Calhoun: The NPS Institutional Archive

DSpace Repository

Center for Cybersecurity and Cyber Operations (C3O) Faculty and Researchers' Publications

2006-04-00

High Robustness Requirements in a Common Criteria Protection Profile

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

John Wiley and Sons, Hoboken, NJ, 2006,

Proceedings of the 4th IEEE International Information Assurance Workshop, Royal Holloway, University of London, UK, April 2006, pp. 66-75.
http://hdl.handle.net/10945/7141

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

High Robustness Requirements in a Common Criteria Protection Profile

Thuy D. Nguyen, Timothy E. Levin, Cynthia E. Irvine
Naval Postgraduate School, Monterey, California
tdnguyen(levin,irvine)@nps.edu

Abstract

The development of a Common Criteria protection profile for high-robustness separation kernels requires explicit modifications of several Common Criteria requirements as well as extrapolation from existing (e.g., medium robustness) guidance and decisions. The draft U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness (SKPP) is intended to be applicable to a class of products (the target of evaluation, or TOE) that includes, but is not limited to, real time and embedded systems. This paper describes certain SKPP concepts and requirements and provides underlying motivations and rationale for their inclusion in the SKPP. Primary areas of focus are the security requirements regarding information flow, dynamic configuration, and the application of the principle of least privilege to restrict actions of active entities.

Keywords: common criteria, separation kernel, high robustness, dynamic configuration, least privilege.

1. Introduction

In the U.S., there has been an increased interest in the use of separation kernels to support real-time embedded systems and virtual machine monitors. A number of products are forthcoming and some of those products will be used in environments requiring high robustness. The effort to create the U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness (SKPP) was started to respond to this need [1]. This protection profile is based on CC Version 2.2 [2].

Since both separation kernels and high robustness are in an uncharted territory of the Common Criteria (CC) landscape, development of a protection profile for this class of products presented several challenges, some of which were:

- Description of the TOE abstractions in the Common Criteria context,
- Extensions to several Common Criteria requirements, and
- Extrapolation from existing guidance and protection profiles for medium robustness environments [3, 4].

Since the initial release for public review in 2004, the SKPP has been substantially revised and is pending release. This paper describes the rationale for certain concepts and requirements that first appeared in the initial version as well as some that are new in the current draft. The general characteristics of a separation kernel and the specific instantiation of those characteristics captured in the SKPP are presented. This is followed by a discussion of some of the high robustness issues currently being addressed for the SKPP. A summary of the on-going transition to CC Version 3.0 is also provided [5]. It is not an objective of this paper to provide a comparative analysis of separation kernel relative to other technologies, e.g., [6, 7, 8, 9, 10].

2. What is high robustness?

The Common Criteria is an internationally-recognized standard for security evaluation of IT products [2, 5]. In the CC framework, a Protection Profile (PP) is a collection of implementation independent security requirements for a class of target products intended to undergo CC evaluation, i.e., a Target of Evaluation (TOE) class. A Security Target (ST), on the other hand, is a set of implementation-specific security requirements for a particular TOE.

The CC defines six building blocks for a PP/ST: TOE description, security environment description, security objectives, security functional requirements, security assurance requirements, and rationale statements on the traceability mapping of the first five building blocks. With respect to assurance, the CC defines a seven-level assurance scale that measures, through independent evaluation, the breadth, depth, and rigor of the security properties of the target system. The evaluation assurance levels (EAL) range from EAL1 (lowest) to EAL7 (highest). Although the CC is internationally accepted, only evaluations at EAL4 and

below are mutually recognized by the participating CC members. Individual members structure CC evaluations with what is called a "national scheme."

In the U.S. scheme, robustness is a metric used to measure the TOE's ability to protect itself and its resources. A PP must declare its degree of robustness [3]. *Robustness* is defined in terms of the strength of the TOE's security mechanisms and its level of assurance [11]. There are three levels of robustness: high, medium, and basic. High robustness is defined as "security services and mechanisms that provide the most stringent protection and rigorous security countermeasures." [12] The need for robustness in a TOE is a function of the value of the data that the TOE protects and the threats identified for the environment in which the TOE is employed. High robustness can counter extremely sophisticated and well-funded threats and therefore can be used to protect high-value data [11].

3. Separation Kernel characteristics

John Rushby introduced the separation kernel concept in 1981 [13] and subsequently discussed it in terms of partitioning kernel in the context of avionics systems [14]. Separation kernel concepts have also been previously discussed with respect to security evaluation [15] and implemented in commercial systems, e.g., PR/SM [16].

Like other security kernels, a separation kernel manages all resources under its control and can protect itself so that it cannot be attacked. A separation kernel divides all resources into partitions representing different policy equivalence classes and controls the interaction between partitions. Controlled interaction means that the actions of a subject in one partition cannot be detected or communicated to a subject in another partition, unless the kernel has established an explicit means for such communication.

Similar to a Type I virtual machine monitor [17], a separation kernel achieves isolation of resources in different partitions by virtualization of shared resources, e.g. devices, memory, and the clock, such that each partition is assigned as a set of resources that appears to be entirely its own. The kernel also creates the abstractions of resources in the form of abstract data types that are exported at the kernel interface.

Another general characteristic is the way a separation kernel manages the runtime resources. Resources are the aggregate of hardware, software and data that are either used for the implementation of the kernel or exported at the kernel interface. Exported resources include subjects, which are active entities and other non-active entities such as memory, files, devices, buffers, volumes, etc.

In addition to these general characteristics, the SKPP requires other characteristics, discussed below.

3.1. Limited security functionality

A SKPP-conformant separation kernel will be subjected to a high level of scrutiny during its evaluation and validation cycles, and thus must be minimized in complexity regarding functionality and design. The core functionalities include the separation and isolation of all resources, the control of information flows between partitions and between resources, the controlled sharing of selected resources, and the management of security functions and audit services. The limited nature of the security functionality allows this type of separation kernels to be used as the trusted foundation of a secure system or as an embedded component within a larger system.

3.2. No runtime user interface

The SKPP is intended to be applicable to embedded as well as other types of systems. For the former, the kernel, once configured, operates autonomously without human intervention. Hence, the SKPP defines no functional requirements for identification and authentication of users, administrative roles support, and the allocation of administrative roles to security management functions. Since there will be no administrators to monitor and maintain the kernel during runtime, additional assurance measures are inherently required to ensure highly robust autonomous execution of the security management functions. Note that it is allowed for the ST author to specify additional requirements for administrative roles in support of a specific kernel implementation.

3.3. Static runtime configuration

In general, the runtime behavior of a separation kernel is determined by its configuration data. The configuration data specifies how resources are allocated for both time and space, and what policy rules are to be enforced. The SKPP supports both static and dynamic configuration. With static configuration, the configuration data is statically defined and maintained during the runtime execution; time and space resource allocations and policy rules cannot be changed after the kernel is initialized.

Because of its simplicity, a completely static separation kernel is ideal for research on the rapid creation of evaluatable high assurance systems. If the configuration does not change during runtime, the kernel design and implementation can be relatively simple and small, making it suitable for high robustness evaluation. The fundamental security

service provided by a statically-configured separation kernel is separation of the activities occurring in different partitions. This enables the kernel to be used as a trusted building block for more complex systems.

However, in some exigent scenarios (e.g., the failure of a peripheral device in a mission critical application), it may be desirable for the kernel to be able to change its security configuration. Thus, the SKPP allows the kernel to provide a capability to change the current configuration during runtime to another, pre-loaded, configuration. Configuration change will be discussed further in Section 5.2.

3.4. Support privileged subjects

For highly robust security systems, the principle of least privilege is a significant factor that, to date, has not been explicitly addressed as a security functional requirement in the Common Criteria. It is vaguely implied in the ADV_INT assurance requirements [2], which could be subject to unsound interpretation. Least privilege prescribes restrictions to actions of active entities including the internal modules and technical measures that comprise the TOE security functions, and the subjects (e.g., application programs) in the TOE scope of control. In the SKPP, the restriction is that these entities must not have any more "privilege" (viz., access to resources) than is necessary to perform the actions for which they were designed. Exceptions to this requirement are allowed for certain degenerate designs. How the principle of least privilege is used in the SKPP is further discussed in Section 5.

3.5. Generate audit records

The SKPP requires the kernel to detect a set of auditable events and capture information that characterizes each event in a form that can be interpreted by an external review mechanism. The kernel is not required to notify the operational environment of the existence of the audit record or to automatically export the information to the environment.

4. SKPP Target of Evaluation overview

As shown in Figure 1, the TOE boundary encompasses the following components:

- TOE Security Functions (TSF)
- Initialization mechanism
- Configuration mechanism
- Trusted delivery mechanism

The CC divides security requirements into two categories, functional and assurance. The security functional requirements are levied only on the TSF while the security assurance requirements are imposed on the entire TOE. The following sections discuss the security requirements included in the current SKPP working draft, which is based on CC Version 2.2.

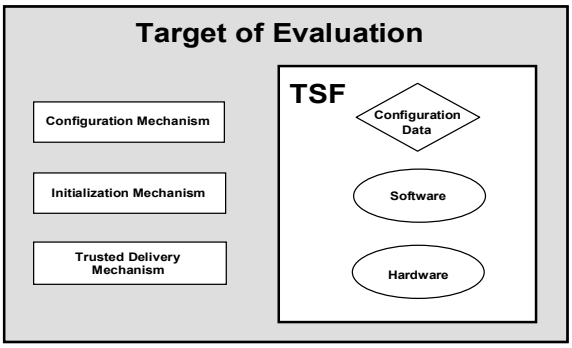

Figure 1. TOE and TSF boundary

4.1. Security functional requirements

The CC organizes security requirements in terms of classes, families and components. The SKPP utilizes six of the eleven CC-defined classes of security functional requirements based on the security objectives identified for the TOE and its environment. Figure 2 summarizes the security functional requirements included in the SKPP at the class and family levels.

Except for two functional classes, Security Audit and User Data Protection, the other four functional classes include extended requirements are that not contained in Part 2 of the CC. These extended requirements are necessary because some of the security objectives identified in the SKPP need requirements that either do not exist in the CC or cannot be directly mapped to the existing CC requirements. In the Identification and Authentication class, the requirements in the FIA_ATD and FIA_USB families are modified to require that the configuration data must specify the security attributes of resources, and that the TSF binds these security attributes to exported resources when the resources are created during initialization.

In the Security Management class, the requirements in the FMT_MOF, FMT_MSA, and FMT_MTD families are modified to require the TSF to restrict access to privileged operations to authorized subjects, to assign authorizations to subjects as specified by the configuration data, and to protect the integrity of the configuration data at all times.

The Protection of the TSF class (FPT) includes five new families and two modified families, FPT_FLS and FPT_RCV. The new families contain requirements

regarding the run-time configuration change capability, establishment of secure state, application of the principle of least privilege on TSF internal functions and modules, secure halt, and secure restart. The FPT_FLS and FPT_RCV requirements are modified to explicitly define the requirements for preservation of secure state and the TSF responses to security failures.

The last extended security functional requirement belongs to a new family in the Resource Utilization class. This requirement addresses the objective that the TSF exhibits predictable utilization of both time and space resources.

achieve a high level of confidence that the TOE initialization mechanism will correctly establish the TSF in its initial secure state, we extended requirements in the Development class to precisely state both functional and assurance requirements for the initialization mechanism. In addition to establishing the initial secure state, the initialization mechanism must also maintain the integrity of the TSF during the initialization process and not interfere with the TSF after the initialization process completes.

Similarly, it is important to levy both functional and assurance requirements on the configuration data

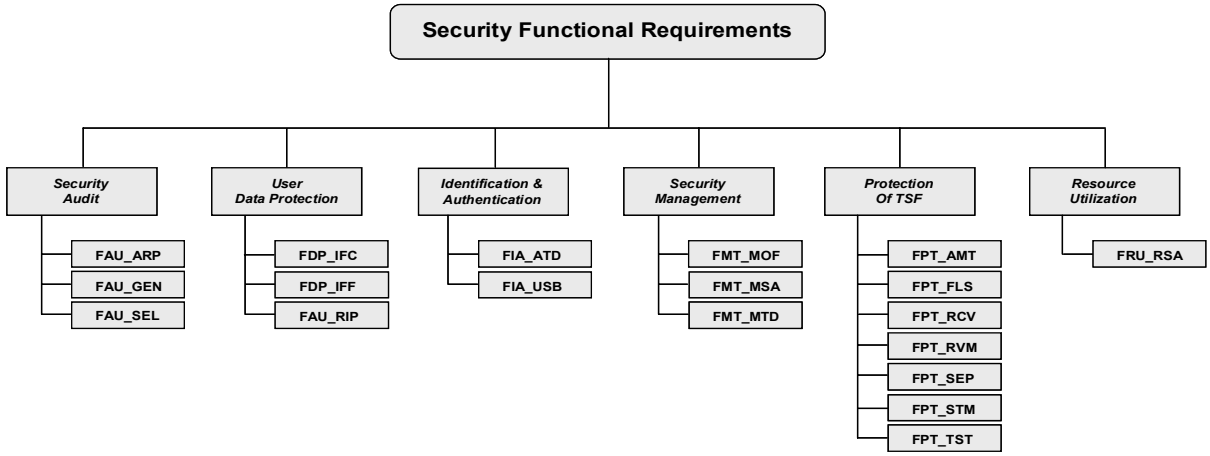

Figure 2. Security functional requirements in SKPP

4.2. Security assurance requirements

The evaluation target of the SKPP-conformant TOE is EAL6 augmented with a formal security policy model and additional assurance requirements related to development, flaw remediation and independent testing. Having a formal model ensures that the security policy implemented by the separation kernel is sound and self-consistent.

Extended security assurance requirements are added to the Development assurance class (ADV) to address the security objectives for the soundness of the architectural design of the TSF with in-depth analysis of the FPT_SEP and FPT_RVM requirements, trusted initialization, and trusted configuration data generation and validation. Furthermore, the ADO_DEL family in the Delivery and Operation assurance class is also modified to require a delivery capability that uses cryptographic services to protect the TOE against modifications and masquerading attempts during the TOE delivery to the users.

In the CC framework, security functional requirements are levied on the TSF, not the TOE. To

generation and validation mechanism because the TSF depends on the correctness of the configuration data to establish the TOE security policy. This mechanism must preserve the semantics of the configuration data in both human and machine readable forms. Regarding assurance requirements, it is also subjected to the same level of analysis and testing as other non-TSF TOE components.

5. SKPP high robustness issues

Notable among the challenges that have surfaced during the development of the SKPP are the application of the principle of least privilege to provide a finer grained information flow control, dynamic configuration change during run-time, and hardware assurance. These issues are critical aspects of assurance in a highly robust TOE.

5.1. Principle of least privilege

The traditional separation kernel approach only addresses resource isolation and ignores the problems relating to all-or-nothing security within an individual

Table 1. Authorization Matrix for the Least Privilege Abstraction [1]

		Partition A	Partition A Resources			Partition B	Partition B Resources			Partition C	Partition C Resources			
			R1	R2	R3		R4	R5	R6		R7	R8	R9	R10
Partition A		**RW**				**R**				**W**				
Partition A Subjects	**S1**		RW	R	W		-	R	-		W	-	W	-
	S2		W	RW	R		R	-	R		W	W	-	W
	S3		R	-	RW		R	R	-		-	W	W	-
Partition B		**-**				**-**				**-**				
Partition B Subjects	**-**		-	-	-		-	-	-		-	-	-	-
	-		-	-	-		-	-	-		-	-	-	-
	-		-	-	-		-	-	-		-	-	-	-
Partition C		**-**				**RW**				**R**				
Partition C Subjects	**S4**		-	-	-		RW	R	W		R	R	-	R
	S5		-	-	-		W	RW	-		-	R	R	R
	S6		-	-	-		R	W	RW		R	-	-	R

partition. Over-privileged programs are left for application developers and evaluators to resolve. For highly robust systems, all-or-nothing security is a significant issue.

Saltzer and Schroeder concluded that least privilege is a design principle that can limit the damages caused by both programmatic and operational errors [18]. They defined least privilege as the restriction that "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." In other words, mechanisms and subjects should have no more privilege than what is necessary to perform the actions for which they were designed.

Several benefits accrue as a result of the application of the principle of least privilege. First, should a component become corrupted, the damage resulting from its failure will be less pervasive. Second, because the privileges afforded the component will be minimal with respect to the overall policy, security analysis of the component is less complex. Finally, 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. Thus, the ability of a secure system to realize the goals of accountability and the confinement of damage will be directly correlated with the level of granularity with which the system is able to invoke the principle of least privilege.

For the SKPP, the principle of least privilege is used in two different contexts. Externally, it is used by the TSF to restrict access by subjects to privileged instructions and resources within a partition. Internally, it is applied to the structure of the TSF to restrict privileges of internal modules and functions.

As a consequence, 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 the job. Hence, a high robustness 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).

5.1.1 Least privilege for flow control

The SKPP defines three different TOE abstractions based on the granularity with which subjects and resources are exported by the TSF. This enables the SKPP to support different use cases of information flow control with varying granularity of control over subject-resource interaction. The ST author is required to select the abstraction that corresponds best to the TSF.

The *least privilege abstraction* is the most fine-grained use case. It requires the TSF to be capable of

supporting multiple heterogeneous subjects and resources in a partition and that to discern between those subjects and resources for the purpose of information flow control. Specifically, the TSF must enforce the information flow control policy in terms of both partition-to-partition flows and subject-to-resource flows. The enforcement rules as paraphrased from the FDP_IFC and FPD_IFF requirements are as follows:

TSF shall allow an operation of a subject on an exported resource only if:

- *The partition-to-partition flow rule explicitly authorizes the operation and*
- *The subject-to-resource flow rule explicitly authorizes the operation*

A sample authorization matrix to illustrate the SKPP least privilege abstraction is depicted in Table 1. Ten exported resources are mapped to partitions, where each exported resource can be allocated to one and only one partition. Similarly, six resources are individually mapped to one and only one partition. No subjects have been mapped to Partition B. The access authorizations of each subject to each resource appear in the cells of the matrix. The authorizations of a set of subjects that are mapped to the same partition can be heterogeneous with respect to both resources in the subjects' own partition as well as to resources in other partitions.

Since the SKPP is intended to be applicable to different implementations, the FDP_IFC and FPD_IFF requirements are structured to support two other use cases where subject-resource interaction is "policy-equivalent" to that of their partition (s).

The second use case is a degenerate configuration where there is only one subject in the subject's partition and only one external resource in the resource's partition. The TSF effectively enforces information flow rules at the granularity of the flows between partitions. This constitutes the *partition abstraction.*

In the last use case, the TSF supports multiple subjects and resources in a partition. However, the resource-access needs of all subjects of a given partition are homogenous and so the TSF treats them all equally with respect to their authorizations to access the exported resources in a given "resource" partition. As with the flow control policy, the TSF does not discern between exported resources in a particular partition. For this *equivalence set abstraction*, the subject-to-resource flow rules are at the granularity of the flows between and within partitions.

To make the equivalence set abstraction more concrete, consider the following example of a system with three partitions: P_X, P_Y, and P_Z. Each partition P_n contains a set of resources R_n, and a distinct set of subjects S_n. The sets may be empty. In the equivalence set abstraction all subjects in a given partition P_n will have the same authorizations, A_{nm} to all resources in another partition, P_m. (where m may equal n). For example, for the resources in P_Y, all subjects in S_X will have the authorization A_{XY}. Similarly for P_Z, they will have authorizations A_{XZ}. For the subjects associated with P_Z, there will be similar sets of authorizations. The access matrix for this arrangement is shown in Table 2.

Table 2. Authorization Matrix for the Equivalence Set Abstraction

		Partition Resources		
		R_X	R_Y	R_Z
Partition Subjects	P_X S_X	A_{XX}	A_{XY}	A_{XZ}
	P_Y -	-	-	-
	P_Z S_Z	A_{ZX}	A_{ZY}	A_{ZZ}

5.1.2 Least privilege for architectural assurance

Although legacy evaluation criteria imposed assurance requirements regarding the application of the principle of least privilege to restrict the actions of internal modules and functions [19], the CC lacks such explicit requirements. When applied to the design and internal structure of the TSF, least privilege can aid reasoning about the assurance properties of the system. Placing limits on the privileges that a module can have helps reduce the complexity of the implementation. Layering, minimization, and least privilege, when used in combination, enable a greater degree of scrutiny, which will increase the confidence in the analysis of the correctness of the TSF. These concepts are closely tied to, but conceptually different from, the notions of modularity and encapsulation [20].

In the SKPP, both functional and assurance requirements are extended to require developers to apply the principle of least privilege to all TSF (i.e., internal) modules and functions. An element is added to the FPT class that limits internal functions and modules of the TSF to minimal scope and accessibility to resources. A parallel element is added to ADV_INT to address the related assurance requirements.

5.2. Dynamic configuration

To support special cases where it is desirable for the TOE to be able to change its configuration (e.g., the occurrence of overriding environmental security conditions), the SKPP allows the TOE to change resource allocations and policy rules during runtime. Dynamicity and complexity go hand-in-hand and there are problems associated with dynamic configuration that the TOE developers and evaluators must address.

First, the necessity to provide a coherent security policy, as reflected in the model, is always an issue for secure systems. Arbitrary changes to the configuration data are hard to understand with respect to policy since any changes would have to be evaluated against the policy to make sure they are still correct. This represents more work, more complexity, and more opportunity for errors. With dynamic reconfiguration of the security policy itself, unless done very conservatively, the analysis of the policy and model may be very difficult – for example, ensuring that secure state is maintained across a policy transition, and that the changes do not cause undefined combinations of policies after the transition. The difficulty of analyzing arbitrary policy mechanisms is known to present an undecidable problem [21]. For the SKPP, the vendor must convince evaluators of the adequacy of their solution.

The SKPP approach to dynamicity is to limit how the policy may change by defining the following hierarchy of configuration changes that a TOE can provide during runtime [22]:

- Static configuration change
- Constrained dynamic configuration change
- Unconstrained dynamic configuration change

Runtime configuration change capabilities are optional. The characteristics and assurance issues of each capability is discussed in the following sections.

It is important to note that a conformant kernel must provide the capability to change the TOE configuration while the TOE is offline. This requires the TOE to access a *vector* of configuration attributes when it initializes. This configuration change capability allows for a complete removal of previous security state as well as pre-analysis of subsequent security policies. The assurance issue associated with this capability is to ensure that each configuration used represents the intended organizational security policy and operational conditions.

5.2.1 Static configuration change

This capability is the most basic form of dynamicity — it requires the TOE to be able to load multiple configuration vectors at initialization, one of which will define the *current* or operational configuration after the completion of initialization. The TSF must also provide a mechanism for an authorized subject to select the *next* vector. i.e., the one that will become operational when the TOE is next initialized.

Alternatively, the TSF may provide a mechanism for the transition from current to next configuration to take effect during runtime, i.e., without reinitialization. The TSF must restrict to authorized subjects the ability to select the next configuration and to cause the configuration transition.

This capability requires the TOE to properly execute the requested configuration change as specified by the authorized subject. When a runtime change is invoked, the TSF has the additional requirement to continuously maintain secure state before, during and after the configuration change. These concerns are in addition to the assurance issue associated with the offline configuration change capability. Other security characteristics such as those relating to covert channels must also be preserved during and after the configuration change.

5.2.2 Constrained dynamic configuration change

This capability allows ad hoc runtime configuration changes that are constrained by static rules defined either in the configuration data or in the TSF itself. Depending on the comprehensiveness of the rules (i.e., the range of allowed changes), the authorized subject that performs these changes may be an extension of the TSF because is shares the responsibility to properly enforce the policy.

The additional assurance requirement associated with this option is to ensure that the constraints and the ad hoc policy change requests are consistent with the organizational security policy. Furthermore, the level of trustworthiness of the authorized subject that is allowed to invoke this capability must be commensurate with the level of trustworthiness vetted in the TSF.

5.2.3 Unconstrained dynamic configuration change

This capability affords an authorized subject the flexibility to arbitrarily change any and all aspects of the current configuration. Unlike the constrained dynamic configuration change, this capability has no constraints. The lack of constraints exacerbates the difficulty in the preservation of secure state before, during, and after the configuration change, and indicates that the authorized subject is an extension of the TSF. Hence, the additional assurance challenge associated with this capability is to provide a convincing definition of “secure transition” in the security policy model.

Since a configuration change potentially alters the security policy of the system, the formal security policy model must represent the change from one policy to the next, and the model proof must show that this transition preserves security. The chain of evaluation evidence linking the system implementation to the model will then ensure that the system preserves security during the configuration change.

Given the issues related to dynamic configuration discussed previously, the moderate and full dynamic configuration change capabilities are considered beyond the scope of the SKPP and will require a ST-based rather than a PP-based evaluation. In this case, the ST must define the functional and assurance requirements for the supported capability, and justify that the core security objectives including the ability to transition to a secure state without being compromised are continuously satisfied during the course of the configuration change.

5.3. Hardware assurance

Since the time of the Trusted Computer System Evaluation Criteria [19], it has been generally understood that, for high robustness, hardware must be considered as part of the TSF. Karger and Kurth had argued that, for high assurance composite evaluation, the ETR-lite concept vetted by the European CC certification agencies is not adequate and that additional information sharing among the hardware developers, software developers, and the various evaluators is critical to the success of a high assurance evaluation [23].

A hindrance to hardware verification is the fact that the hardware vendors do not generally share design documents with their customers, due to proprietary concerns. Thus it is difficult for TOE vendors to produce detailed design and assurance documentation regarding TOE hardware components. Were those documents available, given the abilities of modern hardware design tools (e.g., to organize and document a layered, modular design), many, if not all, high robustness ASRs could be applied in the evaluation of TSF hardware components [24].

The threat of targeted hardware subversion has been a long-standing concern in the development of highly secure systems [25]. In particular, once a given hardware component for the TOE has been selected, this threat, can, in the current environment, be addressed through a requirement to never upgrade hardware (or development tools, for that matter) after the initial TOE design. Such a requirement would be prohibitively expensive.

Descriptions of the TSF's dependencies on the hardware and how the hardware security mechanisms are used correctly in the security architecture of the TSF, is probably the best possible approach in this era. These descriptions should be at a level of detail and rigor that is consistent with the TOE's evaluation level.

In the current version of the SKPP, hardware assurance is still unresolved although we have proposed a set of assurance requirements that is under review by the funding organization. The new composition class, ACO, in CC Version 3.0 was used as a model for these proposed requirements.

6. CC Version 3.0 transition issues

The SKPP was developed based on CC Version 2.2. A preliminary effort to transition the SKPP to CC Version 3.0 was conducted. Hardware assurance is still not addressed in the new version. Some of the other more notable challenges are discussed here.

6.1.1 Information flow policy enforcement

We found that CC Version 3.0 is significantly different, but from our initial investigation, the FDP_ACC requirements in Version 3.0 appears to be simpler to use than the combined FDP_IFC and FDP_IFF classes with respect to expressing information flows rules. In FDP_ACC, the three classes can be presented more simply. This straightforward construction can make the requirements more understandable.

6.1.2 Security attributes binding

The SKPP requires the binding of security attributes to exported resources when the resource is created.

The current thought is to define a two-step process. Registration allows the kernel to use the configuration data to determine the attributes of the exported resources that exist during runtime and to keep track of that information in the kernel data structure. The binding of the security attributes obtained during registration takes place during the Initialization step. This is when the exported resource is actually created. The FIA_URE and FDP_ISA families are being considered for this approach. However, more investigation is still needed in this area.

6.1.3 Covert channel analysis by evaluators

In CC Version 3.0, covert channel analysis is handled rather obliquely, although the TOE developer is required to conduct the CCA via ADV and ATE with respect to the unobservability functional requirement. Since the functional requirement says, in essence, that there will be no covert channels, it follows that the ADV and ATE will ensure that that is

the case, amounting to "covert channel analysis," with the participation of both the vendor and the evaluators. Unfortunately, the lack of explicit wording could lead to a variety of interpretations and consequent confusion regarding the responsibility for CCA.

These issues should be addressed in subsequent versions of the CC.

7. Future work and conclusion

Robustness is "a characterization of the strength of a security function, mechanism, service or solution, and the assurance (or confidence) that it is implemented and functioning correctly." [12] High robustness provides the highest level of confidence in the implementation of security services and mechanisms. The creation of a protection profile based on the Common Criteria that addresses high robustness issues is uncharted territory. Although the Common Criteria provides a framework for describing very rigorous requirements at its highest evaluation assurance levels, we found that several extensions were imposed by the need to articulate high robustness for a separation kernel. Additionally, several new requirements were identified, including those for both external and internal least privilege, and dynamic configuration. For the former, the TSF must be able to restrict access by subjects to privileged instructions and resources within a partition. In the case of the latter, least privilege is applied to the structure of the TSF itself, restricting privileges of internal modules and functions.

The protection profile contains a set of configuration change requirements ranging from manual reconfiguration to a highly dynamic option. That range is paralleled by increasing complexity of the TSF and may motivate innovative assurance arguments.

Acknowledgements

We would like to express our appreciation to the entire SKPP authoring and management teams for their support of the SKPP effort and especially Michael McEvilley for his technical insights.

References

[1] U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, Version 0.621, National Security Agency, 1 July 2004.

[2] Common Criteria for Information Technology Security Evaluation, Version 2.2, CCIMB-2004-01-[001, 002, 003], Common Criteria Project Sponsoring Organizations, January 2004.

[3] Consistency Instruction Manual For Development of US Government Protection Profiles For Use in Medium Robustness Environments, Release 3.0, National Security Agency, 1 February 2005.

[4] U.S. Government Protection Profile for Multilevel Operating Systems in Environment Requiring Medium Robustness, Version 1.68, National Security Agency, 09 February 2004.

[5] Common Criteria for Information Technology Security Evaluation, Version 3.0 Revision 2, CCIMB-2005-07-[001, 002, 003], Common Criteria Project Sponsoring Organizations, June 2005.

[6] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A., "Xen and the art of virtualization," Proceedings of the 20th ACM Symposium on Operating System Principles, October 2003, pp. 164–177.

[7] Schell, R.R., Tao, T.F. and Heckman M., "Designing the GEMSOS Security Kernel for Security and Performance," Proceedings of the 8th National Computer Security Conference, 30 September - 3 October 1985, pp. 108-109.

[8] Shockley, W.R., Tao, T.F. and Thompson, M.F., "An Overview of the GEMSOS Class A1 Technology and Application Experience," Proceedings of the 11th National Computer Security Conference, October 1988, pp. 238-245.

[9] Schellhorn, G., Reif, W., Schairer, A., Karger, P., Austel, V. and Toll D., "Verification of a Formal Security Model for Multiapplicative Smart Cards," 6th European Symposium on Research in Computer Security (ESORICS 2000), October 2000, Lecture Notes in Computer Science, vol. 1895, Springer-Verlag, pp. 17-36.

[10] Bugnion, E., Devine, S., Govil, K. and Rosenblum, M., "Disco: Running commodity operating systems on scaleable multiprocessors," ACM Transactions on Computer Systems, vol. 15, November 1997, pp. 412–447.

[11] Information Assurance Technical Framework, Chapter 4, Release 3.1, National Security Agency, September 2002.

[12] Department of Defense Instruction, Number 8500.2, February 6, 2003.

[13] Rushby. J., "Design And Verification Of Secure Systems," Operating Systems Review, 15(5), 1981.

[14] Rushby, J., "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance," Technical Report, Computer Science Laboratory, SRI International, March 1999.

[15] A Proposed Interpretation of the TCSEC for Virtual Machine Monitor Architectures, Vol. 1: Strict Separation,

Draft of 1 May 1990, Trusted Information Systems, Inc., Glenwood, MD.

[16] Certification Report for Processor Resource/ System Manager (PR/SM) for the IBM eServer zSeries 900, SIDSZ-CC-0179-2003, 27 February 2003, Bundesamt für Sicherheit in der Informationstechnik: Bonn, Germany. http://www.commoncriteriaportal.org/public/files/epfiles/0179a.pdf

[17] Goldberg, R., "Architectural Principles for Virtual Computer Systems," Ph.D. dissertation, Harvard University, Cambridge, MA, 1972.

[18] Saltzer, J. H., and Schroeder, M. D., "The Protection of Information in Operating Systems," Proceedings of the IEEE, vol. 63, no. 9, September, 1975, pp. 1278-1308.

[19] Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, National Computer Security Center, December 1985.

[20] Parnas, D. L., "On the criteria to be used in decomposing systems into modules," Communications of the ACM, 15(12):1053–1058, 1972.

[21] Harrison, M., Ruzzo, W. and Ullman, J., "On Protection in Operating Systems," Communications of the ACM, vol. 19, no. 8, August 1976, pp. 461-471.

[22] McEvilley, M., Private communication, November 2005.

[23] Karger, P. A. and Kurth, H., "Increased Information Flow Needs for High-Assurance Composite Evaluations," Proceedings of the Second IEEE International Information Assurance Workshop, April 2004, pp. 129-140.

[24] Schell, R. R., Private communication, November 2005.

[25] Karger, P. A. and Schell, R. R. "Multics security evaluation: Vulnerability analysis," Tech. Rep. ESD-TR-74-193, Vol. II, Information Systems Technology Application Office Deputy for Command and Management Systems Electronic Systems Division (AFSC), Hanscom AFB, Bedford, MA 01730, 1974.