Formalizing Security Policies for Dynamic and Distributed Systems

David Greve, Matthew Wilding, Raymond Richards
Rockwell Collins Advanced Technology Center
Cedar Rapids, IA
{dagreve,mmwildin,rjricha1}@rockwellcollins.com

W. Mark Vanfleet
Department of Defense
wvanflee@ncsc.mil

16 September 2004

Abstract

Separation kernels play a crucial role in providing security guarantees for MILS systems. Developing useful formalizations of security guarantees, however, can be challenging. In the paper, "A Separation Kernel Formal Security Policy," Greve, Wilding, and Vanfleet propose a formal security policy for separation kernels and argue for its usefulness by using it in a proof involving the correctness of a firewall application. The proposed policy has subsequently been used in the certification of the intrinsic partitioning mechanism of the AAMP7 microprocessor. In this paper we generalize the original security policy formalization to enable its application to a wider variety of systems, including multiprocessor systems and systems exhibiting dynamic behavior. Finally, we argue that security policies are most naturally expressed using graphical representations of system information flow and we show how such representations relate to our generalized security policy.

1 Introduction

In the paper, "A Separation Kernel Formal Security Policy,"[2] Greve, Wilding, and Vanfleet propose a formal security policy for separation kernels. For notational convenience we will follow the lead of [1] and refer to this policy as the GWV policy. The GWV policy is intended to be applied to separation kernels. A separation kernel is an innovation first published in the early 1980's for achitecting secure systems. Interaction between applications, or partitions as they are commonly called, is mediated by the separation kernel, which enforces a security policy of information flow and data isolation on those interactions.

Because the GWV theorem applies to separation kernels, it is assumed that the system contains some number of partitions that can be uniquely identified by their name. One of the partitions is designated as the "current" partition. The idea is that, during a single step of the system state, only the current partition executes. The function `current` calculates the current partition from a machine state.

```
((current *) => *)
```

The entire system state (possibly including a model of the external environment) is assumed to be composed of some number of uniquely identifiable state elements. The unique identifier used to address a particular state element is called a segment (or seg). The names of the segments associated with a particular partition are available from the function `segs`, which takes as an argument the name of a partition.

```
((segs *) => *)
```

The values in a machine state that are associated with a segment are extracted by the function `select`, which takes two arguments: the name of the segment and the machine state.

```
((select * *) => *)
```

The function `selectlist` is a generalization of `select` that takes a list of segments and a state and returns from the state a list of the values associated with each of the segments in the list.

```
((selectlist * *) => *)
```

The separation kernel enforces a communication policy on the segments. This policy is modeled with the function `dia` (DIA : Direct Interaction Allowed) which, given a segment name, returns the list of segments allowed to influence value of that segment during a single step of the system.

```
((dia *) => *)
```

The function intersection takes two lists and returns a list containing only elements common to both arguments.

```
((intersection * *) => *)
```

The final function appearing in the GWV theorem is `next`, which models a single step of the system state. The function `next` takes as an argument a machine state and returns a machine state that represents the effect of a single system step.

```
((next *) => *)
```

The GWV theorem, then, is expressed as follows:

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

This theorem states that the effect of a single step of the system state on an arbitrary segment of the state, `seg`, is a function of the set of segments that are both allowed to interact with `seg` and are associated with the current partition.

The GWV paper argues that a good specification of a system component has two characteristics. First, a good specification can be formally *proved* about a particular system component. Second, a good specification can be *used* in the formal proofs of more abstract properties of larger systems that build upon the evaluated system component. The GWV policy has demonstrated both properties. The GWV policy has been *proven* to hold for the AAMP7 intrinsic partitioning kernel, and it has been *used* to verify information flow properties of a simple firewall. Given these observations and the above criteria, one might conclude that the GWV policy is, in fact, a good specification for a separation kernel.

Despite being a good specification, however, GWV still has several limitations. While GWV has been shown to capture a notion of separation, and although it was shown to be useful in reasoning about certain systems, the formulation of GWV is still too restrictive to be used to describe a number of other, obviously useful systems. In this paper we identify three specific shortcomings

of the original security policy statement and propose an alternative formulation to help address those issues.

2 GWV Shortcomings

The first problem with GWV is a problem in *using* it to prove larger system properties. This issue was first illustrated by example in [1]. In the example, the original GWV firewall system is extended by adding a simple monitoring partition that is able to read the inbox of the firewall partition. In the resulting system, one can prove that the GWV policy does not prohibit the monitoring program from writing information directly into the outbox. In fact, any partition that can read from any element of the DIA of a segment can then, according to the policy, modify the segment using that value. Note that, while DIA does capture the notion of information flow, it does not capture the intent of the system designer that only certain partitions are allowed to actually move the information. What we conclude is that there are certain desirable system-level properties that cannot be adequately expressed using the existing GWV theorem formulation. What is needed is the ability to model system behavior with sufficient detail to capture the intent of the end user.

A second problem with GWV is in *proving* that a particular system implements the policy. Many interesting systems are dynamic in nature. Dynamic systems impact security in many different ways, but one important manifestation is in the creation of covert channels. The AAMP7 has a static schedule and a kernel space which can be shown to be free of interaction from user partitions. Unfortunately, this is not necessarily true of all systems proposed for use in secure applications. Rather, many useful systems have covert channels that do, in fact, allow small amounts of information to leak from one partition to another. The standard approach for dealing with covert channels is to identify and analyze them to provide assurance that the information flow is too small or too difficult to exploit to be of any practical concern. This works well from a pragmatic standpoint. Unfortunately, from GWV's rigorous, mathematical standpoint, any deviation from the ideal system renders the GWV theorem effectively useless[1]. What is needed is a mechanism for employing informal evaluations in the final theorem statement without sacrificing the rigorous mathematics that provide high degrees of assurance.

The final concern with the GWV theorem is that it is simply too specific. There is already a move afoot to define MILS at the network level. The GWV theorem was developed for, and really only applies to, single processor systems. It would be very difficult develop an accurate, usable model of a distributed, asynchronous system that would satisfy the GWV theorem. What is needed is

[1] The theorem may be either unprovable or it may not be inductively strong, depending on how the theorem is instantiated and how the covert storage channel is manifested. An inductively strong GWV theorem guarantees that once the system has attained a secure state, it will remain in that state. Theorems that are not inductively strong cannot guarantee that the system will remain in a secure state and, as a result, are effectively unusable.

a more general characterization of separation that can be applied effectively at different levels of system hierarchy.

In the remainder of this paper we present generalizations of GWV that provide increased specificity, enable more flexible application, and allow for the characterization of a larger class of systems.

3 Agents

To increase the expressiveness of the GWV theorem we introduce the concept of agents. The GWV theorem uses DIA to characterize the overall information flow behavior of the system. In the monitor partition example, however, simple information flow fails to accurately reflect the intended security property. GWV fails in this case because it includes no notion of *who* is allowed to move the data. We propose simply refining the notion of DIA to include a notion of a responsible entity. This can be accomplished through the use of agents. An agent can be thought of as a logical or computational identity (usually a partition) that may carry with it a certain level of trust.

In perhaps the simplest model of agents, they are interpreted as partitions. This model is convenient because we wish to associate certain properties with particular partitions: a partition is trusted, a partition is a firewall, etc. A more general notion of agents, however, interprets them as arbitrary predicates over state. An interpretation of agents would then map agent names in the graph to particular predicate functions over the state[2]. Agents, for example, need not be exclusive and multiple agents may be active at any time.

In our new formulation of GWV we replace the notion of current with the notion of agents. This change allows us to characterize systems with more than one active agent (partition) at any given time. The `dia` function is also replaced with a function, `agent-dia`, that selects only those dependencies that are allowed based on the currently active set of agents. The transition from dia to agent-dia also allows us to remove the intersection computation, which is now done implicitly in agent-dia. Removing the intersection computation also allows us to simplify our hypotheses by removing the statement equating `current` in the two worlds and the one equating the value of `seg` in the two worlds. We can do this because we assume that dependencies on self and dependencies on `current` are included in the agent-dia function and they are not at risk of being lost in the intersection computation[3]. The new theorem appears as follows:

```
(defthm agent-separation
```

[2] An agent could also be a pair: a predicate and a state transition function. The semantics of such an agent would be that, if the predicate is satisfied, the associated function would, at least from the perspective of the node, characterize the behavior of a single step of the system.

[3] In the original GWV formulation, because dependencies on self and `current` were included in the hypothesis by default, they could actually be omitted from the `dia` function. In observing only the dia function, one might erroneously conclude that the value of a particular segment does not depend on the value of `current` or self, when, in fact, these dependencies are implicit in the formulation of the theorem. In the new formulation, all dependencies are explicit and are included in the computation of `dia-agent`.

```
(implies
  (let ((dia-seg (agent-dia (agents st1) seg)))
     (equal (select-list dia-seg st1)
            (select-list dia-seg st2))
 (equal (select seg (next st1))
        (select seg (next st2))))
```

While the hypothesis of this theorem appears asymmetrical, employing (agents st1) while not mentioning (agents st2), proving that a particular system satisfies this theorem will require that the select-list equality hypothesis imply that (agents st1) is equal to (agents st2)[4]. Note that for a a specific segment, `seg`, if one were to compute, over all possible agents, the set of segments returned by `agent-dia`, the resulting set would be equal to the set of segments returned by the original GWV `dia` function. The introduction of agents in this new formulation makes it strictly more expressive than GWV, thus enabling us to describe properties that otherwise could not even be expressed in GWV. Using this new formalization, it becomes possible to accurately formalize the intended behavior of the monitor system and prove the desired property.

4 DIA and Information Flow Graphs

The DIA (direct interaction allowed) function is really the heart of the GWV theorem. The DIA of a particular segment of the system state is the set of all segments in the state that may contribute to the computation of the value stored in seg in the course of a single system step. In the broadest sense, the DIA relationship is the security policy enforced by the system. The DIA function in the original GWV theorem characterizes the overall information flow behavior of the system. Presumably it is the DIA relationship that system designers would evaluate in a larger context to establish the suitability of the system for a particular purpose. To the extent that DIA can be programmed into the system it can be used to reflect a larger security policy. To the extent that DIA is determined by the system itself, the resulting communication policy must be evaluated in the context of the larger system to establish viability.

Implicit in the DIA function is the notion of an information flow graph: a graph describing how information flows in the system. In the information flow graph implicitly defined by the function `agent-dia`, segments are stored as nodes and an information flow from segment A to segment B is modeled as an edge that leads from node A to node B that is labeled with the agent responsible for the information flow. With this model in mind, `agent-dia` can be viewed as a function that returns from the information flow graph a set of all of the nodes with edges that lead to a particular segment and are labeled by one of the specified set of agents.

[4]The actual relationship between (agents st1) and (agents st2) is more subtle and is probably something more like (agent-dia (agents st1) seg) is equal to (agent-dia (agents st2) seg). In some degenerate circumstances even this is stronger than it needs to be.

This graphical view of information flow is important because the circle and arrow diagrams typically used to describe communication policies at more abstract levels are information flow graphs as well. Our proposal is to extract the information flow graph, implicit in the DIA function, make it explicit, and export it as the top level formalization of the system security policy. The resulting graph can then be evaluated, compared, or instantiated with other information flow graphs at the appropriate level of abstraction.

This observation about the nature of the information flow graph implicit in the GWV theorem hints at why the GWV theorem is too restrictive to apply to many systems. It is too restrictive precisely because GWV is a *property* of the information flow graph, rather than a *specification* of that graph. No single property of an information flow graph can hope to be satisfied by all potentially interesting applications. Rather, the formalization of a security policy should focus on what the security policy *is* rather than on a particular aspect of what the policy might *do*, as does GWV.

4.1 The Graphical Abstraction

The common criteria requires the development of some number of models of system behavior. At the top we have a formalization of the security policy that we hope to implement. At the lowest level there is a model of system behavior that can be mapped to the implementation without the need for further design decisions. In between there are some number of model abstractions, each of which commutes with the next higher level via some abstraction.

The graphical representation of information flow can be used as one of those models because an accurate graphical representation of system information flow is a valid abstraction of system behavior. We say that a particular graph is an abstraction of the system if it satisfies a commuting relationship. It is the commuting theorem that connects the sate-based representation to the graphical representation of information flow. The graphical commuting theorem has the following form:

```
(defthm graphical-abstraction
 (equal (next st)
        (apply-graph graph st)))
```

This theorem says that the function `next` adheres to (or, is characterized by) the information flow policy specified by graph as interpreted by apply-graph. Note that, with the appropriate definition of apply-graph, we can prove the original separation theorem using this formulation (It is, after all, just a property of the graph). Note too that the graphical model of system behavior is much simpler and much more abstract than the functional model. Whereas the functional model explicitly describes what the system does, this graphical model merely describes how it does it. The low-level model may require many sophisticated data structures and functions, composed in a variety of ways to describe the functionality of the system. The graphical model, on the other

hand, requires only a single representation of the system and a single operation, graph composition, in order to describe system behavior.

4.1.1 A note on graph utility

Keep in mind that the simplest graph to verify is the completely connected graph that says that everything depends upon everything else. This, however, is not a very useful graph when it comes to using it to implement a specific security policy. Graphs become more difficult to verify as they become more restrictive. More restrictive graphs, however, are generally more useful. Note, however, that just because we can construct a graph with a particular property, it doesn't mean that the system will satisfy that graph. If the graph does not accurately reflect the behavior of the underlying system, it will be impossible to prove that our system implements that graph.

4.2 A Graphical Security Policy

In GWV, the final security property was expressed in terms of the system state (two parallel states, to be exact). We would call this a state-based representation of our security policy. More generally, however, the system security policy is characterized by its information flow graph. Such a characterization of the system security policy is certainly less concise than GWV, but it is far more expressive. Given this graphical representation of information flow in the system, we can now interpret statements about information flow in the system as statements about this graph. GWV is but a single instance of an unbounded number of possible properties satisfied by the information flow graph of the system. In addition, many interesting security properties can be expressed quite naturally as graph-theoretic properties, but only with difficulty (or not at all) in a state-based representation.

4.2.1 Firewall Analysis

In a firewall system we might wish to say that the only way information can move from the red location (R) to the black location (B) is thru the firewall (F). This statement can be interpreted as the graph theoretic statement: any path leading from the black node to the red node must first pass through a firewall edge. Note how natural it is to state this theorem, especially in contrast with the original GWV firewall formulation that relied on a state-based formulation.

An equivalent formulation of the same statement is to say that if, from the graph we remove all edges labeled with F (the firewall agent), then, in the transitive closure of the modified graph, the black node does not depend upon the red node.

4.2.2 Covert Storage Channel Analysis

In order to make useful statements about systems containing covert storage channels, we employ reasoning techniques analogous to the ones used in mod-

eling a firewall. In the case of the firewall, we said that the only information flow between red and black was via an edge labeled by firewall. In the case of covert storage channels, we first identify all of the segments involved in the covert channel. We then say that, modulo the covert storage channels, the system security policy is acceptable. We do this by demonstrating that every path that does not pass through a covert storage channel node is consistent with the desired security policy.

There is the possibility that the storage channel is used by two sources: one being the covert channel and the other being a serious exploit. Simply ignoring the storage channel in this case would give free reign to the serious exploit. For exactly this reason, thorough analysis of covert channels is essential.

5 Conclusion

We identified three weaknesses in the GWV separation theorem: it is not sufficiently expressive, it is impossible to satisfy with a large class of useful systems, and it is targeted too specifically to single processor systems. We then demonstrated how to generalize the original security policy formalization to enable its application to a wider variety of systems, including multiprocessor systems and systems exhibiting dynamic behavior. Finally, we argued that security policies are most naturally expressed using graphical representations of system information flow and then showed that such representations naturally satisfy our generalized security policy.

References

[1] Jim Alves-Foss and Carol Taylor. An Analysis of the GWV Security Policy. In *ACL2 Workshop 2004*, November 2004.

[2] David Greve, Matthew Wilding, and W. Mark Vanfleet. A Separation Kernel Formal Security Policy. In *ACL2 Workshop 2004*, June 2003.