Modeling Information Routing with Noninterference

Ruud Koolen and Julien Schmaltz
Eindhoven University of Technology
{r.p.j.koolen, j.schmaltz}@tue.nl

ABSTRACT

To achieve the highest levels of assurance, systems based on the MILS architecture need to be formally analysed. In this, a key challenge is reasoning about the inter-domain flow of information on a finer scale than the domain level. In this paper, we extend Rushby's model of noninterference with explicit between-domain information transfer, as well as programs that determine domain behavior. These extensions enable the reasoning at an abstract level built on top of noninterference, at a much finer level than allowed by base noninterference. As an illustration of our approach, we formally model and analyze an example system inspired by the GWV Firewall.

1. INTRODUCTION

Formal verification of software properties is a powerful tool for achieving the highest levels of confidence that a piece of software behaves as desired, as codified in verificational standards such as EAL6 and EAL7 of the Common Criteria. When applied to a system built as an instance of the MILS architectural paradigm, this verificational problem consists of three subproblems: in order to verify the system, one needs to verify the behavior of the separation kernel, the individual applications, and the allowed communication scheme between them.

Many requirements have been proposed that formalize the behavior of a separation kernel; examples include Rushby's Intransitive Noninterference [5], Greve, Wilding and Vanfleet's Separation [2], and many variations of these schemes. Formalizing the behavior of individual applications, on the other hand, is a problem for which few satisfactory approaches exist; as we have argued in a previous paper [3], existing solutions —such as illustrated by the GWV Firewall [2]— are too unrealistic to be usable for practical formalizations. A similar predicate holds for formalizations of the information flow between different applications; whereas separation kernel formalizations such as noninterference include facilities for describing the forms of communication allowed between different applications, these mechanisms are far too coarse-grained for many practical verificational challenges.

In this paper, we propose an extension of Rushby's noninterference model that makes it possible to formally reason about the communication behavior of applications running on top of a separation kernel proven to respect a given information flow policy. The main contribution in this model is to extend the abstract notion of noninterference such that the flow of information can be described on a more detailed level than allowed by base noninterference. More precisely, the contributions of our paper are the following:

1. In Section 3, we extend the Rushby system model with an explicit notion of *information*, and the way it can flow between domains.

2. To enable the reasoning about specific programs that run inside domains supported by the separation kernel, we introduce the notion of *domain programs* that determine domain behavior in Seection 4.

3. Finally, in Section 5, we illustrate the applicability of the extensions above by performing the formal verification of an example system, re-visiting the firewall example originally introduced by Greve, Wilding, and Vanfleet.

Note that all models and proofs are all formalized within the logic of the Isabelle/HOL theorem proving system [4].

2. BACKGROUND AND RELATED WORK

A *separation kernel* is a simple type of operating system that provides an environment in which multiple components, known as *domains* or *partitions*, can run independently on a shared piece of hardware without interference from each other. In that sense, it is not much different from a general-purpose operating system, which might provide the same functionality regarding *processes*. Unlike general-purpose operating systems, a separation kernel provides the further guarantee that different partitions cannot affect each other in any way whatsoever, except through a set of well-defined communication channels through which influence may flow. In the absence of such a communication channel between two partitions, one partition should not be able to distinguish whether the other partition is present at all. This is a much stronger guarantee than what is implemented by general-purpose operating systems, in which mechanisms like contention of resources such as processing power and memory,

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

or a shared data storage system such as a filesystem, provide ways in which influence can flow unchecked.

To support formal reasoning about systems based on separation kernels, Rushby [5] introduced the notion of *intransitive noninterference* as a formal model of the services and guarantees provided by separation kernels; we present here the formulation by Van der Meyden [7]. This automata-theoretic formalization models a system as a set of *security domains* —independent components separated to some degree by the separation kernel— that can each access a certain part of the system resources, such as a part of the memory of the system allocated to that domain. Each of these domains can perform a set of *system calls*, requests towards the separation kernel to perform a certain restricted operation that will hopefully change the state of the global system in some way. Together, the domain-accessible resources and executable system calls define a transition system, with system calls identifying labelled actions, and domain resources representing a (structured) state labelling function.

More formally, a *Rushby system* is a deterministic labelled transition system $(S, s_0, D, A, \mathsf{step}, O, \mathsf{obs})$, where

- S is a set of states;
- $s_0 \in S$ is the initial state;
- D is a set of domains;
- A is a set of *actions*, each representing a particular system call performed by a particular domain;
- $\mathsf{step} : S \times A \to S$ is a transition function;
- O is a set of possible domain *observations*, and
- $\mathsf{obs} : S \times D \to O$ is a function describing the contents of the system resources accessible to a particular domain in a given state.[1]

In this definition, an action $a \in A$ represents a system call that can be performed by a particular domain $d \in D$. In particular, each action can be performed only by a single domain, denoted $\mathsf{dom}(a)$; as a notational convenience, the set of actions a for which $\mathsf{dom}(a) = d$ is denoted as A_d. The transition function step describes the way the system state changes as a consequence of the execution of system calls performed by domains; in particular, $\mathsf{step}(s, a)$ is the resulting state after executing the action a in state s. As such, the transition function can also be interpreted as a deterministic transition relation $\to$, with $s \xrightarrow{a} t$ if and only if $t = \mathsf{step}(s, a)$. As abuse of notation, we will also write $\mathsf{step}(s, \alpha)$ for sequences of states α, where $\mathsf{step}(s, []) = s$ and $\mathsf{step}(s, a \cdot \alpha) = \mathsf{step}(\mathsf{step}(s, a), \alpha)$.

The observation function obs abstractly describes the degree to which a given domain can tell states of the whole system apart. For a state $s \in S$ and domain $d \in D$, the observation $\mathsf{obs}(s, d)$ describes the state in s of all resources that d has access to; consequently, if $\mathsf{obs}(s, d) = \mathsf{obs}(t, d)$ for states s and t, the states s and t are indistinguishable to d. This will become critical when formalizing strong separation properties.

[1]The obs function is the main difference between Van der Meyden's formulation of intransitive noninterference and Rushby's original. Instead of obs, Rushby uses a function $\mathsf{output} : S \times A \to O$, defining an action-observed automaton instead of a state-observed one.

$$
\begin{aligned}
&\mathsf{ipurge}'_{\rightsquigarrow}([], E) = [] \\
&\mathsf{ipurge}'_{\rightsquigarrow}(\alpha \cdot a, E) = \\
&\begin{cases}
\mathsf{ipurge}'_{\rightsquigarrow}(\alpha, (E \cup \{\mathsf{dom}(a)\})) \cdot a \\
\qquad \text{if } \mathsf{dom}(a) \rightsquigarrow d \text{ for some } d \in E \\
\mathsf{ipurge}'_{\rightsquigarrow}(\alpha, E) \\
\qquad \text{otherwise}
\end{cases} \\
&\mathsf{ipurge}_{\rightsquigarrow}(\alpha, d) = \mathsf{ipurge}'_{\rightsquigarrow}(\alpha, \{d\})
\end{aligned}
$$

Figure 1: Van der Meyden's formulation of the $\mathsf{ipurge}_{\rightsquigarrow}$ function. $\mathsf{ipurge}'_{\rightsquigarrow}(\alpha, E)$ is the subsequence of actions of α that are allowed to influence at least one domain in E.

A Rushby system as defined above models the behavior of an operating system of sorts in which a domain structure can be recognized; it does not, as such, describe any guarantees regarding the level of domain isolation realized by the operating system kernel.

An *information flow policy* is a description of the degree and forms in which information is allowed to flow between different security domains; in other words, it is a specification of the degree to which domains running on a separation kernel need *not* be perfectly isolated from each other. In the noninterference model, such a policy takes the form of a reflexive binary relation $\rightsquigarrow$ between domains, in which a policy $\rightsquigarrow$ with $d \rightsquigarrow e$ approximately encodes the property that domain e is allowed to learn information available to domain d whenever domain d performs an action. In other words, it roughly specifies that domain d is allowed to influence domain e through its actions.

To substantiate this informal property, Rushby defines a function $\mathsf{ipurge}_{\rightsquigarrow} : A^* \times D \to A^*$, described in Figure 1, that for a sequence of actions α and a domain d defines the subsequence α' of actions whose effects may be noticed by d after the execution of α, according to the information flow policy $\rightsquigarrow$. Based on this function, Rushby defines the *noninterference* property as the requirement that for all action sequences α and β and for all domains d, if $\mathsf{ipurge}_{\rightsquigarrow}(\alpha, d) = \mathsf{ipurge}_{\rightsquigarrow}(\beta, d)$, then $\mathsf{obs}(\mathsf{step}(s_0, \alpha), d) = \mathsf{obs}(\mathsf{step}(s_0, \beta), d)$. In other words, if α and β are action sequences that should have the same observable consequences for domain d according to the information flow policy, the resulting states after executing these action sequences in the initial state s_0 should be indistinguishable to d.

Noninterference for a given information flow policy is a property that may or may not be satisfied by a complete system, consisting of both an operating system of some sort and domains running on that operating system. Noninterference can be considered a property of separation kernels if the separation kernel can guarantee that the noninterference property always holds. That is, an operating system can be considered a separation kernel if, for a given information flow policy $\rightsquigarrow$, it can guarantee that the system as a whole satisfies the noninterference property no matter what applications run inside particular security domains, and no matter what these applications try to do.

Information Flow Content

The noninterference property, as described above, specifies in formal detail the guarantees that a given application can rely on when running as an isolated component on top of a separation kernel. This is a great help when any attempt is made to prove properties about the behavior of an application in the context of a separation kernel; for in this analysis, the behaviors of any applications that are unable to affect the studied application can be disregarded entirely. Domains that *can* affect the studied application remain a complication, but this still greatly reduces the number of cases to consider.

Noninterference does not provide any tools, however, for reasoning about the detailed behavior of domains that are allowed to influence the domain under consideration. As an obvious example, an application domain can very rarely function as desired if a different domain that is allowed to influence it chooses to exercise that allowance by completely wrecking the destination domain's working memory; yet the noninterference property does not contain any clause to disallow this. Meaningful cooperation between such domains is only practically possible if any information exchange between the cooperating domains happens in the form of some well-defined communication protocol; a typical example would be an inter-domain message passing system as implemented by many operating systems in countless variations. As a consequence, any domain-level formal analyses of systems in which inter-domain information exchange takes place must necessarily specify a communication protocol of some sort, and adherence thereto of the domains involved. As noninterference by itself does not provide anything like this, such a system must be specified on top of the noninterference abstractions.

If such a system is taken for granted, it becomes possible to reason formally about the information content transmitted between domains that influence each other. If a domain d is allowed to influence domain e, domain e is not usually supposed to have complete access to all information accessible to d; indeed, the desire to transfer such information from d to e in a *limited* way is one of the main reasons for having d and e as two separate domains in the first place.

In practical systems, one commonly wants to achieve a situation in which a domain d transmits *some* of the information it holds to a receiver domain e, while keeping other, private information away from any domains other than itself. Consequently, when doing formal analysis of systems, this is the sort of property one would like to formally establish.

It is clear that a model describing the flow of information content along domains would be a useful formal basis for doing this sort of analysis. Using such a system, one could formulate properties about the form of information transfer that is being undertaken by particular domains. Combining such properties with the guarantees made by the separation kernel as formalized by the noninterference property, one could then prove that the system as a whole, consisting of both the separation kernel and the various domains, never exhibits undesirable information transfer.

In this paper, we aim to develop a simple model to describe coordinated domain information exchanges of this sort, in a way that connects well with the guarantees delivered by the noninterference property. We also illustrate a method for describing the ways in which domains make use of this system, thus specifying the behavior of domains in regards to information transfer.

To confirm that this model, while abstract, is also sufficiently detailed to express realistic system properties, we use this model to describe the properties of a simple but realistic system containing domains whose behavior depends sensitively on the content of information transferred. This system consists primarily of a "firewall" domain that forwards incoming information exchanges to destination domains while satisfying certain security requirements. The example system is reminiscent of GWV's Firewall [2] example; indeed, this case study can be readily interpreted as the lifting of the GWV Firewall example to Rushby's noninterference formalization.

Related Work

As noted above, the example use case constructed as an applicability test of the communication model is in many ways a variation of the Firewall use case studied by Greve, Wilding, and Vanfleet [2] as a similar test of their own separation kernel formalization, and further studied by Rushby [6] and Van der Meyden [1]. This use case is based on GWV's own formalization of separation kernel behavior, which they call Separation; this formalization is quite different from Rushby's noninterference, and as a consequence the details of the specification of the firewall behavior bear little resemblance to the version we use.

In [3], the authors of this paper argue that the communication model used in the construction of the GWV Firewall is too unrealistic to be useful in describing the behavior of practical systems; while the correctness results in [2] are correct, this is accomplished only by making assumptions on the communication structure that no realistic system can satisfy. As such, this paper is a counterproposal of sorts, aiming to develop a model with a better applicability to practical systems.

3. INFORMATION TRANSFER

In this paper, we do not attempt to specify the internal workings of a system to communicate information between domains in a coordinated way. Instead, we axiomatize the way such a system is to behave, and model its effects in a form that falls within the purview of the noninterference guarantee.

The flows of information are a famously subtle topic; an accurate coverage of the behavior of information, and the ways in which activities can influence it, involve such considerations as results in probability theory, information theory, cryptography, and several other fields. An analysis that takes all these matters into account would make an inordinately powerful vehicle for analyzing systems in which information flows between components. Unfortunately, formulating properties for systems or components to satisfy in such a framework —much less actually proving them— is still too far removed from the state of the art in formal verification to make this a feasible approach. For this reason, we propose a model describing the behavior of flowing information that aims to approximate reality well enough to enable formal verification of practical information flow properties, without claiming to accurately take into account the full subtleties of the problem.

Two key observations inspire the model of information and the communication thereof that we use in this paper. The first observation is the information-theoretically elementary point that in full generality, information about a system can only be acquired by interacting with that system; as long as two systems do not interact, no information flow can occur between them in any direction. The consequence of this in the context of a Rushby system is that no information transfer can happen between two domains without at least one of these domains taking an action with communicative consequences. In particular, it is not possible for a domain to divine information about, or present in, another domain without performing such an action. While a domain can of course modify its memory in a way that resembles the state of having information about some other domain, this cannot have an expected *accurate* bearing on the state of the other domain, and thus cannot create any information about that domain in the information-theoretic sense.

The other observation is that in the noninterference model, information can only flow from a domain d to domain e through actions of d; in particular, it is not possible for domain e to collect information from d by its own accord. That is to say, if domain e has access to a piece of information about domain d after successive execution of actions a_d with $\mathsf{dom}(a_d) = d$ and a_e with $\mathsf{dom}(a_e) = e$, the information transfer must be modelled to have happened atomically during execution of a_d; any other model would contradict the noninterference property.

What is more, the information flow policy behind a noninterference property is not necessarily symmetric; that is, it is possible for domains d, e to have $d \rightsquigarrow e$ but $e \not\rightsquigarrow d$. As a consequence, communication must generally take the form of one-way message passing, in which a sender domain transmits information to a receiver domain without paying much heed to the fate of this message.

Combining these observation readily yields a theory of information in which information is a resource held by particular domains, which can then transmit this information to other domains —in accordance with the information flow policy— by performing actions. Domains can only acquire information by receiving it from other domains in this way, with one exception: each domain is presumed to at all times have perfect information about its own state. Importantly, this is the (only) way in which information can ever enter the formal system; there is no other way in which information can be synthesized from nothing.

The model sketched here can be formalized as an extension to the Rushby formalization of operating systems. A *Rushby system with information* is a Rushby system as defined in Section 2, together with a set I of possible *units of information*. In each state s, each domain d has access to a certain set of pieces of information; and this set is part of the observation $\mathsf{obs}(s, d)$ that the domain can make of the state. That is to say, for a Rushby system with information with observation domain O, the set O has the form $O = 2^I \times O'$; for convenience of notation, we will just write $i \in \mathsf{obs}(s, d)$ to denote that domain d has access to information unit i in state s.

Each unit of information $i \in I$ is presumed to describe the state of a particular domain d, which is called the *subject* of the unit of information; this subject is denoted as $\mathsf{subject}(i)$, for $\mathsf{subject} : I \to D$. Based on this notion, we can describe the message passing semantics that characterize the flow of information between domains. These message passing semantics amount to the property that information may be transmitted through the execution of actions, from the domain executing the action, assuming that domain has access to the transmitted information, to arbitrary receiver domains. Another property implied by the message passing semantics is that a domain can never remove information from being accessible to another domain; once a domain has received a unit of information, it can only be deleted by a voluntary action performed by that domain.

Interpreted in the context of the set I and observation function obs, these semantics translate to two formal axioms that a Rushby system with information must satisfy:

- if $i \in \mathsf{obs}(s, d)$ and a is an action such that $\mathsf{dom}(a) \neq d$, then $i \in \mathsf{obs}(\mathsf{step}(s, a), d)$ (information may not be removed by anyone other than the domain holding it); and

- if $i \in \mathsf{obs}(\mathsf{step}(s, a), d)$ and $i \notin \mathsf{obs}(s, d)$, then either $i \in \mathsf{obs}(s, \mathsf{dom}(a))$, or $\mathsf{subject}(i) = \mathsf{dom}(a)$ (information transmitted by a domain is accessible to that domain, either inherently due to having that domain as its subject, or due to having received this information in the past).

Together, these axioms approximately characterize the way information behaves when transported through a message-passing-style communication system.

A Rushby system with information —that is to say, a Rushby system with an information domain I and an observation function satisfying these axioms— models an operating system that features a well-defined inter-domain communication system. It does not, by itself, have any bearing on the separation guarantees offered by the operating system. In particular, it is notable that the message passing axioms do not in any way mention the information flow policy; domains can transmit information to arbitrary receivers, with no concern for the opinion of any information flow policy in regards to this information transfer.

From a formal point of view, the consequence of this —by design— is that the message-passing properties and the noninterference property are independent properties that a Rushby system might satisfy. That is to say, the message-passing properties over some information domain I, and the noninterference property for some information flow policy $\rightsquigarrow$, can be interpreted as orthogonal extensions to the base Rushby system.

However, we hold that the message passing properties are defined in such a way that the noninterference property, if applicable, has the expected semantics when applied to the concept of information as defined by the message passing theory. Because both the noninterference property and the message passing properties are defined in terms of the observation function, any restrictions offered by the noninterference property regarding allowed sources of influence to this observation function apply to message-passing information transfer as well. In particular, if d and e are domains such that $d \not\rightsquigarrow e$, any information transmission from d to e is disallowed by the noninterference property; for if d were to transmit a unit of information i to e through execution of the action a, this action would modify the observation of domain e, which is disallowed by the noninterference property. We hold that this is exactly the desired behavior of information in the context of noninterference.

It must be pointed out that the model of information presented here, while inspired by information-theoretical concerns, does not come close to capturing the full semantics of information that that theory dictates. In particular, this model cannot express a situation in which a domain receives information from different sources, combines it in a nontrivial way, and transmits the combination (but not the source material). If, for example, a domain is to receive sensitive information from some source domain, encrypt it using some secret key, and transmit the resulting non-sensitive ciphertext to some receiver domain, the theory of information proposed here cannot model this exchange in a sensible way.

Nonetheless, we hold that this theory of information is an approximation of the real semantics that is sufficient to accurately model many formal verification properties that involve information flow across domains. In Section 5, we support this proposition through an example verification of an information flow property in a noninterference system using this model of information.

4. DOMAIN PROGRAMS

In Rushby's formalization of systems based on a separation kernel, the noninterference property describes a guarantee that the separation kernel makes, no matter what the individual domains attempt to do; that is, it models domains as black boxes with no specified behavior at all. When formalizing of larger systems running on top of a separation kernel, this is only one piece out of many in the complete formalization effort. In such a system, the individual domains tend to run components that also have a well-defined specification of their own, and the desired behavior of the system as a whole can only come to pass if the individual domains meet their own specifications. It follows, then, that in order to prove properties of the system as a whole, one first needs to prove properties regarding the behavior of the individual domains.

At first sight, the noninterference formalization does not seem like it can easily express any such properties. The Rushby model of a system running an operating system kernel (which may or may not be a separation kernel) defines a transition system in which domains can perform any action from a certain action set in any state of the system; while the response of the kernel to this action is deterministic, the domain action taken is not. Domains tend to run programs of some sort, that in particular system states choose particular actions to execute, and as such are very different from nondeterministic action-taking processes; but this distinction does not easily fit into the Rushby model.

This semantic divide is not an actual conflict, however. The transition system defined in Section 2 describes the way the separation kernel responds to system calls from domains in particular states *if the domain were to issue these system calls*. As such, it does not model a complete system with separation kernel and domain implementations; rather, it models a platform created by the separation kernel on which domain applications can run. To model a complete system, one needs to provide a Rushby transition system combined with a description of the actions particular domains choose to take, which models the application programs running inside the domains the separation kernel defines.

In order to model these application behaviors, we propose the notion of a *domain program*, which for a given domain describes the actions a domain chooses to take in each state of the system as a whole. Formally, for a domain d, a *program for* d is a function $P : S \to A_d$, that for any given state $s \in S$ defines the action $P(s)$ with $\mathsf{dom}(P(s)) = d$ that the domain d chooses to execute in state s.[2]

Given a program for a domain d, or indeed programs for all domains $e \in E$ for some $E \subseteq D$, we can study the property that the system as a whole satisfies some requirement *as long as the behavior of the domains* E *is described by the programs* P. This proposition can be fleshed out as the requirement that for all action sequences α, if each action a in α for which $\mathsf{dom}(a) \in E$ is the action specified by the relevant program P, the system-wide requirement holds after executing the action sequence α.

We can formally define this scheme as follows. We say an action sequence $\alpha = a \cdot \alpha'$ *respects* a program P for domain d in state s, if and only if

- either $\mathsf{dom}(a) \neq d$, or $a = P(s)$; and
- α' respects P in state $\mathsf{step}(s, a)$.

We can then say that a property Q holds for a system running programs $P_1 \dots P_n$, if and only if Q holds in any state reached by executing, in the initial state s_0, an action sequence α that respects all of $P_1 \dots P_n$.

By combining this framework with the noninterference property, a system characterization can be given that describes the behavior of both the separation kernel, and the applications running in any of the domains we wish to specify. Using this characterization, one can prove that under the assumption that noninterference holds, for any action sequence that respects the specified domain programs, the resulting state has the desired property. We argue that this is a meaningful and natural way to prove system properties based on formal specifications of individual components. In the next section, we demonstrate this technique by applying it to the task of proving the correct behavior of an example firewall system, based on properties assumed to hold for the firewall program.

5. THE FIREWALL SYSTEM

In order to test the suitability of the theories presented in the previous two sections for the job of compositional verification of information-handling systems, we performed a formal verification of a simple example system. This example system is based on GWV's Firewall example system [2], and like the original it is centered around a domain that prevents certain sensitive information from reaching an untrusted receiver by scrutinizing the information it passes on.

The system whose global properties we seek to verify consists of four components that are relevant to the security requirements. The system is based on an operating system, which we presume to satisfy the guarantees of a separation kernel for some as of yet to-be-determined information flow policy. Inside this separation kernel live a trusted domain t, with access to sensitive information i with $\mathsf{subject}(i) = t$; an untrusted domain u; and a firewall domain f with the responsibility of mediating any information transfers from t

[2]This makes a domain program a deterministic quantity: a specific action is deterministically chosen based on the system state. This model can easily be extended to a nondeterministic variant with no real consequences; in this paper, we use the deterministic version for simplicity.

to u. We assume that t, u and f are all distinct domains[3]. For this system, we want to ensure —and formally verify— that u never gets access to i.

The system is designed around two design properties that together ensure this security requirement. One property is that the operation system is a separation kernel, configured for an information flow policy such that no information flow from t to u is possible which does not pass f. The other property is that the firewall domain contains a program that takes care never to forward the information i to such a domain that the information might reach u — a property which we shall need to specify in further detail.

To verify the desired security requirement, we specify some requirements that the system design informally described above should satisfy. Based on that, we can then proceed to prove that whenever a system meets all those requirements, the desired security requirement should follow.

For a given information flow policy, a *communication path* from a domain d to domain e is a sequence of domains $[d_1, d_2, \ldots, d_n]$, such that $d_1 = d$, $d_n = e$, and $d_i \rightsquigarrow d_{i+1}$ for all $i < n$. A communication path contains a domain c if $d_i = c$ for some i. Using this definition, we can formalize the requirement that no information flow from t to u is possible that does not pass f — this property can be codified as the proposition that no communication path from t to u exists that does not contain f.

To describe the behavior of the firewall domain, a predicate can be described regarding the program P that determines its behavior. Naively, one might try to specify the firewall behavior as the requirement that the firewall program P never chooses to transmit information i to domain u; but this is unsufficient, as the firewall might instead transmit the information i to a domain u', with $u' \rightsquigarrow u$.

Instead, a stronger version of this property is required. Let $E \subseteq D$ be the set of all domains e such that a communication path from e to u exists that does not contain f. Then the behavior of the firewall domain can be fully specified as the following requirement: as above, let P be the program running in domain f. Then we require that for all states s and for all domains $e \in E$, if $i \notin \mathsf{obs}(s, e)$, then $i \notin \mathsf{obs}(\mathsf{step}(s, P(s)), e)$. That is to say, P never chooses to transmit the information i to any domain in E.

Together, these two properties are sufficient to prove the desired security requirement that the information i never reaches domain u. In more formal detail, we can prove the following theorem:

- For a Rushby system with information I, distinct domains t, u, and f, and information unit $i \in I$;
- assuming the information message passing axioms as defined in Section 3 hold; and
- assuming noninterference holds for an information flow policy $\rightsquigarrow$; and
- assuming $\rightsquigarrow$ satisfies the information flow property defined above; and
- assuming P is a program for f satisfying the program property above: then
- for all action sequences α that respect P:
- it holds that $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha), u)$.

[3]Though technically, the proof below still works unchanged if $t = f$.

With the requirements defined as above, the proof of this property is actually pretty trivial. By induction, we can prove that $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha), e)$ for all $e \in E$. For $\alpha = \alpha' \cdot a$, we can recognize three cases:

- Either $\mathsf{dom}(a) \in E$, in which case by induction $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha'), e)$. From the message passing axioms it follows that $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha' \cdot a), e)$; or
- $\mathsf{dom}(a) = f$, in which case $a = P(\mathsf{step}(s_0, \alpha'))$. By induction we have $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha'), e)$, and from the property of P we get $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha' \cdot a), e)$; or
- no communication path from $\mathsf{dom}(a)$ to e exists that does not pass through f, which in particular means that $\mathsf{dom}(a) \not\rightsquigarrow e$. By noninterference it follows that $\mathsf{obs}(\mathsf{step}(s_0, \alpha' \cdot a), e) = \mathsf{obs}(\mathsf{step}(s_0, \alpha'), e)$, and thus by induction $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha' \cdot a), e)$.

Together these cases show that $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha), e)$ for all $e \in E$, which in particular means that $i \notin \mathsf{obs}(\mathsf{step}(s_0, \alpha), u)$.

We hold that the existence of this proof suggests that the theories proposed in Sections 3 and 4 are sensible models of the phenomena they describe. Certainly, it implies that both theories are sufficiently powerful to describe the necessary components in a formal verification of a system that is reasonably realistic.

We feel that the simplicity of the correctness proof above is a strong indication that the model of information and domain programs used here is a fairly natural one. To further support this claim, we codified both the theories in this paper and the correctness proof above in the logic of the Isabelle/HOL theorem proving system [4]; this serves the dual purpose of both verifying the correctness of the proof, and determining whether the proof is as simple as the paper version above suggests. We can confirm that the proof is both correct, and as simple as we had hoped; this in stark contrast to the complicated technical circumlocutions necessary to finish many other computer-verified proofs of system correctness. For reference, the Isabelle/HOL proof script making up this formalization is available on http://www.win.tue.nl/~jschmalt/publications/mils16/mils16.html .

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a theory modeling the detailed flow of information in a system based on a separation kernel described by Rushby's noninterference formalization. We also described a method for specifying the contribution of domain applications to the realization of formally verified properties. We have shown that this combination yields a practical formal verification framework through the case study of the verification of a variant of the GWV Firewall system, in an effort backed up by the logic of the Isabelle/HOL theorem proving system.

One thing that remains unclear is the degree to which the simplified theory of information used in this verification is necessary, as opposed to basing verification on the real information theory of which the model presented here is, ultimately, a crude approximation. Formalizing a verification framework that takes into account the full tenets of information theory, probability theory, cryptography, and related subjects remains a daunting task, and tools for performing actual properties on actual systems using this framework even more so. On the other hand, if successful, such a project could vastly expand the range of properties and systems for which formal verification is feasible, while rooting existing verifications in ever more solid ground; the authors shall follow any new approaches in this area with great interest.

Acknowledgments

We acknowledge funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n^o 318353 (EURO-MILS project: http://www.euromils.eu).

7. REFERENCES

[1] R. V. der Meyden. Remarks on the gwv firewall. Available at http://www.cse.unsw.edu.au/˜meyden/research/gwv-firewall.pdf, October 2010.

[2] D. Greve, M. Wilding, and W. M. Vanfleet. A separation kernel formal security policy. In *Fourth International Workshop on the ACL2 Theorem Prover and Its Applications*, July 2003.

[3] R. Koolen and J. Schmaltz. Formal methods for MILS: Formalisations of the GWV firewall. In *International Workshop on MILS: Architecture and Assurance for Secure Systems*, January 2015.

[4] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. 2002.

[5] J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical report, December 1992.

[6] J. Rushby. A separation kernel formal security policy in PVS. Technical report, Computer Science Laboratory, SRI international, 2004.

[7] R. van der Meyden. What, indeed, is intransitive noninterference? In J. Biskup and J. LÃşpez, editors, *Computer Security âĂŞ ESORICS 2007*, volume 4734 of *Lecture Notes in Computer Science*, pages 235–250. Springer Berlin Heidelberg, 2007.