This project has received funding from the European Unions Seventh Framework Programme for research, technological development and demonstration under grant agreement no 318353

D21.4
Formal Framework for MILS Integration

Project number:	318353
Project acronym:	EURO-MILS
Project title:	EURO-MILS: Secure European Virtualisation for Trustworthy Applications in Critical Domains
Start date of the project:	1st October, 2012
Duration:	36 months
Programme:	FP7/2007-2013

Deliverable type:	Report
Deliverable reference number:	ICT-318353 / D21.4 / 1.0
Activity and Work package contributing to deliverable:	Activity 2 / WP 2.1
Due date:	January 2016 – M40
Actual submission date:	3rd February, 2016

Responsible organisation:	TUE
Editor:	J. Schmaltz
Dissemination level:	PU
Revision:	1.0 (r-2)

Abstract:	This document presents our research results about the development of a formal environment to analyse MILS architectures. Its second chapter presents and discusses three existing approaches. This chapter shows their shortcomings. The third chapter proposes possible extensions of Rushby's model to address them.
Keywords:	separation kernel, intransitive noninterference, Isabelle/HOL, firewall application

Editor

J. Schmaltz (TUE)

Contributors (ordered according to beneficiary numbers)

H. Blasum (SYSGO)

B. Langestein (DFKI)

B. Leconte (AOS)

K. Müller (EADS IW)

F. Verbeek (OUNL)

R. Koolen (TUE)

J. Schmaltz (TUE)

Disclaimer
The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 318353.

Executive Summary

To achieve security certification according to the highest levels of assurance, formal models and proofs of security properties are required. In the MILS context, this includes formalisation of key components – such as separation kernels – and the formalisation of applications built on top of these verified components. In the second chapter of this document, we use the Isabelle/HOL proof assistant to formalise the Firewall application built on top of a verified separation kernel according to the model of Greve, Wilding, and Vanfleet (GWV). This Firewall application has been formalised twice after the original effort by GWV. These different efforts have been compared and discussed on paper. Our main contribution is to provide a formal comparison between these formalisations in the formal logic of a proof assistant.

In the third chapter of this document, we extend Rushby's model of noninterference with explicit between-domain information transfer, as well as programs that determine domain behaviour. 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 analyse an example system inspired by the GWV Firewall.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To achieve security certification at the highest levels of assurance (e.g. EAL6 or EAL7 of the Common Criteria), formal models and proofs are required. In the context of MILS architectures, this not only means the formalisation of key components, like separation kernels, but also the formalisation of more mundane applications and their composition in a complete system. Within the EURO-MILS project, we aim at providing a modelling and validation environment based on a generic MILS architecture. This environment should ease the development of formal models and proofs of systems built according to the MILS architectural paradigm.

Chapter 2 discusses three previous efforts about the application of formal methods to a system built on top of a separation kernel. This application is a Firewall originally proposed and formalised by Greve, Wilding, and Vanfleet [GWV03], who also proved its correctness using the ACL2 theorem prover [KMJ00]. This formalisation was later replicated by Rushby [Rus04], who proved the relevant properties in the logic of the PVS [ORS92] proof system; to do so, he also refined the axiomatisation of the Firewall behaviour. Subsequently, a further refinement of this formalisation was proposed by Van der Meyden [dM10], who also proves relations between the three efforts using informal pen-and-paper proofs. The main contribution of Chapter 2 is to formalise Van der Meyden's axiomatisation and proofs in the Isabelle/HOL proof assistant [NPW02]. We also re-formulate in Isabelle/HOL the formalisations by GWV and Rushby and the relations between the three axiomatisations. As part of this effort, we found a small flaw in Van der Meyden's axiomatisation, for which we present a corrected version. The conclusion of this chapter is that all current approaches to the formal verification of MILS-like architectures make unrealistic assumptions.

In Chapter 3, we propose an extension of Rushby's noninterference model that makes it possible to formally reason about the communication behaviour of applications running on top of a separation kernel proven to respect a given information flow policy. The goal of this extension is to leverage the unrealistic assumptions made in existing approaches. The main contribution of our new 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. To achieve this, we introduce an explicit notion of information and express the flow of this information between domains. We introduce explicit domain programs to make it possible to reason about the specific actions run inside domains. We illustrate the applicability of our new model by revisiting the firewall application introduced by Greve, Wilding, and Vanfleet.

Chapter 4 presents concluding remarks and points to further research directions.

Chapter 2

Formalisations of the GWV firewall

This chapter presents and discusses three existing efforts about the formal verification of an application built on top of a verified separation kernel. In the next three sections, we introduce the Firewall example application, the GWV model of formalised security, and express the Firewall in term of the GWV model. Afterwards, we compare the three different axiomatisations of the Firewall, proving relevant relations between them. We point out a flaw in the axiomatisation of Van der Meyden, and present a corrected version. Finally, we formally show how all three axiomatisations are sufficient to prove the desired properties of the larger system containing the Firewall, which we take as the compositional overall correctness proof.

2.1 The Firewall Application

The application Greve, Wilding, and Vanfleet used as an example of their formalisation of security is one that sanitises useful but sensitive information for use by an untrusted application. This so-called Firewall takes as input information presented by trusted parts of the system, which may be sensitive. It then filters and censors this information to produce a version that can safely be passed to applications that are not trusted to handle it securely, and delivers this sanitised information to a location where the untrusted application can find it. Under the assumption that the Firewall application is the only source of information to the untrusted application, this should provably ensure that no sensitive information ever ends up within reach of the untrusted application.

The Firewall application does not exist in a vacuum. It runs on top of an operating system of some sort, specified in more detail in the next section, which provides controlled access to memory. Its job is to divide the system memory into segments and enforce limits on which applications can access which memory segments. To ensure security, it is assumed that the operating system is configured in such a way that the Firewall application is the only component that can write to memory segments that are accessible to the untrusted application; moreover, it can only write to a single such a segment, denoted as **outbox**. This configuration is depicted in Figure 2.1.

For the purpose of the Firewall example, Greve, Wilding and Vanfleet do not try to express in detail what information is and is not sensitive. Instead, they assume the presence of a predicate **black** that, for a given memory segment and system state, expresses whether or not that segment contains only nonsensitive information. Thus, in a given state, a memory segment is **black** if and only if its contents are not sensitive. The function of the Firewall, then, is to make sure it only ever writes black data to **outbox**. The security requirement we seek to formalise can then be expressed as requiring that none of the memory segments accessible to the untrusted application ever becomes nonblack. The main goal of the different formalisations presented in the remainder of this paper is to prove that this is the case under the assumption that the

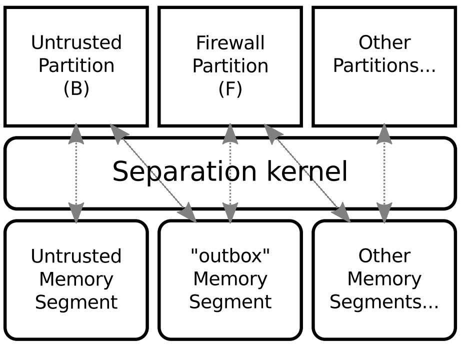

Figure 2.1: The Firewall MILS Example.

operating system and Firewall work as specified.

2.2 The GWV model of separation

The system model proposed by Greve, Wilding and Vanfleet [GWV03] (GWV) guarantees a security property called Separation. Extensions and variations of this model have then been proposed [WGW10, GWRV04] and discussed [Gre10, AFT04]. We will nonetheless use the original GWV model [GWV03], which is sufficient for the purposes of this paper.

The GWV model defines a mathematical formulation of systems similar to the one presented in Figure 2.1. At its base, a GWV system is a deterministic state machine, with states denoted s or t. Execution consists of repeatedly changing from a state s to the *next state* denoted **next**(s). A GWV system contains a finite set of *memory segments*, which in each state s have *contents* denoted **select**(s, a) for segment a. Furthermore, it contains a finite set of *partitions*, which represent independent subcomponents akin to processes in general-purpose operating systems. In any state, a single partition is currently active and executing; this partition is denoted **current**(s) for state s. This basic model is formalised in Isabelle parlance using the following axioms:

$$\texttt{fixes}\ \textbf{current}\ ::\ \mathsf{State} \Rightarrow \mathsf{Partition}$$
$$\texttt{fixes}\ \textbf{select}\ ::\ \mathsf{State} \Rightarrow \mathsf{Segment}::\texttt{finite} \Rightarrow \mathsf{Value}$$
$$\texttt{fixes}\ \textbf{next}\ ::\ \mathsf{State} \Rightarrow \mathsf{State}$$

Here, State, Partition, Segment, and Value represent arbitrary sets. The finiteness condition on Partition is irrelevant for the correctness of any of the formalised proofs and has been omitted. Finiteness of Segment, on the other hand, turns out to be crucial.

The GWV model assumes that the different partitions run on top of a *separation kernel*, a basic operating system tasked with the duty of restricting memory access of partitions to those accesses that satisfy a given *security policy*. One part of this security policy is a set of memory segments **segs**(p) for each partition p describing the memory segments that that partition is allowed to access. A more subtle component of the security policy is an *information flow policy*, represented by a binary relation between memory segments, with the semantics that any computation step that writes to memory segment a may only do so while reading from a limited set of input memory segments. Thus, information may only flow along the edges of the directed graph represented by the information flow policy. GWV formalise the information flow policy as a function **dia**, short for *direct interaction allowed*, which for each memory segment a returns a set **dia**(a) of memory segments that are allowed to directly influence it:

```
fixes segs :: Partition ⇒ ℙ(Segment)
fixes dia  :: Segment ⇒ ℙ(Segment)
```

The security requirement enforced by the separation kernel can then be expressed as requiring that all memory accesses must respect the security policy. Greve, Wilding, and Vanfleet call this property *separation*. Rather than describing how a separation kernel might enforce such a security policy, GWV define separation as requiring that all actions performed by a partition must be independent of the contents of memory segments that are not allowed to influence the action. Specifically, they require that whenever a partition writes to a memory segment a, the contents written may depend only on the contents of the segments that are both in the accessible segments of the executing partition and **dia**(a):

assumes **Separation :** $\forall s, t \in \mathsf{State}, a \in \mathsf{Segment}.$

$$\begin{gathered}\mathbf{current}(s) = \mathbf{current}(t) \wedge \\ \mathbf{select}(s, a) = \mathbf{select}(t, a) \wedge \\ \forall b \in \mathbf{dia}(a) \cap \mathbf{segs}(\mathbf{current}(s)).\ \mathbf{select}(s, b) = \mathbf{select}(t, b) \\ \rightarrow \mathbf{select}(\mathbf{next}(s), a) = \mathbf{select}(\mathbf{next}(t), a)\end{gathered}$$

This definition reads that for any segment a, for any states for which both the contents of a and the active partition are equal, the contents of a in the next state must be a function of the contents of the memory segments that are both readable by the active partition and allowed to influence a. Thus, in changing the contents of a, the executing partition may not make use of any information other than that allowed by the security policy.

In the GWV system model, the presence and correct functioning of a separation kernel is taken as an assumption, as formalised by the Separation axiom. Greve, Wilding, and Vanfleet propose that this axiom is a useful base for proving security properties of larger systems that rely on a separation kernel as a key component. They use their Firewall application as an example of how to prove security properties of a larger system by relying on the separation kernel as a provider of the base security infrastructure.

2.3 Firewall in GWV

In this section, we formally define both the Firewall application and the security property it is supposed to provably establish.

The Firewall application is a partition F that collects sensitive information from unspecified locations in the system, and passes a sanitised version of this information along to a different partition, B, that cannot be trusted to handle it securely. Relying on the separation kernel to ensure that no other partitions can write to memory segments accessible to the untrusted application, this should ensure that the untrusted application can only ever get access to information that has been judged safe by the Firewall.

GWV satisfy this information flow property as the specific requirement that there is a single memory segment **outbox** accessible to B which may be influenced by segments accessible to partitions other than B. Furthermore, any such segments that can influence **outbox** can only be accessible by F and B. Formally:

```
fixes B :: Partition
fixes F :: Partition
fixes outbox :: Segment

assumes FW_Pol: ∀a, b ∈ Segment, P ∈ Partition.
  a ∈ segs(B) ∧
  b ∈ dia(a) ∧
  b ∈ segs(P) ∧
  P ≠ B →
  (P = F ∧ a = outbox)
```

Together with the Separation axiom described in the previous section, this should suffice to ensure that the only information that ends up in segments accessible to B is information that the Firewall put there.

As described in Section 2.1, the behaviour of the Firewall is modelled using the **black** predicate, which models the distinction between sensitive and nonsensitive information. A memory segment is black in a given state if the contents of that segment in that state does not contain any sensitive information. The security functionality of the Firewall, then, is that it never writes any information to **outbox** that would cause it to become nonblack. This can be formalised as the proposition that **outbox** never changes from black to nonblack while the Firewall partition F is executing:

```
fixes black :: State ⇒ Segment ⇒ 𝔹

assumes FW_Blackens: ∀s ∈ State.
  current(s) = F ∧ black(s, outbox) →
  black(next(s), outbox)
```

We can now state a formal definition of the correctness of the Firewall application. The desired security property of the complete system including the Firewall, the untrusted application, and any possible other applications is that the segments accessible to B never become nonblack. The requirement that the segments of B *start* black is not a property of the Firewall; the weaker property that can be guaranteed by the Firewall is that the segments of B are already black, they will *remain* black. Introducing a function **run** to express the execution of a number of computation steps, this can be formalised as follows:

```
fun run :: ℕ ⇒ State ⇒ State where
  run(0, s) = s
  run(Suc(n), s) = run(n, next(s))

theorem FW_Correct: ∀s ∈ State, n ∈ ℕ, a ∈ segs(B).
  black(s, a) → black(run(s, n), a)
```

The combination of the **Separation**, **FW_Blackens**, and **FW_Pol** axioms is insufficient to prove the **FW_Correct** property. For this to be the case, we need further properties describing the behaviour of the **black** predicate; for example, if black data in the segments accessible to B could become nonblack on its own accord, the **FW_Correct** security property quickly falls apart. It is in the axiomatisation of the **black** predicate that GWV, Rushby, and Van der Meyden differ in their approaches. These three approaches will be the topic of the next section.

2.4 Axiomatisatising blackness

The behaviour of the Firewall is defined in terms of the **black** predicate, which models the property of a segment of memory of not containing any sensitive information. It would be expected for this property to satisfy certain axioms, such as the proposition that a segment cannot change from black to nonblack without the segment contents being modified. Certainly, if a nonsensitive chunk of memory were suddenly to become sensitive without anyone touching that piece of memory, this would violate our assumptions on what sensitivity is supposed to mean.

In their publications, GWV, Rushby, and Van der Meyden take different approaches in characterising the assumed behaviour of the **black** predicate. The basic notion of all three approaches is that nonblack data cannot be generated from black data; any computational process that takes only nonsensitive data as its input must surely produce output that is also nonsensitive. In this section, we compare the three different axiomatisations of this notion, and prove relevant relations between them.

2.4.1 The GWV Formalisation

One of the main properties that GWV require the **black** predicate to have is that in a system in which all segments are black, all segments will remain black; this is a special case of the "no spontaneous generation of nonblack data" principle described above. Another property they require is that blackness is a function of the content of a memory segment; it is not allowed that the same data is considered black or nonblack depending on the context, as this would allow sensitive data to leak into a completely idle partition.

`assumes` S5: $(\forall a \in \mathsf{Segment}.\mathbf{black}(s,a)) \rightarrow$
  $(\forall a \in \mathsf{Segment}.\mathbf{black}(\mathbf{next}(s),a))$
`assumes` S6: $\mathbf{select}(s,a) = \mathbf{select}(t,a) \rightarrow$
  $\mathbf{black}(s,a) = \mathbf{black}(t,a)$

These two properties are not sufficient to prove all desired properties of blackness, however. In particular, we would like to be able to prove a version of S5 restricted to a particular partition: the proposition that when all segments of a partition P are black in a state in which P is active, then all these segments will still be black in the next state. A lemma like this has an obvious role to play in any potential proof of **FW_Correct**.

To make this possible, GWV assume that for every state s and any segment a, a state can be constructed in which a is black but which is otherwise identical to s; such a state could be constructed by, say, wiping the contents of the memory segment a. To formalise this notion, they posit the existence of a function **scrub** producing such a state $\mathbf{scrub}(a,s)$, with straightforward properties:

`fixes` **scrub** `::` $\mathsf{Segment} \Rightarrow \mathsf{State} \Rightarrow \mathsf{State}$

`assumes` S1:
  $\mathbf{scrub}(a,\mathbf{scrub}(b,s)) = \mathbf{scrub}(b,\mathbf{scrub}(a,s))$
`assumes` S2:
  $a \neq b \rightarrow \mathbf{select}(\mathbf{scrub}(b,s),a) = \mathbf{select}(s,a)$
`assumes` S3:
  $\mathbf{black}(\mathbf{scrub}(b,s),a) \leftrightarrow (a = b \vee \mathbf{black}(s,a))$

`assumes S4:`
  $\textbf{current}(\textbf{scrub}(a, s)) = \textbf{current}(s)$

Axioms S1, S2, and S4 together specify that **scrub** does not change anything relevant about a state other than the contents of the scrubbed segment; axiom S3 specifies that a scrubbed segment is black.

With these properties, the lemma sketched above can be proven. For if all segments accessible to partition P are black in a state s with $\textbf{current}(s) = P$, we can construct a state t in which all segments are black by scrubbing all other segments; per axiom S5, the next state $\textbf{next}(t)$ of t also has all P-accessible states black. But s and t have the same contents of all memory segments accessible to P; according to the Separation axiom, the same must hold for $\textbf{next}(s)$ and $\textbf{next}(t)$. Thus, per S6, because all P-accessible segments in $\textbf{next}(t)$ are black, the same must hold for $\textbf{next}(s)$.

The axioms S1 ... S6 together with the **FW_Blackens**, **FW_Pol**, and **Separation** properties are sufficient to prove the desired **FW_Correct** theorem. We shall prove this by showing that the GWV axioms are stronger than the Rushby axioms, and that the Rushby axioms are sufficient to prove **FW_Correct**, both claims of which are described in the section below.

2.4.2 Rushby's Version

The formalisation proposed by Rushby is a reasonably minor refinement of the original by GWV. Like GWV, Rushby includes the two main properties from the GWV formalisation:

`assumes B4:` $\textbf{select}(s, a) = \textbf{select}(t, a) \rightarrow$
  $\textbf{black}(s, a) = \textbf{black}(t, a)$
`assumes B5:` $(\forall a \in \mathsf{Segment}.\textbf{black}(s, a)) \rightarrow$
  $(\forall a \in \mathsf{Segment}.\textbf{black}(\textbf{next}(s), a))$

The difference between the two formalisations is that instead of a function **scrub** that replaces the contents of a single segment by a blackened version, Rushby posits a function **blacken** that for a given state scrubs all segments that are not black.

`fixes` **blacken** `::` $\mathsf{State} \Rightarrow \mathsf{State}$

`assumes B1:`
  $\textbf{black}(\textbf{blacken}(s), a)$
`assumes B2:`
  $\textbf{black}(s, a) \rightarrow \textbf{select}(s, a) = \textbf{select}(\textbf{blacken}(s), a)$
`assumes B3:`
  $\textbf{current}(s) = \textbf{current}(\textbf{blacken}(s))$

It is clear that this axiomatisation is quite similar to GWV's original. The lemma that was proven for the GWV formalisation can be proven for Rushby's version in a very similar way. Furthermore, it is clear that the Rushby axioms follow from the GWV axioms: the **blacken** function can be constructed by calling **scrub** on each segment that is not black[1], and the resulting function clearly satisfies the B1, B2, and B3 axioms.

Unfortunately, formalising this fact in Isabelle proved challenging. In the logic of Isabelle, the fact that $\mathsf{Segment}$ is finite is expressed as a nonconstructive assertion that a function f ::

[1]Here the fact that the number of segments is finite is critical: without this requirement, the function **blacken** could not be defined based on **scrub**.

$\mathbb{N} \Rightarrow \mathsf{Segment}$ and a natural number n exist such that the segments $f(m)$ for $0 \leq m < n$ exactly cover $\mathsf{Segment}$. Because this is a nonconstructive assertion, we can only similarly prove in a nonconstructive way that a function **blacken** must exist that behaves like a repeated application of **scrub**. Both proving that this function exists, and working with this function to prove properties such as B1 ... B3 about it, are technically challenging; moreover, they result in cumbersome proofs that are difficult to understand due to the technical tricks intermixing the substance of the argument. We consider this a weakness of the Isabelle proof system; a more readable way of dealing with nonconstructively existing entities would be a welcome improvement. Moreover, to avoid turning this section into an unreadable mess of proof trickery, we have omitted the formal proof that the GWV axioms imply the Rushby axioms.

Similarly to the previous section, we prove that the axioms B1 ... B5 combined with the **FW_Blackens**, **FW_Pol**, and **Separation** properties are sufficient to prove the desired **FW_Correct** theorem by reducing this problem to the related problem for Van der Meyden's axiom. Doing so will be the subject of the remainder of this paper.

2.4.3 Van der Meyden's Axiom

Van der Meyden argues that these axioms defined by GWV and Rushby unnecessarily restrict the class of systems to which the results apply. He proposes an alternative formulation consisting of one axiom over predicate **black** without the need of ancillary functions, and thus the richness of the state space they imply. Rushby's axiom B5 basically states that if the entire system is black, then this is still the case in the next state. Van der Meyden generalises this notion by requiring that if the value of a memory segment is computed based only on the values of a set of memory segments X, and all segments in X are black, then the computed segment must be black in the next state.

The notion of *being computed based only on a set of memory segments* is just the same that GWV use to define the Separation axiom. In the Separation axiom, the requirement of the security policy is that the content of a memory segment in the next state is a function of its current content, the active partition, and the contents of the memory segments that are allowed to influence it. Using the same construction, Van der Meyden defines his sole axiom formalising **black** as follows:

```
fun equals :: $\mathbb{P}(\mathsf{Segment}) \Rightarrow \mathsf{State} \Rightarrow \mathsf{State} \Rightarrow \mathbb{B}$ where
  $\mathbf{equals}(X, s, t) = \forall a \in X.\mathbf{select}(s, a) = \mathbf{select}(t, a)$

assumes Black: $\forall X \in \mathbb{P}(\mathsf{Segment}), s \in \mathsf{State},$
      $a \in \mathsf{Segment}.$
  $(\forall r, t \in \mathsf{Segment}.\mathbf{equals}(X, r, t) \wedge$
    $\mathbf{current}(r) = \mathbf{current}(t) \rightarrow$
    $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a)) \wedge$
  $(\forall b \in X.\mathbf{black}(s, b)) \rightarrow$
  $\mathbf{black}(\mathbf{next}(s), a)$
```

This axiom states that if the value of segment a in the next state of s is a function of the segments in X and the active partition, then this function preserves blackness. In other words, as the value of a is computed based on X, so is the blackness of a inherited from X.

The **Black** axiom follows from the Rushby axioms; indeed, the proof for this in Isabelle is very simple. The proof starts by assuming the two premises stated in the **Black** axiom:

```
fix $X$ $s$ $a$
assume 1:
  $(\forall r, t \in \text{Segment}.\mathbf{equals}(X, r, t) \land$
  $\mathbf{current}(r) = \mathbf{current}(t) \rightarrow$
  $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a))$
assume $(\forall b \in X.\mathbf{black}(s, b))$
```

Because a is already black in s and is thus unaffected by **blacken** due to B2, and because **blacken** does not change the active partition of s, according to assumption 1 we have $\mathbf{select}(\mathbf{next}(s), a) = \mathbf{select}(\mathbf{next}(\mathbf{blacken}(s)), a)$.

```
hence $\mathbf{select}(\mathbf{next}(s), a) = \mathbf{select}(\mathbf{next}(\mathbf{blacken}(s)), a)$
  by (metis 1 B2 B3 equals_def)
```

But of course $\mathbf{blacken}(s)$ is black for all segments; by B5, so is $\mathbf{next}(\mathbf{blacken}(s))$. Because by B4 blackness is a function of the contents of a memory segment, we get

```
thus $\mathbf{black}(\mathbf{next}(s), a)$
  by (metis B1 B4 B5)
```

which completes the proof.

Unfortunately, the **Black** axiom is *not* sufficient to show the **FW_Correct** security requirement. This problem is the topic of the next section.

2.5 Preservation of blackness

In the paper introducing the **Black** axiomatisation of the **black** predicate[dM10], Van der Meyden appears to prove that together with the **FW_Pol**, **FW_Blackens**, and **Separation** axioms, the **Black** axiom is sufficient to prove the **FW_Correct** theorem specifying the secure operation of the Firewall. Unfortunately, this proof is incorrect, and the **Black** axiom is in fact not strong enough to ensure that the **black** predicate is sufficiently well-behaved.

To prove **FW_Correct**, Van der Meyden correctly shows that partitions other than B, including the Firewall partition F, can never make any segments of B nonblack. He also shows that B can never make any segments other than **outbox** nonblack without some other segment already being nonblack. A problem occurs, however, in proving that B can never make **outbox** nonblack. For this to be the case due to the **Black** axiom, there needs to be a set of segments X such that the next contents of **outbox** is a function of the active partition and the contents of the segments in X.

Van der Meyden shows that a set of segments X exists such that among all states s for which $\mathbf{current}(s) = B$, the contents of **outbox** in $\mathbf{next}(s)$ is a function of the contents of the segments in X. That is, he constructs a set X such that for all segments r and t for which $\mathbf{current}(r) = \mathbf{current}(t) = B$, if $\mathbf{equals}(X, r, t)$ holds, then we can conclude $\mathbf{select}(\mathbf{next}(r), \mathbf{outbox}) = \mathbf{select}(\mathbf{next}(t), \mathbf{outbox})$. Based on the "no spontaneous generation of nonblack data" intuition, we would expect this to be sufficient to show that $\mathbf{black}(\mathbf{next}(s), \mathbf{outbox})$ holds under the assumption that $\mathbf{current}(s) = B$ and all segments in X are black in s, and Van der Meyden argues exactly this in his proof of **FW_Correct**. This does not follow from the **Black** axiom, however.

Indeed, there is nothing in the **Black** axiom that requires a computation step of B to maintain the blackness of **outbox** when all segments accessible to B are black. This would require the next contents of **outbox** to be a function of the active partition and $\mathbf{segs}(B)$. But in general, this is not the case; the Firewall partition generally writes to **outbox** based on segments not accessible to B, which means that the next content of **outbox** is not independent of those segments. Appendix 2.7 describes a specific counterexample in which this is the case, showing that the **Separation**, **FW_Pol**, **FW_Blackens**, and **Black** axioms can all be true while **FW_Correct** is false.

To remedy this, we propose a stronger version of the **Black** axiom that does not suffer from this problem, which we feel better formalises the intuition behind the **Black** axiom. In his flawed proof, Van der Meyden inadvertently argues that the next contents of **outbox** are a function of the active partition and the contents of a set X of segments *among those states s for which* **current**$(s) = B$; because this predicate **current**$(s) = B$ is true for the specific state he is considering, he concludes that blackness follows for **outbox** in the next state of this specific state. We feel this line of reasoning should hold for **black** for arbitrary predicates of s. That is, if the functionality of segment a on segments X property holds for all states matching some predicate P, and all segments of X are black in a state s also matching this predicate P, then a should be black in the next state of s. Formally:

```
assumes StrongBlack: $\forall P \in \mathbb{P}(\mathsf{State})$,
        $X \in \mathbb{P}(\mathsf{Segment}), s \in \mathsf{State}, a \in \mathsf{Segment}.$
  $(\forall r, t \in \mathsf{Segment}. P(r) \wedge P(t) \wedge$
    $\mathbf{equals}(X, r, t) \wedge$
    $\mathbf{current}(r) = \mathbf{current}(t) \rightarrow$
    $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a)) \wedge$
  $(\forall b \in X. \mathbf{black}(s, b)) \wedge$
  $P(s) \rightarrow$
  $\mathbf{black}(\mathbf{next}(s), a)$
```

This definition is a generalisation of **Black**; using $P(s) = \mathsf{true}$ yields **Black** as a special case. Using this stronger axiom, Van der Meyden's proof does hold; the problem described above no longer applies, and the rest of the proof goes through unchallenged.

We feel that the **StrongBlack** axiom is a more accurate characterisation of the idea that non-black data cannot be generated from black data. The added predicate makes it possible to show more fine-grained instances of subsystems only having access to black data, and conclude the expected consequences of that fact for these limited cases. In the next section, we demonstrate how this notion can be used to conclude useful properties of **black**.

Unlike the **Black** axiom, the **StrongBlack** axiom does not follow from Rushby's formalisation, and neither does the Rushby formalisation follow from the **StrongBlack** axiom. The two formalisations are formally incomparable. For both directions, the reason that the implication does not hold is straightforward. Rushby's **blacken** function requires the existence of a large set of states, including lots of states in which all segments are black; the **StrongBlack** axiom, however, can easily hold in systems in which particular segments are always black. Conversely, because the **blacken** function does not preserve arbitrary predicates P, it is of no help in proving the **StrongBlack** axiom for arbitrary values of P. Constructing specific counterexamples for both implications is left as an exercise for the reader.

Like Rushby's axiomatisation, the **StrongBlack** axiom is sufficient to prove the correctness of the Firewall when combined with the **Separation**, **FW_Pol**, and **FW_Blackens** postulates. We will prove both in the next section.

2.6 Proving FW_Correct

In this section, we formally prove the correctness of the Firewall under the GWV, Rushby, and **StrongBlack** axiomatisations of the **black** predicate. That is, for these three axiomatisations, we prove that those axioms combined with the **Separation**, **FW_Pol**, and **FW_Blackens** axioms together imply the security property **FW_Correct**.

As described in Section 2.4.2, the Rushby axioms follow from the GWV axioms. We can therefore prove the correctness of both using a single proof that uses only the Rushby axioms as an assumption. No such luck applies to the Rushby and **StrongBlack** axiomatisations, however.

To avoid having to prove the same property twice for the Rushby and **StrongBlack** axioms, we first construct an axiomatisation that is weaker than either. This axiomatisation only functions as an artefact of proof; it does not aim to fully characterise the **black** predicate, but is only there to simplify the proofs. The axiomatisation we have in mind is a variant of **StrongBlack** that generalises **Black** in a minimal way while still being powerful enough to support the attempted usage in Van der Meyden's proof:

```
assumes WeakBlack: $\forall X \in \mathbb{P}(\mathsf{Segment}), s \in \mathsf{State},$
        $a \in \mathsf{Segment}.$
  $(\forall r, t \in \mathsf{Segment}.\mathbf{current}(s) = \mathbf{current}(r) \wedge$
    $\mathbf{equals}(X, r, t) \wedge$
    $\mathbf{current}(r) = \mathbf{current}(t) \rightarrow$
    $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a)) \wedge$
  $(\forall b \in X.\mathbf{black}(s, b)) \rightarrow$
  $\mathbf{black}(\mathbf{next}(s), a)$
```

The **WeakBlack** axiom is a special case of the **StrongBlack** axiom produced by substituting the predicate $P(t) \equiv \mathbf{current}(s) = \mathbf{current}(t)$ for the variable P; thus, it trivially follows from **StrongBlack**. More interestingly, it also follows from Rushby's axioms, using almost exactly the same proof as the one in Section 2.4.3:

```
fix $X$ $s$ $a$
assume 1:
  $(\forall r, t \in \mathsf{Segment}.$
  $\mathbf{current}(s) = \mathbf{current}(t) \wedge \mathbf{current}(r) = \mathbf{current}(t) \wedge$
  $\mathbf{equals}(X, r, t) \rightarrow$
  $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a))$
assume $(\forall b \in X.\mathbf{black}(s, b))$
hence $\mathbf{select}(\mathbf{next}(s), a) = \mathbf{select}(\mathbf{next}(\mathbf{blacken}(s)), a)$
  by (metis 1 B2 B3 equals_def)
thus $\mathbf{black}(\mathbf{next}(s), a)$
  by (metis B1 B4 B5)
```

When using the **WeakBlack** axiom instead of **Black**, Van der Meyden's proof is correct. We prove this below by presenting Van der Meyden's proof in fully formalised form in the Isabelle proof system.

The theorem we want to prove is that **FW_Correct** holds under the assumption of the **Separation**, **FW_Blackens**, **FW_Pol**, and **WeakBlack** axioms:

```
theorem
assumes Separation: $\forall s, t \in \text{State}, a \in \text{Segment}.$
  $\textbf{equals}(\textbf{dia}(a) \cap \textbf{segs}(\textbf{current}(s)), s, t) \land$
    $\textbf{current}(s) = \textbf{current}(t) \land$
    $\textbf{select}(s, a) = \textbf{select}(t, a) \to$
  $\textbf{select}(\textbf{next}(s), a) = \textbf{select}(\textbf{next}(t), a)$

assumes FW_Pol: $\forall a, b \in \text{Segment}, P \in \text{Partition}.$
  $a \in \textbf{segs}(B) \land$
  $b \in \textbf{dia}(a) \land$
  $b \in \textbf{segs}(P) \land$
  $P \neq B \to$
  $(P = F \land a = \textbf{outbox})$

assumes FW_Blackens: $\forall s \in \text{State}.$
  $\textbf{current}(s) = F \land \textbf{black}(s, \textbf{outbox}) \to$
  $\textbf{black}(\textbf{next}(s), \textbf{outbox})$

assumes WeakBlack: $\forall X \in \mathbb{P}(\text{Segment}), s \in \text{State},$
      $a \in \text{Segment}.$
  $(\forall r, t \in \text{Segment}.\textbf{current}(s) = \textbf{current}(r) \land$
    $\textbf{equals}(X, r, t) \land$
    $\textbf{current}(r) = \textbf{current}(t) \to$
    $\textbf{select}(\textbf{next}(r), a) = \textbf{select}(\textbf{next}(t), a)) \land$
  $(\forall b \in X.\textbf{black}(s, b)) \to$
  $\textbf{black}(\textbf{next}(s), a)$

shows $\forall s \in \text{State}, n \in \mathbb{N}.$
  $(\forall a \in \textbf{segs}(B).\textbf{black}(s, a)) \to$
  $(\forall a \in \textbf{segs}(B).\textbf{black}(\textbf{run}(n, s), a))$
```

Proof of this property is ultimately by induction on n. To simplify things, we first prove the correctness of the induction step as a lemma.

```
proof -
have 0: $\forall s \in \text{State}, a \in \text{Segment}.$
  $(\forall b \in \textbf{segs}(B).\textbf{black}(s, b)) \to$
  $a \in \textbf{segs}(B) \to \textbf{black}(\textbf{next}(s), a)$
proof -
fix $s$ $a$
assume 1: $\forall b \in \textbf{segs}(B).\textbf{black}(s, b)$
assume 2: $a \in \textbf{segs}(B)$
```

We now need to prove that $\textbf{black}(\textbf{next}(s), a)$. This proof will proceed by cases, but first we prove the simple lemma that **FW_Pol** applies to a: in other words, that if something can influence a, then a must be **outbox** and that something must be accessible only to B and F. This is a triviality, but proving it here will spare us the effort of having to duplicate the proof in all the cases that make use of it.

with **FW_Pol** have 3:
  $\forall b \in \mathsf{Segment}, P \in \mathsf{Partition}.$
  $b \in \textbf{segs}(P) \to P \neq B \to$
  $b \in \textbf{dia}(a) \to (a = \textbf{outbox} \wedge P = F)$
by simp

The proof of $\textbf{black}(\textbf{next}(s), a)$ will proceed by cases. We first consider the case where $a \neq \textbf{outbox}$.

show $\textbf{black}(\textbf{next}(s, a))$
proof cases
assume 4: $a \neq \textbf{outbox}$

Because $a \neq \textbf{outbox}$, by **FW_Pol** and **Separation** it follows that the next contents of a are a function of the contents of $\textbf{segs}(B)$ and the active partition. The **WeakBlack** axiom then requires that the blackness of the segments in $\textbf{segs}(B)$ which we assumed in assumption 1. To show this, we first need to establish the functional dependence of a on $\textbf{segs}(B)$ and the active partition as a lemma to later feed to **WeakBlack**.

have $\forall r, t \in \mathsf{State}.$
  $\textbf{current}(s) = \textbf{current}(r) \to$
  $\textbf{current}(r) = \textbf{current}(t) \to$
  $\textbf{equals}(\textbf{segs}(B), r, t) \to$
  $\textbf{select}(\textbf{next}(r), a) = \textbf{select}(\textbf{next}(t), a)$
proof auto
fix r t
assume 5: $\textbf{current}(s) = \textbf{current}(t)$
assume 6: $\textbf{current}(r) = \textbf{current}(t)$
assume 7: $\textbf{equals}(\textbf{segs}(B), r, t)$

Because $a \in \textbf{segs}(B)$ and $\textbf{equals}(\textbf{segs}(B), r, t)$, we have $\textbf{select}(r, a) = \textbf{select}(t, a)$.

with 2 have 8: $\textbf{select}(r, a) = \textbf{select}(t, a)$
  unfolding equals_def by simp

Because of **FW_Pol** and the fact that $a \neq \textbf{outbox}$, we must have $\textbf{dia}(a) \subseteq \textbf{segs}(B)$. Because $\textbf{equals}(\textbf{segs}(B), r, t)$ and $(X \cap Y) \subseteq X$, we certainly have the following:

from 3 4 7 have
  $\textbf{equals}(\textbf{dia}(a) \cap \textbf{segs}(\textbf{current}(r)), r, t)$
unfolding equals_def by auto

But then **Separation** gives us the desired result:

with 6 8 **Separation** show
  $\textbf{select}(\textbf{next}(r), a) = \textbf{select}(\textbf{next}(t), a)$
by simp
qed

This finishes the lemma stating that the next contents of a are a function of the contents of $\textbf{segs}(B)$ and the active partition. The **WeakBlack** axiom will now prove the blackness of a in $\textbf{next}(s)$.

with 1 **WeakBlack** show $\textbf{black}(\textbf{next}(s), a)$ by auto

This finishes the case where $a \neq$ **outbox**.

Next, we make a further case distinction on the value of the active partition in s. We consider three cases: $\mathbf{current}(s) = B$, $\mathbf{current}(s) = F$, and $\mathbf{current}(s) \neq B \wedge \mathbf{current}(s) \neq F$.

```
next assume 4: $a \neq \mathbf{outbox}$
show ?thesis proof cases
assume 5: $\mathbf{current}(s) = B$
```

The case where $\mathbf{current}(s) = B$ uses the same structure as the $a \neq$ **outbox** case. It first proves the lemma that the next contents of a are a function of the contents of $\mathbf{segs}(B)$ and the active partition.

```
have $\forall r, t \in \mathsf{State}.$
  $\mathbf{current}(s) = \mathbf{current}(r) \rightarrow$
  $\mathbf{current}(r) = \mathbf{current}(t) \rightarrow$
  $\mathbf{equals}(\mathbf{segs}(B), r, t) \rightarrow$
  $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a)$
proof auto
fix $r$ $t$
assume 6: $\mathbf{current}(s) = \mathbf{current}(t)$
assume 7: $\mathbf{current}(r) = \mathbf{current}(t)$
assume 8: $\mathbf{equals}(\mathbf{segs}(B), r, t)$
with 2 have 9: $\mathbf{select}(r, a) = \mathbf{select}(t, a)$
  unfolding equals_def by simp
```

The details of this step are different from the $a \neq$ **outbox** case, however. Here, the next contents of a are only a function of the contents of $\mathbf{segs}(B)$ and the active partition for those states s with $\mathbf{current}(s) = B$; in other words, this is the point in the proof in which the difference between **WeakBlack** and **Black** is crucial, and it's the point where Van der Meyden's original proof is incorrect. Because $\mathbf{current}(r) = \mathbf{current}(s)$, we have $\mathbf{equals}(\mathbf{dia}(a) \cap \mathbf{segs}(\mathbf{current}(r)), r, t)$:

```
from 5 6 7 8 have
  $\mathbf{equals}(\mathbf{dia}(a) \cap \mathbf{segs}(\mathbf{current}(r)), r, t)$
unfolding equals_def by simp
```

The remainder of this case proceeds in the same way as the case where $a \neq$ **outbox**.

```
with 7 9 Separation show
  $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a)$
by simp
qed
with 1 WeakBlack show $\mathbf{black}(\mathbf{next}(s), a)$ by auto
```

This concludes the case where $\mathbf{current}(s) = B$.

The case where $\mathbf{current}(s) = F$ is almost trivial; blackness of a in $\mathbf{next}(s)$ follows immediately from its blackness in s and **FW_Blackens**:

```
next assume 5: $\mathbf{current}(s) \neq B$
show ?thesis proof cases
assume 6: $\mathbf{current}(s) = F$
from 1 2 4 have $\mathbf{black}(s, \mathbf{outbox})$ by simp
with 4 6 FW_Blackens show ?thesis by simp
```

All that remains is the case where both $\mathbf{current}(s) \neq B \wedge \mathbf{current}(s) \neq F$. This, too, is a simple case: the security policy forbids the active partition from modifying any of the segments of B, which means the same proof used for the $a \neq \mathbf{outbox}$ case applies.

```
next assume 6: $\mathbf{current}(s) \neq F$
have $\forall r, t \in \mathsf{State}.$
  $\mathbf{current}(s) = \mathbf{current}(r) \rightarrow$
  $\mathbf{current}(r) = \mathbf{current}(t) \rightarrow$
  $\mathbf{equals}(\mathbf{segs}(B), r, t) \rightarrow$
  $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a)$
proof auto
fix $r$ $t$
assume 8: $\mathbf{current}(s) = \mathbf{current}(t)$
assume 9: $\mathbf{current}(r) = \mathbf{current}(t)$
assume $\mathbf{equals}(\mathbf{segs}(B), r, t)$
with 2 have 10: $\mathbf{select}(r, a) = \mathbf{select}(t, a)$
  unfolding equals_def by simp
with 3 5 6 8 9 have
  $\mathbf{equals}(\mathbf{dia}(a) \cap \mathbf{segs}(\mathbf{current}(r)), r, t)$
unfolding equals_def by auto
with 9 10 $\mathbf{Separation}$ show
  $\mathbf{select}(\mathbf{next}(r), a) = \mathbf{select}(\mathbf{next}(t), a)$
by simp
qed
with 1 $\mathbf{WeakBlack}$ show $\mathbf{black}(\mathbf{next}(s), a)$ by auto
qed qed qed qed
```

Because this is the last case, this finishes the proof of the lemma which states the correctness of the induction step of the main theorem. That is, we just proved $\forall s, a.(\forall b \in \mathbf{segs}(B).\mathbf{black}(s, b)) \rightarrow a \in \mathbf{segs}(B) \rightarrow \mathbf{black}(\mathbf{next}(s), a)$.

With this lemma in hand, we can now easily prove the main theorem, with an appeal to the lemma 0 in the induction step:

```
shows $\forall s \in \mathsf{State}, n \in \mathbb{N}.$
  $(\forall a \in \mathbf{segs}(B).\mathbf{black}(s, a)) \rightarrow$
  $(\forall a \in \mathbf{segs}(B).\mathbf{black}(\mathbf{run}(n, s), a))$
proof -
fix $s$ $n$
assume $\forall a \in \mathbf{segs}(B).\mathbf{black}(s, a)$
then show $\forall a \in \mathbf{segs}(B).\mathbf{black}(\mathbf{run}(n, s), a)$
proof (induction $n$, auto)
fix $n$ $x$
assume 2: $\forall x \in \mathbf{segs}(B).\mathbf{black}(\mathbf{run}(n, s), x)$
assume 3: $x \in \mathbf{segs}(B)$
with 0 2 show $\mathbf{black}(\mathbf{next}(\mathbf{run}(n, s)), x)$ by simp
qed qed qed
```

This completes the proof.

2.7 Counterexample to Van der Meyden's axiom

In Section 2.5, we described how a system can be constructed that satisfies the **Separation**, **FW_Pol**, **FW_Blackens**, and **Black** axioms, while still not satisfying **FW_Correct**. For the sake of completion, we provide here a specific minimal system for which this is the case.

Consider a GWV system as described in Section 2.2 consisting of two segments **outbox** and **inbox**, a partition B with $\textbf{segs}(B) = \{\textbf{outbox}\}$, and a second partition F with $\textbf{segs}(F) = \{\textbf{outbox}, \textbf{inbox}\}$. The **dia** function does not constrain any types of influence: $\textbf{dia}(a) = \{\textbf{outbox}, \textbf{inbox}\}$ for both values of a. The system has three possible states, named S_1, S_2, and S_3. The three states succeed each other in a cycle: $\textbf{next}(S_1) = S_2$, $\textbf{next}(S_2) = S_3$, and $\textbf{next}(S_3) = S_1$.

The contents of the memory, its sensitivity, and the active partition are summarized in Table 2.1. The F partition is active in the states S_1 and S_2, and B is active in S_3. The contents of **outbox** are equal for states S_1 and S_2, and different for S_3; the contents of **inbox** are equal for S_1 and S_3 and differnet for S_2. Of course, the exact values are irrelevant.

The **outbox** segment is black in states S_2 and S_3; the **inbox** segment is never black. This has the curious property that the states S_1 and S_2 have the same contents for the **outbox** segment, but differing blackness for that segment; the GWV and Rushby axiomatisations of the **black** predicate would not allow this, but it is possible under the **Black** axiom.

The system described here satisfies the **Separation** axiom. Because the **dia** function describes the complete relation and thus does not forbid anything, and because B does not write to **inbox** in the one state in which it is active, this is trivial.

Because B and F are the only partitions in the system and **outbox** is the only segment in $\textbf{segs}(B)$, **FW_Pol** is trivially satisfied. The **FW_Blackens** axiom is easily checked by verifying that $\textbf{black}(\textbf{next}(s), \textbf{outbox})$ is true for every state s with $\textbf{current}(s) = F$.

Showing that this system satisfies **Black** is less obvious. The value of **outbox** in the next state of s is not a function of the current value of $\{\textbf{outbox}\}$ and the active partition; indeed, the states S_1 and S_2 have the same active partition and the same values for **outbox**, yet $\textbf{select}(\textbf{next}(S_1), \textbf{outbox}) = 1 \neq 2 = \textbf{select}(\textbf{next}(S_2), \textbf{outbox})$. The value of **outbox** in the next stage of s *is* a function of the current value of $\{\textbf{inbox}\}$ and the active partition, and therefore also of the superset $\{\textbf{outbox}, \textbf{inbox}\}$. However, all we can conclude from this using the **Black** axiom is that if all segments in $\{\textbf{inbox}\}$ are black, then **outbox** must be black in the next state. Because **inbox** is never black, this is vacuously true. Thus, the system satisfies **Black** in a vacuous way.

The **WeakBlack** axiom, and therefore also the **StrongBlack** axiom, would note that among all states for which $\textbf{current}(s) = B$, the contents of **outbox** in the next state *is* a function of the active partition and the contents of $\{\textbf{outbox}\}$; indeed, it is even a function of the active partition and the contents of $\emptyset$. Thus, they would require that whenever **outbox** is black in a state for which B is the active partition, then **outbox** must still be black in the next state. This system, then, does not satisfy the **WeakBlack** axiom, and is not a counterexample against it.

s	$\textbf{next}(s)$	$\textbf{current}(s)$	$\textbf{select}(s, \textbf{outbox})$	$\textbf{select}(s, \textbf{inbox})$	$\textbf{black}(s, \textbf{outbox})$	$\textbf{black}(s, \textbf{inbox})$
S_1	S_2	F	1	3	false	false
S_2	S_3	F	1	4	true	false
S_3	S_1	B	2	3	true	false

Table 2.1: The Firewall MILS Example.

For the **Black** axiom, however, it is indeed a counterexample. This system does not satisfy **FW_Correct**; in state S_3 all segments of B are black, yet in the next state S_1, this is no longer the case. That makes this a system for the **Separation**, **FW_Pol**, **FW_Blackens**, and **Black** axioms are satisfied, yet **FW_Correct** does not hold, and a proof that Van der Meyden's proof is flawed.

2.8 Conclusion

In the previous section, we have proven the formal property **Separation** $\wedge$ **FW_Pol** $\wedge$ **FW_Blackens** $\wedge$ **WeakBlack** $\rightarrow$ **FW_Correct**. That is, we have shown that the desired security property that the untrusted application does not gain access to unprivileged information holds, under assumptions that the separation kernel and Firewall application behave in a certain way. An issue remains that the corrected axiom from Van der Meyden somehow assumes separation. This is not a satisfactory solution as the main objective is to obtain a clear decomposition of the responsibility. Regarding the firewall, we should only assume that it behaves as a firewall and not as firewall together with a separation kernel. The next chapter proposes a first solution to this issue.

Chapter 3

Modeling Information Routing with Noninterference

Many requirements have been proposed that formalise the behaviour of separation kernel; examples include Rushby's Noninterference [Rus92], Greve, Wilding and Vanfleet's Separation [GWV03], and many variations of these schemes. Formalising the behaviour of individual applications, on the other hand, is a problem for which few satisfactory approaches exist; as we have argued in the previous chapter, existing solutions —such as illustrated by the GWV Firewall [GWV03]— are too unrealistic to be usable for practical formalisations. A similar predicate holds for formalisations of the information flow between different applications; whereas separation kernel formalisations 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 verification challenges.

In this chapter, we propose an extension of Rushby's noninterference model that makes it possible to formally reason about the communication behaviour 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 3.2, 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 behaviour in Seection 3.3.

3. Finally, in Section 3.4, 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 formalised within the logic of the Isabelle/HOL theorem proving system [NPW02].

3.1 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, 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 [Rus92] introduced the notion of *noninterference* as a formal model of the services and guarantees provided by separation kernels. This automata-theoretic formalisation 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.

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 $t = step(s, a)$.

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 formalising strong separation properties.

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

Figure 3.1: Rushby's definition 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 behaviour of an operating system of sorts in which a domain structure can be recognised; it does not, as such, describe any guarantees regarding the level of domain isolation realised 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 3.1, 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 $\mathsf{obs}(s_0 \cdot \alpha, d) = \mathsf{obs}(s_0 \cdot \beta, d)$. 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 behaviour of an application in the context of a separation kernel; for in this analysis, the behaviours 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 behaviour 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 formalised 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 this system, thus specifying the behaviour 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 behaviour 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 [GWV03] example; indeed, this case study can be readily interpreted as the lifting of the GWV Firewall example to Rushby's noninterference formalisation.

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 [GWV03] as a similar test of their own separation kernel formalisation, and further studied by Rushby [Rus04] and Van der Meyden [dM10]. This use case is based on GWV's own formalisation of separation kernel behaviour, which they call Separation; this formalisation is quite different from Rushby's noninterference, and as a consequence the details of the specification of the firewall behaviour bears little resemblance to the version we use.

In the previous chapter, we argue that the communication model used in the construction of the GWV Firewall is too unrealistic to be useful in describing the behaviour of practical systems; while the correctness results in [GWV03] are correct, this is accomplished only by making assumptions on the communication structure that no realistic system can satisfy. As such, this chapter is a counterproposal of sorts, aiming to develop a model with a better applicability to practical systems.

3.2 Information Transfer

In this chapter, we do not attempt to specify the internal workings of a system to communicate information between domains in a coordinated way. Instead, we axiomatise 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 behaviour 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 analysing 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 behaviour 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 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. 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 two 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 synthesised from nothing.

The model sketched here can be formalised as an extension to the Rushby formalisation of operating systems. A *Rushby system with information* is a Rushby system as defined in Section 3.1, 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 characterise 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 domain 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 characterise 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 behaviour 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 3.4, we support this proposition through an example verification of an information flow property in a noninterference system using this model of information.

3.3 Domain Programs

In Rushby's formalisation 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 behaviour at all. When formalising of larger systems running on top of a separation kernel, this is only one piece out of many in the complete formalisation 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 behaviour 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 behaviour of the individual domains.

At first sight, the noninterference formalisation 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 3.1 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 behaviours, 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 a 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.[1]

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 behaviour 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 characterisation can be given that describes the behaviour of both the separation kernel, and the applications running in any of the domains we wish to specify. Using this characterisation, 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 behaviour of an example firewall system, based on properties assumed to hold for the firewall program.

3.4 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

[1] 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.

of a simple example system. This example system is based on GWV's Firewall example system [GWV03], and like the original it is centred around a domain that prevents certain sensitive information from reaching an untrusted receiver by scrutinising 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 to u. We assume that t, u and f are all distinct domains[2]. 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 formalise 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 behaviour of the firewall domain, a predicate can be described regarding the program P that determines its behaviour. Naively, one might try to specify the firewall behaviour as the requirement that the firewall program P never chooses to transmit information i to domain u; but this is insufficient, 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 behaviour 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 3.2 hold; and

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

- 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 $i \notin \mathsf{obs}(s_0 \cdot \alpha, u)$.

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

- Either $\mathsf{dom}(a) \in E$, in which case by induction $i \notin \mathsf{obs}(s_0 \cdot \alpha', e)$. From the information message passing axioms it follows that $i \notin \mathsf{obs}(s_0 \cdot \alpha' \cdot a, e)$; or
- $\mathsf{dom}(a) = f$, in which case $a = P(s_0 \cdot \alpha')$. By induction we have $i \notin \mathsf{obs}(s_0 \cdot \alpha', e)$, and from the property of P we get $i \notin \mathsf{obs}(s_0 \cdot \alpha' \cdot a, e)$; 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 $\mathsf{obs}(s_0 \cdot \alpha' \cdot a, e) = \mathsf{obs}(s_0 \cdot \alpha', e)$, and thus by induction $i \notin \mathsf{obs}(s_0 \cdot \alpha' \cdot a, e)$.

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

We hold that the existence of this proof suggests that the theories proposed in Sections 3.2 and 3.3 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 [NPW02]; 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 formalisation is available on `http://www.win.tue.nl/~jschmalt/publications/mils16/mils16.html`.

3.5 Conclusions

In this chapter, we introduced a theory modelling the detailed flow of information in a system based on a separation kernel described by Rushby's noninterference formalisation. We also described a method for specifying the contribution of domain applications to the realisation 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.

Chapter 4

Conclusions

The second chapter considered parts of the verification of a firewall application running on top of a separation kernel. The point of this exercise is to study techniques for the formal verification of system properties in a compositional way. In this chapter we did not prove any properties about the behaviour of system components such as the separation kernel or the Firewall; instead, these properties were taken for granted. The problem studied here is how to make use of independently proven properties describing individual system components to prove properties of the system as a whole in which these components play a role; that is, to compose verified behaviour of components into verified behaviour of the complete system.

When verifying practical systems, one would presumably independently prove behavioural properties regarding individual system components, and then later attempt the composition in a way similar to the methods used in this paper. Based on our experience formalising the notions presented in this paper, we feel confident that compositional formal validation of system properties is a practical technique for certifying desired system properties in applications such as security certification. The main issue with these approaches is that they are not fully compositional in the sense that the correct axiom for the firewall already assumes that the firewall itself has the property of separation, whereas this property should be ensure solely by the separation kernel.

The third chapter makes a first proposal towards a solution. It extends Rushby's model with an explicit notion of information and explicit domain programs. It revisits the firewall example showing a clear compositional solution. 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. Formalising 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; we shall follow any new approaches in this area with great interest.

Bibliography

[AFT04] Jim Alves-Foss and Carol Taylor. An analysis of the GWV security policy. In *In Fifth International Workshop on ACL2 Prover and Its Applications*, 2004.

[dM10] Ron Van der Meyden. Remarks on the gwv firewall. Available at http://www.cse.unsw.edu.au/~meyden/research/gwv-firewall.pdf, October 2010.

[Gre10] DavidA. Greve. Information security modeling and analysis. In David S. Hardin, editor, *Design and Verification of Microprocessor Systems for High-Assurance Applications*, pages 249–299. Springer US, 2010.

[GWRV04] David Greve, Matthew Wilding, Raymond Richards, and W. Mark Vanfleet. Formalizing security policies for dynamic and distributed systems. *Unpublished*, September 2004.

[GWV03] David Greve, Matthew Wilding, and W. Mark Vanfleet. A separation kernel formal security policy. In *Fourth International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2 '03)*, July 2003.

[KMJ00] M. Kaufmann, P. Manolios, and J S. Moore. ACL2 Computer-Aided Reasoning: An Approach, 2000.

[NPW02] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. 2002.

[ORS92] S. Owre, J.M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *Proceedings of the Eleventh International Conference on Automated Deduction (CADE'92)*, volume 607, pages 748–752, June 1992.

[Rus92] John Rushby. Noninterference, transitivity and channel-control security policies. Technical report, Computer Science Laboratory, SRI international, 1992.

[Rus04] John Rushby. A separation kernel formal security policy in PVS. Technical report, Computer Science Laboratory, SRI international, 2004.

[WGW10] Michael W. Whalen, David A. Greve, and Lucas G. Wagner. Model checking information flow. In David S. Hardin, editor, *Design and Verification of Microprocessor Systems for High-Assurance Applications*, pages 381–428. Springer US, 2010.