An Analysis of the GWV Security Policy

Jim Alves-Foss and Carol Taylor

Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 83844
{jimaf, ctaylor@cs.uidaho.edu}

Abstract

The use of formal models of security policies are required for high assurance security systems. One benefit of formal methods is that it allows for a precise presentation of items, allowing for analysis by others and subsequent discussion. In this paper we examine the presentation and use of the formal security policy developed in ACL2 as presented by Greve, Wilding and Vanfleet in 2003. We found that the ACL2 model and corresponding textual description left some points ambiguous. We clarify these points in this paper.

1 Introduction

The need for computer systems that operate safely and securely has never been greater. Society increasingly relies on computers for a wide range of services from simple communication to delivery of electricity, water and other essential services. Yet, the specification and verification of nonfunctional system properties such as safety or security is not straightforward and a number of methods have been proposed for insuring that computers exhibit desired behavior (safety) plus exhibit no behavior that is not specifically allowed (security).

The process of evaluating software for nonfunctional properties involves assessment that the software demonstrates the stated properties. For military systems and avionics systems on commercial aircraft, assessment is typically accomplished through external review and certification.

For software avionics systems, the FAA is the certifying authority. They have adopted a document, DO-178B [RTCA92], that specifies how software should be developed that is part of a commercial aircraft. While it is not a requirement that avionics system software follow DO-178B, developers must demonstrate to the FAA that their software meets the same high level of development criteria if they choose an alternate development process.

In guaranteeing security, the security community has adopted the Common Criteria (CC) [Nis99] as its assurance standard. The CC ensures product security through independent verification against seven predefined assurance levels, Evaluation Assurance Levels (EAL1 – EAL7). At the highest assurance levels, EAL6 and 7, formal specification of a product's security policy is required. The high level design of the target component must then be mathematically proven to satisfy the formal policy specification.

In this paper, we present a clarification of a security policy modeled in ACL2 presented by Greves, Wilding and Vanfleet [GWV03]. The security policy presented in GWV models a separation kernel, which enforces partitioning between applications running on a single processor system. Separation is a concept that was introduced by John Rushby as a way to build and specify secure systems [Rus81]. A separation kernel forms the basis for an architecture that handles Multiple Independent Levels of Security (MILS). MILS[1] is an on-going collaborative effort between government, academia and industry to build a high assurance architecture for real-time embedded systems. The MILS architecture is a three-tiered architecture consisting of a separation kernel, middleware and applications where each layer enables the higher layers to enforce their own security policies [ATO04]. The overall security policy of the MILS architecture is one of data isolation and information flow in order to support data and applications at different classification levels.[2]

Implementations of a separation kernel will need to be certified at a CC level of at least EAL 6. Since certification at that level requires a formal mapping between the separation security policy and an implementation, it is important that the separation security policy used in the certification be clearly stated so that the full intent of the policy is supported by the implementation. While the GWV policy clearly defines the basic separation axioms of non-exfiltration and non-infiltration[3], other concepts involving definitions of state and the active partition are somewhat ambiguous. We found that a reader of the original GWV paper could possibly misapply the presented policy. This paper is an attempt to prevent such misuse.

In our discussion of the GWV policy, we present several underlying concepts that were left out of the original paper but are important to understanding how the policy enforces separation. The main benefit of a separation kernel is the control of direct communication between applications running in separate partitions. In military systems another concern is implicit information flow via covert channels. Non-interference is a security policy that enables assessment of covert channels. A covert channel is when implicit signaling occurs between users such as through a shared resource or as a result of process timing. Showing that a separation kernel prevents covert channels requires that the separation policy be at least as strong as non-interference. Non-interference was originally proposed in the 80's by Goguen and Meseguer [GM82], and states that if two users are non-interfering then actions by one user do not affect the outputs seen by the other user. We include a proof that GWV maps to noninterference.

The paper is organized into six sections. In section 2 we review the GWV policy as presented in [GWV03]. Section 3 discusses the underlying assumptions and clarifies the intent of GWV. Support for noninterference by GWV is presented in section 4.

[1] Information about MILS can be found in [ATO04].

[2] Current emphasis in the MILS architecture is support for traditional government classification levels such as unclassified, secret, classified and top secret. However, MILS can also support commercial security classifications.

[3] Non-exfiltration indicates that an executing partition will not influence memory segments outside of its permitted set of segments, and non-infiltration indicates that the executing partition can only use information from its permitted set of segments to affect its execution behavior.

Limitations of the GWV policy are discussed in section 5 and we conclude the paper in section 6.

2 The GWV Policy

In this section we provide an overview of the GWV policy and restate the policy function definitions. The GWV policy formally specifies a policy that will be used in certifying a separation kernel implementation. A separation kernel implementation must demonstrate that it correctly provides application separation on a single processor system. In general, a security policy is deliberately written abstractly in order to capture the main ideas of the policy without system specific details. The goal is to create a policy specification that can be re-used for multiple implementations.

2.1 GWV Definitions

An actual separation kernel supports multiple partitions each containing a fixed memory size. The following functions define a model of multiple partitions, with memory segments permanently assigned to partitions. The *current* function returns the identifier of the current partition given a machine state as input.

((current *) ⇒ *)

Associated with partitions are a fixed number of memory segments. The memory segments are assigned names and can be distinguished from each other. The function *segs* accepts a partition name as input and returns a list of segments associated with the partition.

((segs *) ⇒ *)

There are also values associated with a memory segment given a particular machine state. The function *select* accepts two inputs, a memory segment and a machine state and returns the value associated with a memory segment in that state.

((select * *) ⇒ *)

A separation policy requires constraints on direct communication between system entities. Entities will be allowed or denied communication according to a policy based on government classification level or some other grouping scheme. The communication policy will be enforced by a separation kernel. GWV models these communication constraints through a function, *direct interaction allowed* (*dia*), which represents the set of memory segments that are allowed to communicate to the specified segment.

((dia *) ⇒ *)

Another function necessary to model a state based policy is *next* which accepts as input a machine state and returns the next machine state representing one step of computation.

((next *) $\Rightarrow$ *)

GWV defines three more functions that are used in their policy model. *Selectlist* accepts a memory segment list and returns a list of values associated with the segments. *Segslist* takes a list of partitions and returns the memory segments mapped to that set of partitions. *Run* accepts an initial machine state, *st* plus a number, *n,* representing the number of computation steps and returns the machine state after execution of *n* steps. These functions are defined as follows:

```
(defun selectlist (segs st)
   (if (consp segs)
      (cons
          (select (car segs) st)
          (selectlist cdr segs) st))
    nil))

(defun segslist (partnamelist)
    (if (consp partnamelist)
        (append
            (segs (car partnamelist))
            (segslist (cdr partnamelist)))
     nil))

(defun run (st n)
   (if (zp n)
       st
     (run (next st) (1 – n)))
```

The GWV policy is shown in Figure 1, expressed in ACL2 [KMM00].

```
(let ((srcsegs (intersection-equal (dia seg) (segs (current st1)))))
    (implies
     (and
      (equal (selectlist srcsegs st1) (selectlist srcsegs st2))
      (equal (current st1) (current st2))
      (equal (select seg st1) (select seg st2)))
     (equal
      (select seg (next st1))
      (select seg (next st2))))))
```

Figure 1. GWV Separation Policy in ACL2

GWV is modeled at the level of memory segment interactions. It is a state based model where the advancement of system state is through the *next* function. In their paper, the authors state that the advancement to the next state is a single step of computation. The policy says that for any given segment, *seg*, the values of the segment are only affected by memory segments that are allowed to communicate with it and that are part of the currently executing partition. If separation is preserved, then the only apparent way that a given segment could change is from interaction with segments that are allowed to affect it and that are in *dia* (*seg*).

3 Clarification of the GWV Policy

Since the GWV policy represents the formal policy description used in certifying a separation kernel, it must be clearly written with little chance of misinterpretation. Unfortunately, as presented in [GWV03], the policy is vague in several areas and has unstated assumptions. In this section we define precisely what GWV is saying and discuss underlying concepts critical to understanding the policy.

Three concepts in need of further clarification include the *current* partition, the *next* state functions and the mapping of segments to *segs* and *dia* functions. As part of our research with the MILS architecture, we compared GWV with other security information flow policies. The mapping between GWV and other policies highlighted the interpretation difficulties. One problem we encountered was how to define the content of the currently executing partition. An executing process usually has an associated program counter and stack. These are typically stored in locations allocated by the operating system and they must be accounted for in a model of interacting memory segments. Yet, no mention was made of these details in the GWV paper. Another question concerns the *next* function. Does next include one instruction or multiple instructions and what is a precise definition of the state argument, *st* which is input to *next*? Our final questions include the intent of the *segs* and *dia* functions. What restrictions, if any, are placed on these functions? The authors of GWV were contacted and provided details which were not included in the original paper. Highlights from the discussion are presented in the following paragraphs.

3.1 Current Partition State

The separation policy is to be applied at what is called a *cut point*. A *cut point* is the point in the execution of the separation kernel where the previous partition microprocessor state has been saved to a kernel *partition save area.* At the *cut point,* the next partition has not been loaded into the microprocessor and all partition microprocessor saved states are stored in the *partition save area* inside of kernel-protected memory. At this point there is no partition-specific microprocessor state. The program counter, registers, and other microprocessor-specific information is stored in memory segments for the specific partition. Therefore, it is appropriate for the GWV security policy to not address the microprocessor state, a concern that was not addressed in the original paper. However, the placement of these values in a segment, require that this segment be addressed appropriately by the *dia* and *segs* functions (see Section 3.3).

3.2 The Next Function

At the *cut point,* the next partition to be executed is called the *current* partition. When we execute the function, *next*, several steps are performed. First, the saved microprocessor state of current is loaded into the microprocessor. *Current* is then executed until a partition event occurs which has been termed, *run-until-partition-event (rupe)*. At the partition-event, the state of the microprocessor is saved back into memory and finally the microprocessor is sanitized of any user information within the microprocessor (Figure 2).

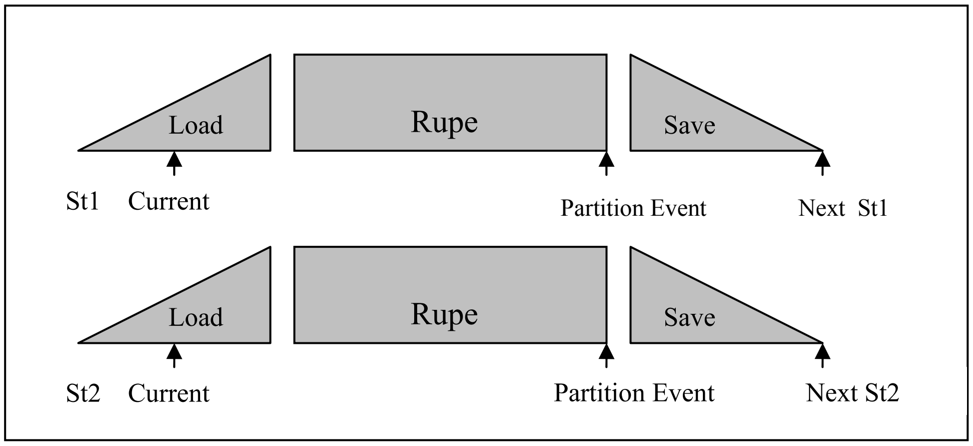

Figure 2. Current partition load, save and execution in St1 and St2

There is no requirement that the *next* function be a single microprocessor instruction or a set of instructions. In the *cut point* model of the world, it is more realistic to assume that the *next* function implements many microprocessor instructions. However, this assumption forces an additional implicit assumption, that externally visible changes to the state between cut-points will not be security relevant. A classic security bypass attack is to violate a security policy temporarily and then fix it before a check. Here that would refer to a write to an I/O memory segment incorrectly and then write over it correctly. A system that did this would not violate GWV, but could violate the intent of the security policy.

3.3 DIA and SEGS functions.

According to GWV, the *segs* function refers to the memory segments of a partition. Specifically, this means the memory segments that are readable by code in a partition. These are the segments that include any data that affects a partitions behavior, including code segments and the saved state segments.

The *dia* function is the instantiation of the security policy in the separation kernel. If *seg1* is not a member of *seg(seg2)* then data values stored in *seg1* are not allowed to influence the change in the values in *seg2*. What does this mean? This means that the following must hold:

- If *seg0* is the memory segment corresponding to the portion of the kernel partition save area of partition *A*, then *segs(A)* should be equal to *dia(seg0)* and *seg0* is in *dia(seg)* for every segment*, seg*, where A is permitted by the policy to modify the contents of *seg*. This prevents other partitions from accessing (reading or writing) this partition-specific save space. The only exception may be a special purpose partition (such as a loader) that will be in *dia(seg0)* so that it can initialize the partition.
- If the policy states that partition *B* can modify segment *seg* in *segs(A)*, then *segs(B)* must be a subset of *dia(seg).* We place this restriction since commodity microprocessors can not follow the flow of information from a source segment, through registers, to a destination segment. Therefore, if a partition can use information from one segment to modify *seg* then we allow it to use information from any of the segments it can access to modify *seg*. One caveat with this is that a multi-level secure (MLS) partition can not use the kernel to help it prevent internal violations of the security policy (e.g., the kernel will not prevent a MLS partition from copying top secret data to a secret segment). This is acceptable since the intent of the MILS architecture is to use the kernel to *support* the application level security policy, but not to fully implement it. Separation within an MLS partition needs to be enforced by the code executing within that partition.

3.4 GWV Non-interference view of the world

In the GWV separation policy, we examine different hypothetical universes, *st1* and *st2* with respect to a particular memory segment *seg.* If these universes have the same current partition and, if the memory segments of current that can interact with *seg* contain the same values, and if *seg* itself is the same in both states, then we can deduce the conclusion of the implication in this theorem. The conclusion is that the values of memory segments that can not interact with *seg* are not relevant to the value of *seg* after the step, in other words they can not interfere.

The policy looks at the state of memory at a *cut point,* steps the microprocessor in two different universes, st1 and st2, and then looks at the resulting state of memory again. All actions of the security policy are with respect to before and after cut points. When executing a *run-until-partition-event,* multiple instructions will be executed.

Between the two universes, *st1* and *st2*, the value of *seg* must be consistent at the point that memory is examined, both before and after execution. While it is obvious that segments in the *dia* function can change the value of *seg*, what can be overlooked from the GWV paper is that events outside machine execution can also affect the value of *seg*. This allows for the modeling of a machine counter which can be detected by the partition of *seg* or DMA which can change the value of memory segments independently of partition execution. This differs from traditional views of non-interference and must be addressed appropriately, as we discuss in a later section.

4 Modification of the GWV Policy

In the previous section we presented a clarification of the use of the GWV model. One crucial point that we covered, is the use of the *dia* and *segs* functions. Assume system *s* has been proven to satisfy GWV policy. As presented in the original paper, it appears that a user could take system *s* and instantiate it with any choice of *dia* or *segs* that they desire. However, reality is much different.

4.1 Limiting Flow Based on Source Segments

Assume *seg1* and *seg2* are in *segs(B)*, and *seg1* is also in *dia(seg)*, but *seg2* is not. The intent of these memberships is that information in *seg1* and *seg2* be readable by code executing in partition *B* and that the information contained in *seg1* is authorized to flow into *seg*. A GWV system is expected to enable the information flow from *seg1* and prevent the flow from *seg2*.

For commercial microprocessors and operating systems, this capability is too powerful. Special purpose hardware, or emulation software, is needed to provide the ability to restrict information flow both from a particular memory segment to a destination segment. For example, consider the cutpoint-to-cutpoint verification approach discussed in the previous section. During a partition's execution window it could copy information from either *seg1* or *seg2* into a register and then write the results into *seg.* The only way to stop the copy from *seg2* is to tag the information based on its original source. This is not a capability of modern commercial microprocessors. Therefore, we specified the following theorem in the encapsulation model of separation:

```
(defthm dia-complete
  (implies
     (member-equal seg (segs part))
     (subsetp-equal (segs part) (dia seg)))
```

Any full system specification will necessarily have to satisfy the theorem along with the separation theorem.

4.2 Limiting Flow Based on Trustworthiness of Code

High assurance systems typically are built from a combination of trustworthy and untrustworthy components. The trustworthy components are responsible for control of the critical aspects of the system, and as such must be rigorously evaluated to perform their operations correctly. The purpose of a separation kernel is to enable the secure and safe deployment of trusted and un-trusted components within a single computing system. The separation kernel maintains data separation, fault-containment and controlled information flow. As stated, the GWV policy does not specifically provide for this capability unless the user of the policy precisely implements the *cut-point* model of verification. All state aspects of a partition must be represented by the segments. If a portion of state is not mapped to a segment then we can have some problems.

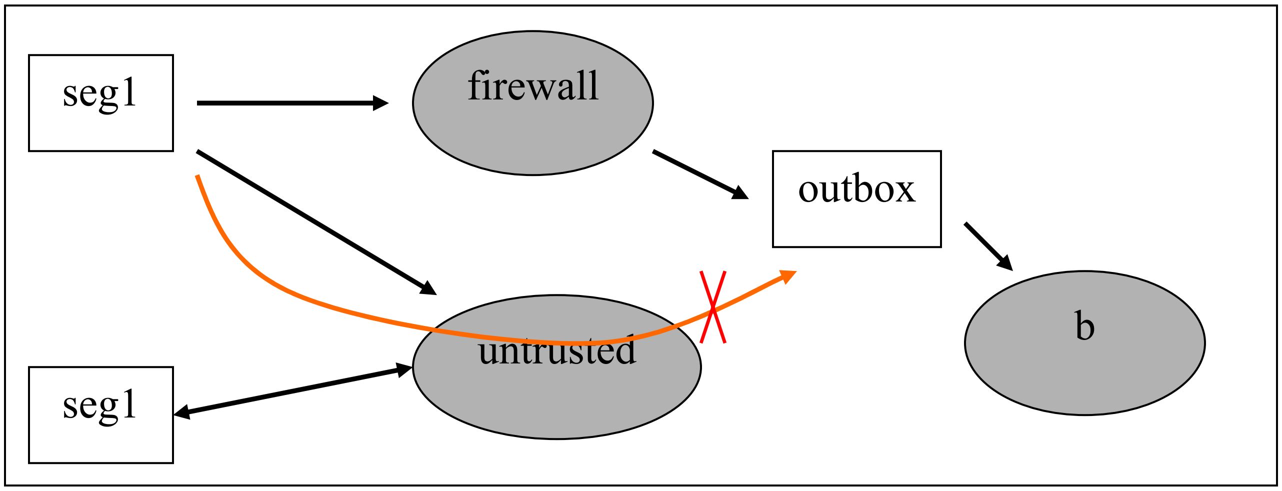

Figure 3. A firewall example with an untrusted audit process

Consider the system depicted in Figure 3, derived from the original GWV paper. In this system, as graphically depicted, we wish to allow the '*firewall* to take information from a memory segment *seg1* and transfer it to the *outbox* memory segment, only after "blackening" it. Assume we wish to have an auditing process, *untrusted*, that monitors information that is sent from *seg1* an creates an audit record of it in *seg2*. If the user of the GWV policy does not specify microprocessor state and code in a memory segment, there is nothing to prevent *untrusted* from writing to the outbox (since information flow is specified only in terms of sources of information, which in this case is *seg1*, and not who is transferring the information). If we follow the GWV example and use the defaxiom approach of specifying the mapping of *segs* and *dia*, we can define the following mappings.

```
dia(outbox)  = {seg1}
dia(seg1)  = { }
dia(seg2)  = {seg1}
segs(firewall)  = { seg1 }
segs(b)  =  { outbox }
segs(untrusted) = { seg1, seg2 }
```

With these mappings and an abstract definition of *next*, we have shown that it is possible for *untrusted* to modify outbox; in other words, it does not have to be '*firewall* executing code to modify outbox.

```
(defthm untrusted-writing
  (implies
   (and
      (not (equal (select outbox (next st1)) (select outbox (next st2)))
      (equal (current st1) (current st2)))
   (equal (current st1) 'firewall)))
```

A modification to the dia function to include partition identification information would correct the problem of unauthorized writing.

Note that the corrections we are suggesting to GWV are important for creating a strong, non-bypassable security policy that can assist in the certification of a separation kernel. We are not suggesting that GWV needs substantial rewriting or has major flaws.

5 GWV and Non-interference

In this section we discuss the application of the GWV policy in the context of a desired non-interference policy. For the purposes of this paper we will use the state-machine model of non-interference as proposed by Bevier and Young [BY94].

In the state-machine model of non-interference there are two specific theorems that must be proven. If these theorems hold true of the system, which are based on the security of single steps in the systems, then when the system starts in a secure state, it stays in a secure state. These theorems are the inductive steps of an inductive proof of the security of the system during a run. For details of the proof, the reader is referred to Bevier and Young's paper [BY94].

For our purposes in this paper we are only concerned about mapping GWV to the inductive steps. Specifically, we wish to map to the steps defined for deterministic systems that support an intransitive security policy[4], which we will call BY in the following. These steps are WSC (weakly step consistent) and LR (locally respects).

5.1 Weakly Step Consistent

A system that is weakly step consistent is one that exhibits non-infiltration. The results of executing a partition will be consistent with its view of the world. In the notation of BY:

$$b \;\alpha\; a \Rightarrow s_1 \overset{a}{\sim} s_2 \wedge s_1 \overset{b}{\sim} s_2 \Rightarrow Step(s_1, b) \overset{a}{\sim} Step(s_2, b)$$

This equation states that if *b* is allowed to communicate with *a,* then if two states s_1 and s_2 are similar with respect to *a*'s view of the world and *b's* view of the world, then *a's* view of the world remains consistent after stepping *b* one step.

How do we map GWV to this policy? First, we need to notice that GWV has a finer granularity of permission than BY. GWV specifies information flow from *b* to specific segments of *a*. We have two choices here:

[4] A transitive policy requires that if information can flow from a to b, and from b to c, then it must be able to flow from a to c. The GWV policy's *dia* function does not require transitivity. The GWV policy allows for specification of a policy that forces information flow between entities to be routed through a specific intermediary – which is at the heart of the MILS approach to security.

- If we want BY precisely specified we require that if *b* can communicate with *a* then all segments in *b* are in the *dia* of each segment in *a.*
- When instantiating the security domains of BY, we specify them in terms of modifiable segments. So instead of specifying *b* can communicate with *a*, we can say *b* can communicate with a_j, where *j* is the index of the memory segment in *a* that *b* can write to.

These solutions are easy to implement in ACL2.

5.2 Locally Respects

A system locally respects the policy if it satisfies non-exfiltration. The results of execution of partition *b* should not modify the state of partition *a* if *b* can not communicate with *a*. In the notation of BY this is:

$$not(b \ \alpha \ \ a) \Rightarrow Step(s,b) \overset{a}{\sim} s$$

Therefore if *b* cannot communicate with *a* then executing *b* does not modify the state in *a's* view of the world. This can not be directly implemented by GWV, since GWV allows spontaneous influences of the state. We need to modify this requirement as follows:

$$not(b \ \alpha \ \ a) \Rightarrow s_1 \overset{a}{\sim} s_2 \Rightarrow Step(s_1,b) \overset{a}{\sim} Step(s_2,b)$$

This states that if *b* can not communicate with *a* then if s_1 and s_2 are the same in *a's* view of the world then *a's* view changes consistently irregardless of *b's* view of the world.

These solutions are easy to implement in ACL2.

6 Conclusion

Formal models can be used to communicate the desired behavior of a system, and the assumptions of that system. Using the formal models represented by GWV we hav\]
+e been able to raise some questions concerning the use of the models, and discuss these with the authors. In any formal system we find that you must be careful about your assumptions, especially when you use a system with axiomatization behavior (such as the ACL2 encapsulation model). Encapsulation allows you to prove theorems about an exemplary model and then export those theorems without restrictions on the specification of the functions that are implicit in the model. We demonstrated this with an example.

Mapping GWV to the BY non-interference policy proved doable through modification of the non-interference equations. These changes were necessary to accommodate the differences between GWV and BY notions of system state change and entity identification. We proved that GWV is at least as strong as general non-interference and supports intransitive non-interference, which is useful for correctness proofs of downgraders and encryption components.

References

[ATO04] Alves-Foss, J., C. Taylor and P. Oman. "A multi-layered approach to security in high assurance system development". *In Proceedings of 37th Annual Hawaii Int. Conf. on System Science (HICSS-37), Jan. 5-8, 2004,* Hawaii, 2004.

[BY94] Bevier, W. and Young, W. "A state-based approach to noninterference", *Journal of Computer Security*, Vol 3. pp. 55-70, 1994.

[GM82] Goguen, J.A. and J. Meseguer. "Security policies and security models". *In Proceedings of IEEE Symposium on Security and Privacy,* pg. 11-20, Oakland, CA 1982.

[GWV03] Greve, D. and M. Wilding and W.M. Vanfleet. "A Separation Kernel Formal Security Policy", *In Proceedings of the ACL2 Workshop* 2003, July 2003.

[KMM00] Kaugman, M., P. Maniolios, J.S. Moore. Computer-aided Reasoning: An Approach. Kluwar Adademic Publ., 2000.

[MWTG00] Martin W., P.White, F.S. Taylor and A. Goldberg. "Formal construction of the mathematically analyzed separation kernel". *In Proc. of 15th IEEE Int. Conf. on Automated Software Eng. (ASE '00),* 2000.

[Nis99] NIST. Common Criteria for Information Security Evaluation. Parts 1, 2, 3. 1999. http://csrc.nist.gov/cc/ccv20/ccv2list.htm, NIST, 1999.

[RTC92] RTCA. DO-178B/ED-12B. "Software Considerations in Airborne Systems and Equipment Certification". RTCA, 1992.

[Rus81] Rushby, J. "Design and verification of secure systems," *In Proc. ACM Symposium on Operating System Principles,* Vol. 15, pp. 12-21, 1981.