETE 2006, CCIS 9, pp. 146–157, 2008.

1.1 A Least Privileged Separation Kernel

In the context of a research project to build a high assurance separation kernel [10] we have extended the separation kernel abstraction so that the principle of least privilege can be examined at the model level and can be verified to be enforced by systems that conform to that model.

The traditional separation kernel paradigm describes a security policy in which activities in different blocks of a partitioned system are not visible to other blocks, except perhaps for certain specified flows allowed between blocks. (Here, "block" is defined in the traditional mathematical sense as a member of the non-intersecting set of elements that comprise the partition 0). If information flow is described only at the block level, then everything in a block can flow to everything in another block. This is contrary to the principle of least privilege required in high assurance systems. The least privilege separation model builds on the traditional separation abstraction by extending the granularity of described elements to the subjects [9] and resources within the partition. An orthogonal flow control policy can then be expressed relative to subjects and resources, thus providing all of the functionality and protection of the traditional separation kernel, combined with a high level of confidence that the effects of subjects' activities may be minimized to their intended scope.

In the sections that follow we will elaborate on the concept of separation kernels and the need for least privilege in such systems. In particular, the granularity of inter-block flows will be discussed in terms of "subject" and "resource" abstractions. A formalization of the least privilege separation model is presented and several aspects of secure system design and verification are discussed with respect to the model. The last sections of the paper review related work, and summarize our results.

2 Concepts

2.1 The Separation Kernel

The term separation kernel was introduced by Rushby, who originally proposed, in the context of a distributed system, that a separation kernel creates "within a single shared machine, an environment which supports the various components of the system, and provides the communications channels between them, in such a way that individual components of the system cannot distinguish this shared environment from a physically distributed one" [18]. A separation kernel divides all resources under its control into blocks such that the actions of an active entity (i.e., a subject) in one block are isolated from (viz., cannot be detected by or communicated to) an active entity in another block, unless an explicit means for that communication has been established (e.g., via configuration data).

A separation kernel achieves isolation of subjects in different blocks by virtualization of shared resources: each block encompasses a resource set that appears to be entirely its own. To achieve this objective for resources that can only be utilized by one subject at a time, such as the CPU, the ideal separation kernel must ensure that the temporal usage patterns of subjects from different blocks are not apparent to each other. Other resources, such as memory, may be accessed by different blocks simultaneously, while preserving idealized isolation, if the separation kernel ensures, for example, that blocks are allocated

different and non-interacting portions of the resource. Furthermore, kernel utilization of its own internal resources must also preserve the desired isolation properties.

Separation kernels differ from virtual machine monitors, in that support for communication between blocks is required in the former, whereas a functional replication of the hardware interface is required in the latter. Specific implementations may, however, provide both kinds of support.

2.2 The Principle of Least Privilege

Saltzer and Schroeder concluded that least privilege is one of the eight design principles that can reduce design flaws [19]. They defined least privilege by stating "every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur."

A decade later, the U.S. Department of Defense included a similar definition of least privilege in the Trusted Computer System Evaluation Criteria (TCSEC) [6]. Layering, modularity and information hiding are constructive techniques for least privilege that can be applied to the internal architecture of the underlying trusted foundation (e.g., separation kernel) to improve the system's resistance to penetration. The kernel can also be configured to utilize protection mechanisms such as access control and fine-grained execution domains to limit the abilities of a subject so that it is constrained to perform only the tasks for which it is authorized.

2.3 High Assurance Criteria and Least Privilege

The TCSEC refers to the principle of Least Privilege in two different contexts: the internal structure of the "trusted computing base" (TCB), and the ability of the TCB to grant to subjects a minimal set of authorizations or privileges. Despite the lack of an explicit reference to the principle of least privilege, the Common Criteria (CC) [5] provides the groundwork for it in several ways. It defines assurance as "grounds for confidence that an entity meets its security objectives." The CC explains that the correctness and effectiveness of the security functions are the primary factors for establishing the assurance that security objectives are met. A high assurance separation kernel must be proven to correctly implement the security functions defined in its specifications and effectively mitigate risks to a level commensurate with the value of the assets it protects. To complement the formal proof, a constructive analysis is used to demonstrate that the implementation maps to the specification. Thus, a focus on resource separation and the structured allotment of privileges affords simplicity to the separation kernel, and enables a high assurance analysis of the correctness of its implementation.

If a system cannot restrict individual users and programs to have only the access authorizations that they require to complete their functions, the accountability mechanisms (e.g., audit) will likely be less able to accurately discern the cause of various actions. A securely deployed system must be capable of supporting least privilege, and must have been administratively configured such that any programs that might

execute will be accorded access to the minimal set of resources required to complete their jobs. To provide high assurance of policy enforcement, a system should be able to apply least privilege at the same granularity as the resource abstractions that it exports (e.g., individual files and processes).

2.4 Practical Considerations

In the commercial security community, the use of the principle of least privilege has taken on the primary meaning, over time, of placing limits on the set of simultaneous policy-exemption privileges that a single user or application program can hold, such as may be associated with a 'root' process on a UNIX system. The commercial use of "least privilege" is not concerned with internal TCB structure or with the limitation of normal file-access authorizations for non-privileged processes. Note however, that a separation kernel has no notion of policy-exemption privileges or of privileged processes -- if the SK does not provide individual authorizations to the resources available at its interface, it cannot be used provide least privilege protection in the application domain. It is also noted that commercial product vendors have long ignored the assurance benefits of well-structured code. Thus, commercial product development experience and precedence in the area of PoLP is not germane to the construction of high robustness separation kernels, wherein both contexts of PoLP must be applied.

In practice, a separation kernel providing strict isolation is of little value. Controlled relaxation of strict separation allows applications to interact in useful ways, including participation in the enforcement of application-level policies. In the latter case, applications hosted on a separation kernel will need to be examined and evaluated to ensure that the overall system security policies are enforced. A monolithic application that runs with the same set of privileges throughout all of its modules and processes is hard to evaluate. In order to reason about the assurance properties of the system, the applications should be decomposed into components requiring varying levels of privilege. Such decomposition is more meaningful if the privilege boundaries are enforced by the separation kernel, rather than relying on, for example, error-prone ad hoc agreements between programmers or integrators. The principle of least privilege affords a greater degree of scrutiny to the evaluation of both the kernel and the application, resulting in a higher level of assurance that the overall system security objectives are met.

To better understand the use of least privilege in a separation kernel, we now turn to a closer examination of isolation and flows in these systems.

3 Inter-block Flows

The first-order goal of a separation kernel is to provide absolute separation of the (effects of) activities occurring in different blocks. In practice, however, separation kernels are often used to share hardware among kindred activities that have reason to communicate in some controllable fashion. Therefore, we include in the separation kernel a policy and mechanism for the controlled sharing of information between blocks.

The control of information flow between blocks can be expressed abstractly in an access matrix, as shown in the example of Table 1. This allows arbitrary sharing to be defined, establishing the inter-block flow policy to be enforced on the separation kernel applications.

Table 1. Block-to-Block Flow Matrix

	Block A	**Block B**	**Block C**
Block A	RWX	W	-
Block B	-	RWX	W
Block C	-	-	RWX

Notice that an inter-block flow policy in which the flow relationships partially order the blocks, such as in Table 1, may be suitable for the enforcement by the separation kernel of a *multilevel* confidentiality or integrity policy if meaningful sensitivity labels are immutable attributes of the blocks. Under the conditions that a static separation kernel does not change the policy or resource allocation during execution, and that the policy is not changed while the separation kernel is shut down, the policy may be considered to be global and persistent, viz. non-discretionary. In this example, information flows (represented by $\Rightarrow$) form the following ordering: Block A $\Rightarrow$ Block B $\Rightarrow$ Block C. An assignment of labels to these blocks in conjunction with the rules defined in Table 1 results in a recognizable multilevel security policy:

```
Block A := Unclassified
Block B := Secret
Block C := Top Secret
```

The block-to-block flow policy allows all of the information in a "source" block (e.g., Block A, above) to flow to every element of a "target" block (e.g., Block B, above). Extending the Table 1 scenario, if block B is also allowed to write to block A, for example to implement a downgrade function with respect to the assigned labels, then all of the code or program(s) in block B would need to be examined to ensure that their activities correspond to the intended downgrading semantics. If this assurance of correct behavior cannot be provided, such a circular flow (A $\Rightarrow$ B $\Rightarrow$ A) would create, in effect, one large policy equivalence class consisting of all of the information in blocks A and B.

To limit the effects of block-to-block flows, we next introduce the notion of controlling how much information is to be allowed to flow between and within blocks.

4 Least Privilege Flow Control

The implementation of a separation kernel results in the creation of active entities (subjects) that execute under the control of the separation kernel and the virtualization of system resources exported at the kernel interface (see Figure 1). Historically, many security models have utilized the abstraction of an object [10]. Because objects have

been classified in various ways, we decided to avoid this nomenclature issue by simply modeling "resources." Similarly, as the definition of "resources" includes the abstractions that are exported by the separation kernel, "subjects" are defined to be a type of resource.

Resources are defined as the totality of all hardware, firmware and software and data that are executed, utilized, created, protected or exported by the separation kernel. Exported resources are those resources (including subjects) to which an explicit reference is possible via the separation kernel interface. That interface may include programming, administrative, and other interfaces. In contrast, internal resources are those resources for which no explicit reference is possible via the kernel interface.

Various implementations of separation kernels have elected to describe the system only in terms of blocks without describing the active system entities that cause information flow. Since the concept of subjects [10] is a term of art – and for good reason – we will use it to describe the active entities exported by in the separation kernel.

We have found the use of the subject abstraction to be indispensable for reasoning about security in secure systems. Without the subject abstraction, it may be difficult to understand, for example, which block in a partitioned system is the cause of a flow between blocks [1] (e.g., the flow could have been caused by the receiving block as a reader or by the sending block as a writer), which application programs within a block need to be trusted (e.g., evaluated with respect to the security policy), and how to minimally configure the programs and resources of such a system to achieve the principle of least privilege. Just as when writing prose, if actions are described *passively* (i.e., not attributable to the subject of a sentence) the cause of the action can be ambiguous. In addition, use of subjects permits construction of a resource-to-block allocation that provides a minimal configuration for least privilege (see Section 4.3). Modeling of subjects within a partition also allows the representation and examination of more complex architectures such as multiple rings of execution, as well as multithreaded and multi-process approaches.

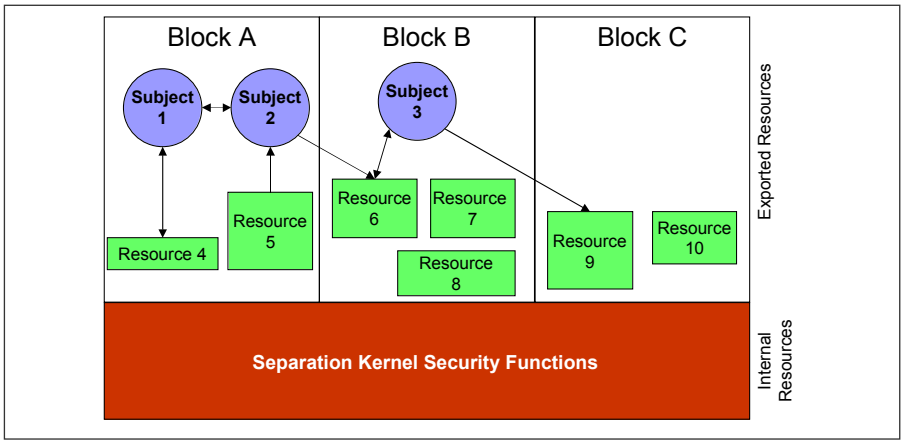

Fig. 1. Example Separation Kernel Configuration

Figure 1 shows an example separation kernel system with three blocks, three subjects, a set of other resources, and some designated flows.

An allocation of subjects and other exported resources to blocks is illustrated in Table 2, i.e., a "tagging" of each subject and resource with its partition (per Figure 1). Of the resources described in this table, the first three are subjects and the remaining exported resources are passive. Every resource is allocated to one and only one block. Consequently, we can state that the blocks of the separation kernel constitute a partition (in the mathematical sense) where: R is the nonempty set of resources and B is a nonempty set of subsets of R such that each element of R belongs to exactly one of the elements of B. From elementary set theory, it is known that a partition, B, can be used to create an equivalence relation on R. Thus we may induce that the allocation of resources to partitions creates equivalence classes.

Table 2. Resource to Block Allocation

	Resources									
	1	2	3	4	5	6	7	8	9	10
Blocks	A	A	B	A	A	B	B	B	C	C

The principle of least privilege requires that each subject be given only the privileges required to do its particular task and no more. The separation kernel can support this objective by assigning access rights appropriately to the subjects within the block. Rules can be defined for accessing different resources within a block. Table 3 illustrates how allocations to support the principle of least privilege are possible when the separation kernel supports per-subject and per-resource flow-control granularity: no subject is given more access than what is required to allow the desired flows (only the resources that are part of a flow are shown in this table).

Table 3. Subject-to-Resource Flow Matrix

		Resources					
		1	2	4	5	6	9
Subjects		-	R W	R W	-	-	-
		R W	-	-	R	W	-
		-	-	-	-	R W	W

Together, Tables 2 and 3 show abstract structures which allow only the flows illustrated in Figure 1. It is clear that the corresponding Block-to-Block flow matrix in Table 1, by itself, would allow many more flows than those illustrated in Figure 1.

5 Applications of Least Privilege

5.1 Kernel-Controlled Interference

In practical MLS system policies, several cases arise in which the normal information flow rules are enhanced. For example, (1) a high confidentiality user may need to downgrade a file and send it to a low confidentiality user, and (2) a high integrity user may need to read a low integrity executable file (viz., a program). In both cases, the system may allow the transfer if the file passes through an appropriate filter: in the former, the filter must ensure that the file does not include any high-confidentiality information; in the latter case, the filter must ensure that the file does not include any Trojan Horses. These system policies allow a "controlled" interference of the low sensitivity domain (sometimes called "intransitive noninterference" [18]). That is, a flow connecting two endpoint processes is prohibited except when going through an intermediate filter process.

A typical implementation of these policies in separation kernel and security kernel architectures is to use a "trusted subject," in which the filter process is assigned a security range that spans the confidentiality or integrity range of the endpoint processes. However, this solution has the drawback that the kernel allows the filter process to access all information in both domains. With a Least Privilege Separation Kernel, the kernel can be configured to restrict the interference to a specific subset of the information in each domain, thereby requiring less trust in to be placed the filter process, as shown in Figure 2.

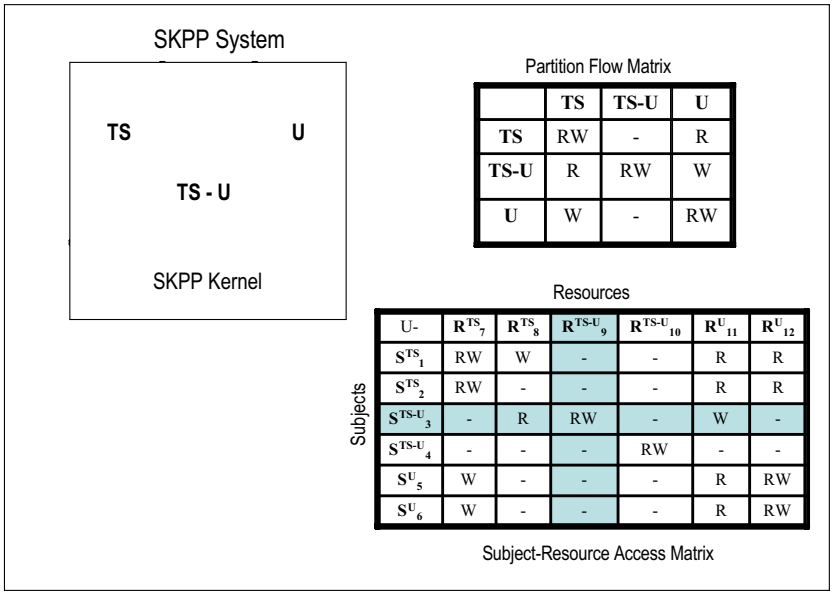

Fig. 2. Kernel-based Strictly-Controlled Interference

5.2 Regraders

Within a block, it may be necessary to perform certain transformations on information that change its security attributes. For example, a guard [3] performs information review and downgrading functions; and a quality assurance manager transforms prototype code into production code by re-grading it in terms of reliability. For each of these transformations there may be several subjects within a block performing various aspects of the task at hand. The principle of least privilege requires that each of these subjects be given only the privileges required to do its particular task and no more.

An example of the application of least privilege separation is that of a "downgrader," (see Figure 3) for re-grading selected information from classified to unclassified. An initiator (UInit) process in A writes selected classified information to a classified holder buffer in Block A. An untrusted copier process moves the contents of the holder to the dirty-word search workspace in Block B. An untrusted dirty-word search process (UDWS) in B provides advisory confirmation that the information is "suitable" for downgrading and copies the information into the clean results buffer (note that this process's actions should be considered "advisory" since it is fully constrained by the mandatory policy enforcement mechanism). Then the trusted downgrader (TDG) program in C reads the information from the clean results buffer and writes it to an unclassified receiver buffer in D where it may be accessed by an unclassified end-point process (UEnd). As constrained by least privilege as encoded in the subject-to-resource flow matrix, the downgrader process in Block C cannot read from any resource other than the clean results and cannot write to any resource in D other than the receiver.

This limits damage in the event of errors, for example in the downgrader, initiator or search processes, and contributes to a substantive argument that only the downgrader program needs to be *trusted* with respect to the application-level multilevel policy (viz., *depended* on to write down only when appropriate), and thus requires security verification with respect to that policy.

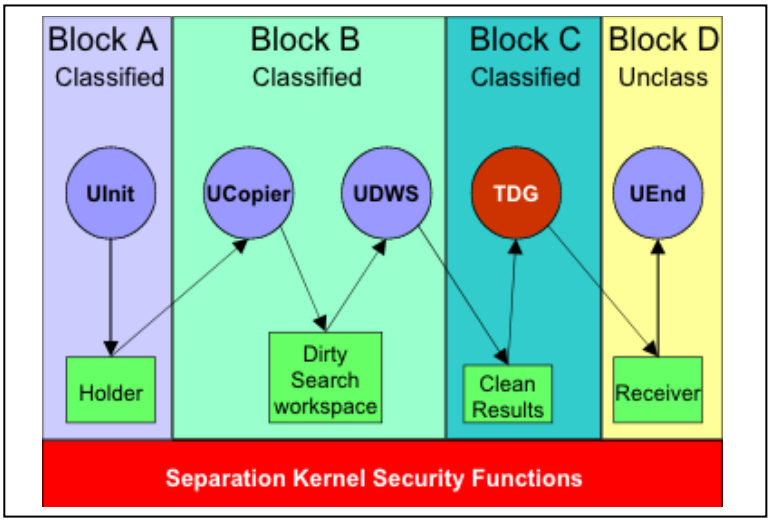

Fig. 3. Trusted Downgrader. Dark areas with white text are trusted.

6 Related Work

6.1 Protection Profiles for Separation Kernels

The Common Criteria security evaluation paradigm includes a document called a *protection profile* that specifies the security functionality and assurance for an entire class of IT products, as well as a document called a *security target*, which provides a similar specification for a specific IT product. The protection profile is evaluated for consistency with the Common Criteria requirements for protection profiles; the security target is evaluated for consistency with the Common Criteria requirements for security targets, as well as for consistency with an identified protection profile (if any); and finally the product is evaluated against the security target.

A forthcoming high robustness protection profile for separation kernels [11]; [15] includes least privilege requirements regarding subjects as well as kernel-internal mechanisms. Several commercial efforts are underway to develop separation kernels to meet this profile, include those at Greenhills, and LinuxWorks [2].

6.2 Trusted Computing Exemplar Project

Separation kernel technology is being applied in our *Trusted Computing Exemplar* project [7]. This ongoing effort is intended to produce a high assurance least privilege separation kernel. The kernel will have a static runtime resource configuration and its security policy regarding access to resources will be based on process/resource access bindings, via offline configuration (e.g., via an access matrix, such as are shown in Figures 1, 2 and 4). The static nature of resource allotment will provide predictable processing behavior, as well as limit the *covert channels* based on shared resource utilization [10]; [9]; [13]. Simple process synchronization primitives will also be provided, that can be implemented to be demonstrably free of covert channels (Reed, 1979). This kernel is also used as the security foundation for the SecureCore architecture [7].

6.3 Type Enforcement Architectures

Bobert and Kain [4] described a "type enforcement architecture" with the capability to provide least privilege at a fine granularity, a form of which is used in the SELinux project [12]. There are currently no high assurance instances of such systems today.

7 Conclusions

The separation kernel abstraction and the principle of least privilege are significant tools for the protection of critical system resources. In this paper, we described a fusion of the separation abstraction with the least privilege principle. In addition to the inter-block flow control policy prescribed by the traditional separation kernels, this approach supports an orthogonal, finer-grained flow control policy by extending the granularity of protected elements to subjects and resources, as well as blocks, in a partitioned system. We showed how least privilege provides assurance that the effects of subjects' activities may be minimized to their intended scope.

In summary, application of the principle of least privilege, resource separation and controlled sharing are synergistic security properties in a separation kernel. Each subject is only given a minimum set of logically separated resources necessary to perform its assigned task, and the sharing of resources between subjects is rigorously controlled by the kernel. A separation kernel that correctly implements these properties can meet the objective to minimize and confine damage with a high level of assurance.

Acknowledgements

We like to thank Michael McEvilley for his helpful comments regarding the history of the principle of least privilege.

References

1. Alves-Foss, J., Taylor, C.: An Analysis of the GWV Security Policy. In: Proc. of Fifth International Workshop on the ACL2 Theorem Prover and its Applications (ACL2-2004) (November 2004)
2. Ames, B.: Real-Time Software Goes Modular. Military & Aerospace Electronics 14(9), 24–29 (2003)
3. Anderson, J.P.: On the Feasibility of Connecting RECON to an External Network. Tech. Report, James P. Anderson Co. (March 1981)
4. Boebert, W.E., Kain, R.Y.: A Practical Alternative to Hierarchical Integrity Policies. In: Proc. of the National Computer Security Conference, vol. 8(18) (1985)
5. Common Criteria Project Sponsoring Organizations (CCPSO). Common Criteria for Information Technology Security Evaluation. Version 3.0 Revision 2, CCIMB-2005-07-[001, 002, 003] (June 2005)
6. Department of Defense (DOD). Trusted Computer System Evaluation Criteria. DoD 5200.28-STD (December 1985)
7. Irvine, C.E., Levin, T.E., Nguyen, T.D., Dinolt, G.W.: The Trusted Computing Exemplar Project. In: Proc. of the 2004 IEEE Systems, Man and Cybernetics Information Assurance Workshop, West Point, NY, June 2004, pp. 109–115 (2004)
8. Irvine, C. E., SecureCore Project. (last accessed April 8, 2006) (last modified April 5, 2006), `http://cisr.nps.edu/projects/securecore.html`
9. Kemmerer, R.A.: A Practical Approach to Identifying Storage and Timing Channels. In: Proc. of the 1982 IEEE Symposium on Security and Privacy, Oakland, CA, April 1982, pp. 66–73 (1982)
10. Lampson, B.: Protection. In: Proc. of 5th Princeton Conference on Information Sciences, Princeton, NJ, pp. 18–24 (1971), Reprinted in Operating Systems Reviews 8(1), 18-24 (1974)
11. Levin, T.E., Irvine, C.E., Nguyen, T.D.: A Note on High Robustness Requirements for Separation Kernels. In: 6th International Common Criteria Conference (ICCC 2005), September 28-29 (2005)
12. Loscocco, P.A., Smalley, S.D.: Meeting critical security objectives with Security-Enhanced Linux. In: Proc. of the 2001 Ottawa Linux Symposium (2001)
13. Millen, J.K.: Covert Channel Capacity. In: Proc of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, April 1987, pp. 60–66 (1987)

14. National Security Agency (NSA). U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness (July 1, 2004), http://niap.nist.gov/pp/draft_pps/pp_draft_skpp_hr_v0.621.html
15. Nguyen, T.D., Levin, T.E., Irvine, C.E.: High Robustness Requirements in a Common Criteria Protection Profile. In: Proceedings of the Fourth IEEE International Information Assurance Workshop, Royal Holloway, UK (April 2006)
16. Preparata, F.P., Yeh, R.T.: Introduction to Discrete Structures for Computer Science and Engineering. Addison-Wesley, Reading (1973)
17. Reed, D.P., Kanodia, R.K.: Synchronization with Eventcounts and Sequencers. Communications of the ACM 22(2), 115–123 (1979)
18. Rushby., J.: Design And Verification Of Secure Systems. Operating Systems Review 15(5) (1981)
19. Saltzer, J.H., Schroeder, M.D.: The Protection of Information in Operating Systems. In: Proceedings of the IEEE, vol. 63(9), pp. 1278–1308 (1975)