Inferring Constructive Labels from Semantics of Information Flow for Secure Operating Systems

Abstract

Lattice model of information flow proposed by Denning [8] held a lot of promise for securing multilevel systems, but its potential remained largely unutilized in practice. Myers and Liskov [27, 28] revived the field by allowing users to dynamically create labels that form the lattice, rather than use a pre-defined lattice to achieve end-to-end security guarantees. Since then, there has been much research on adapting their model for securing programs, operating systems and distributed systems. Usage of these systems is still not wide-spread mainly because of the abstractness of the labels.

In this paper, we present a label model based on set of readers and writers, which makes policy specification intuitive and simple, and checking of information flow efficient. Moreover, the labels are inferred from the semantics of information flow and therefore constructive in nature. Generality of the model is established by proving that every specification in Denning's model is captured. Further, the proposed model enables a clean combination with existing discretionary access control system and its verification for securing operating systems. Advantages of our approach in comparison with related work, in terms of simplicity of specification and label management functions, is discussed.

1 Introduction

The ability to control the release and propagation of information lies at the heart of systems security. Standard access control models [15, 35, 10, 36, 5, 9, 4, 6] control the release of information but do not provide the means for controlling its subsequent propagation. Lattice model of secure information flow proposed by Denning in [8] provides support for controlling information propagation by assigning security levels (come from a lattice) to data sets and subjects, and allowing information flow from a level to a higher or equal level.

The lattice model is a simple policy with several desirable features like compositionality, which makes it possible to specify and verify end-to-end security guarantees. The lattice model of secure information flow succinctly captures well known security models like the Bell-LaPadula model for secrecy/confidentiality [3], Biba's model for integrity [16], the Chinese-Wall security policy [7] etc.

Andrew Myers and Barbara Liskov [27] present the Decentralized Information Flow Control (DIFC) model which allows users to dynamically create labels to control the flow of their data rather than use a predefined lattice. DIFC became very popular, particularly because of the decentralized nature of its flow control and there exist several implementations and applications for securing the programs (viz., [26, 29, 38]), operating systems (for example [42, 47, 18, 48]) and distributed systems ([21, 1, 49, 33]). Indrajit Roy et al., introduced Laminar [32], a system to enforce DIFC at run-time using a single set of abstractions for both OS resources and heap-allocated objects.

The lattice model of information flow concerns itself with controlling the direct (i.e. via legitimate channels) information flows. Goguen and Meseguer introduced a notion of non-interference [12, 13] to control what the lower level processes can infer from the actions of high level processes. Non-interference property captures both direct and indirect (i.e., via some covert channels) information flows. For practical applications, we need limited ways in which information flows from high levels to lower levels. To achieve this, *declassification* (relaxes confidentiality policies), and *endorsement* (relaxes integrity policies) are introduced [30, 46, 2].

However, current DIFC systems are still not widely used because of the inability of common users to specify their information flow policy in terms of the various label models provided. This is true particularly in the case of DIFC operating systems. Each DIFC system proposes a new label model tuned to its target application. Some of the important label models proposed are the decentralized label model (DLM) [28]; label model of Asbestos OS [42] and HiStar OS [47]; label model of DStar system [49, 48], Laminar system [32] and Flume OS [18]; and

the disjunction category labels (DC labels) [40]. An assessment of the practical enforcement of information flow control in operating systems and the remaining challenges are nicely brought out by Lampson [19].

Our main objective and motivation is to provide a label model with an underlying computational framework which allows wider intuitive penetration and acceptance of DIFC systems. Note that (i) the ultimate purpose of an information flow model is to define/determine whether a subject can read/write an object, and (ii) most users are familiar with the concepts of discretionary access policies, in terms of which users are allowed to read and write their files. Moreover, it is important to note that even when the security policy of interest is only confidentiality / integrity of information, both reading and writing play a vital role.

In this paper, we propose the readers-writers flow model (RWFM) using a set of readers and writers as labels and prove that it is sound and complete with respect to Denning's lattice model. In the context of operating systems security, we demonstrate the usefulness of RWFM by showing how the labels can be automatically inferred from the existing discretionary policy. Further, RWFM also simplifies label management by utilizing the facilities already present in the operating systems. Some important merits of RWFM are:

- simple and intuitive policy specification,
- efficient checking of information flow,
- automatic extraction of labels from the policy,
- simple label management, and
- seamless integration with discretionary policies.

Rest of the paper is organized as follows: Section 2 provides the basics of Denning's lattice model. In section 3, we present the intuitions, define the readers-writers flow model and prove its completeness. Application of RWFM to operating systems security and comparison with related work is discussed in section 4, and section 5 provides concluding remarks and directions for future work.

2 Background

In this section, we introduce the lattice model of secure information flow.

Denning [8] introduced a lattice model for securing information flow in a system, which is derived from security classes and is justified by the semantics of information flow. The salient feature of this model is that it encompasses several well known models like the Bell-LaPadula model for cofindentiality/secrecy [3], Biba's integrity model [16], the Chinese-Wall security policy [7] etc.

Denning's information flow model (DFM) is defined by the five tuple $DFM = (S, O, SC, \oplus, \leqslant)$, where (i) S is a set of *subjects/principals* (active agents responsible for all information flow), (ii) O is a set of *objects* (information containers), (iii) SC is a set of *security classes*, (iv) $\oplus$ is the *class-combining binary operator* (associative and commutative) that specifies, for any pair of operand classes, the class in which the result of any binary function on values from the operand classes belongs, and (v) $\leqslant$ is a binary relation on security classes that specifies *permissible information flows*. $sc_1 \leqslant sc_2$ means that information in security class sc_1 is allowed/permitted to flow into security class sc_2.

Example 1. An example of security classes in DFM could be $SC = \{l_1, l_2\}$, with $l_1 < l_2$ as the ordering. This means that information at security class l_1 is allowed to flow to security class l_2, but not vice-versa.

A pictorial representation of this lattice along with more example lattices is given in Figure 1.

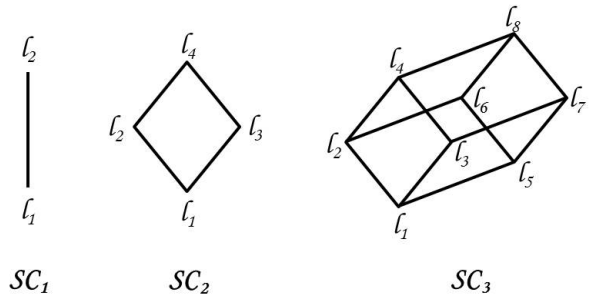

Figure 1: Hasse diagrams of some example information flow lattices

The pictorial representations are to be understood as follows: if there is an upward path from class l to l', then information is allowed to flow from class l to l'. For example, in SC_1, information is allowed to flow from class l_1 to l_2 but not vice-versa. Similarly, in SC_2, information is allowed to flow from l_2 to l_4 but not to l_1 or l_3. In SC_3, information is allowed to flow from l_3 to l_4, l_7 and l_8 but to no others. □

Subjects and objects are bound to security classes (either statically or dynamically depending on the application) by a labelling function, $\lambda : S \cup O \rightarrow SC$, that defines the *access control policy*. Note that, when a subject s reads an object o, information flows from o to s and this

is permissible only if $\lambda(o) \leqslant \lambda(s)$. Similarly, when a subject s writes an object o, information flows from s to o and this is permissible only if $\lambda(s) \leqslant \lambda(o)$.

Example 2. Consider the security lattice SC_1 given in Example 1. Let s_1 and s_2 be the only subjects in the system i.e., $S = \{s_1, s_2\}$. Similarly, let $O = \{o_1, o_2\}$.

An example access policy is given by $\lambda_1(s_1) = \lambda_1(o_1) = l_1$ and $\lambda_1(s_2) = \lambda_1(o_2) = l_2$. According to λ_1, s_1 can read o_1, because $\lambda_1(o_1) \leqslant \lambda_1(s_1)$ is satisfied by λ_1. Similarly, it is easy to verify that λ_1 permits s_1 to write o_1 and o_2 but not read o_2; and s_2 can read and write o_2 and can read but not write o_1.

Another policy could be defined by $\lambda_2(o_1) = l_1$ and $\lambda_2(s_1) = \lambda_2(s_2) = \lambda_2(o_2) = l_2$. If policy λ_2 is enforced, then both s_1 and s_2 are allowed to read and write o_2 and read but not write o_1. □

A system enforcing Denning's flow model DFM is *secure* if and only if execution of any sequence of operations of the system cannot give rise to a flow that violates the permissible information flow relation. Further, the natural conditions required of information flow force the structure $(SC, \leqslant)$ to be a lattice with $\oplus$ as the least upper bound operator.

3 Readers-Writers Flow Model

In this section, we shall define a unified flow model referred to as **Readers-Writers Flow Model** (`RWFM`), characterize its expressive power, demonstrate simplicity of `RWFM` specifications and discuss its merits for studying information flow properties.

Denning's flow model $DFM = (S, O, SC, \oplus, \leqslant)$ together with a labelling function $\lambda : S \cup O \rightarrow SC$ is used for determining whether a subject can read or write an object. In most applications of information flow models, the following happens: (i) permissible flows are defined on abstract entities (security classes) (ii) the answer to a concrete question, whether a subject can read/write an object, proceeds via the abstraction of security classes. Our motivation was to make security classes more concrete and intuitive so that the information flow models become widely used and applied without much difficulty.

We start by noting that most users are comfortable with the concepts of traditional discretionary access control systems. They understand security policies better if specified in terms of who can read/write a particular file. Let us see if we can utilize these concepts to define security classes that can be used for information flow control. Suppose o_1 is readable by s_1 and s_2, and o_2 is readable by s_1 but not by s_2, then information in o_1 should be allowed to flow to o_2 but not the other way. Similarly, suppose o_3 is writable by s_3 and s_4, and o_4 is writable by s_3 but not s_4, then information in o_4 should be allowed to flow to o_3 but not the other way. From the discussion above, we conclude that the permissible flows can be described in terms of set of subjects allowed to read an object and set of subjects allowed to write an object. Therefore, the set of readers and writers of an object identify its security class. As information flows to higher levels, the set of readers decrease and the set of writers increase.

We formally define the readers-writers security class (RW Class) as follows.

Definition 1 (RW **Class).** *A RW class, written (R, W), is a tuple of set of subjects in an information system, i.e., $R \in 2^S$ and $W \in 2^S$, where S is the set of subjects in the system and 2^S denotes the powerset of S.*

In a RW class, R denotes the set of subjects allowed to read objects of this class, and W denotes the set of subjects allowed to write objects of this class.

As per the discussion above, information is allowed to flow from one class to another only if readers decrease and writers increase.

Definition 2 (Permissible Flows). *Given any two RW classes, $RW_1 = (R_1, W_1)$ and $RW_2 = (R_2, W_2)$, information is allowed to flow from RW_1 to RW_2, denoted $RW_1 \leqslant_{RW} RW_2$, only if $R_1 \supseteq R_2$ and $W_1 \subseteq W_2$.*

The only component left to define is the least-upper bound (join) and the greatest-lower bound (meet) operators. Intuitively, join defines the least security class to which information from both the input classes is permitted to flow, and meet defines the highest security class from which information is permitted to flow into both the input classes. When information readable by subjects in R_1 is combined with information readable by subjects in R_2, the resulting information can only be read by subjects in both R_1 and R_2. Similarly, when information writable by subjects in W_1 is combined with information writable by subjects in W_2, the resulting information can be written by subjects in either W_1 or W_2. When information readable by subjects in R_1 is allowed to flow into information readable by subjects in R_2 and into information readable by subjects in R_3, then it should be the case that every subject in R_2 and R_3 is in R_1. When information writable by subjects in W_1 is allowed to flow into information writable by subjects in W_2 and into information writable by subjects in W_3, then it should be the case that every subject in W_1 is also in W_2 and W_3. These ideas are formalized in the definition of join and meet below.

Definition 3 (Join and Meet of RW Classes). *Let $RW_1 = (R_1, W_1)$ and $RW_2 = (R_2, W_2)$ be any two RW classes. Their join ($\oplus_{RW}$) and meet ($\otimes_{RW}$) are defined as*

$$RW_1 \oplus_{RW} RW_2 = (R_1 \cap R_2, W_1 \cup W_2)$$
$$RW_1 \otimes_{RW} RW_2 = (R_1 \cup R_2, W_1 \cap W_2).$$

3.1 Characterization of RWFM

In this section, we show that the ordering $\leqslant_{RW}$ on RW classes satisfy certain desirable properties.

Theorem 1 (Soundness). *The set of all RW classes $SC_{RW} = 2^S \times 2^S$, together with the ordering $\leqslant_{RW}$, join $\oplus_{RW}$ and meet $\otimes_{RW}$ form a bounded lattice with minimum element $\bot = (S, \emptyset)$ and maximum element $\top = (\emptyset, S)$.*

Proof. The proof is trivial and follows by observing that it is a product of two power-set lattices, the first one (readers lattice) ordered by reverse inclusion and the second (writers lattice) by inclusion. □

Theorem 1 means that, the readers-writers lattice satisfies the conditions required by Denning's formulation, and hence can be used for studying information flow properties.

Combining the above results leads us to the definition of readers-writers flow model (RWFM).

Definition 4 (Readers-Writers Flow Model). *Readers-Writers flow model RWFM is defined as a six tuple $(S, O, SC_{RW}, \leqslant_{RW}, \oplus_{RW}, \otimes_{RW})$, where S is the set of subjects and O is the set of objects in an information system, $SC_{RW} = 2^S \times 2^S$, $\leqslant_{RW} = (\supseteq, \subseteq)$, $\oplus_{RW} = (\cap, \cup)$ and $\otimes_{RW} = (\cup, \cap)$.*

The first component of the security class in a RWFM is to be interpreted as the set of readers, and the second component as the set of writers. Note that RWFM is fully defined by S and O only.

A flow model together with a labelling function defines the access policy. Let $\lambda : S \cup O \to SC_{RW}$ be a labelling function. For simplicity, we use $R_\lambda(e)$ and $W_\lambda(e)$ to denote the first and second components of the security class assigned to an entity (subject or object) e. Further, the subscript λ is omitted when it is clear from the context. Access rules in the flow model are defined below.

Definition 5 (Access Rules in RWFM). *Given a RWFM, and functions R and W describing a labelling,*

- *A subject s is allowed to read an object o if $R(o) \supseteq R(s)$ and $W(o) \subseteq W(s)$*

  and

- *A subject s is allowed to write an object o if $R(s) \supseteq R(o)$ and $W(s) \subseteq W(o)$.*

Information flows upwards in the lattice as readers decrease and writers increase.

Next we argue that, not only is the readers-writers flow model suitable for studying direct information flow, but it is also all we ever need.

Theorem 2 (Completeness). *Given a Denning's flow model $DFM = (S, O, SC, \oplus, \leqslant)$ and a policy $\lambda : S \cup O \to SC$, there exists a labelling, $\lambda_{RW} : S \cup O \to SC_{RW}$, in the readers-writers flow model that enforces the same policy i.e.,*
(i) s is permitted to read o by Denning's policy if and only if it is permitted by readers-writers policy
and
(ii) s is permitted to write o by Denning's policy if and only if it is permitted by readers-writers policy.

Proof. The proof is by construction. We define a labelling and prove that it satisfies the conditions.

For ease of understanding, λ_{RW} is defined by defining readers (R) and writers (W) components as follows. For objects, we want the labels to capture who can read and write it, and can be defined in a straight-forward manner as follows.

$$R(o) \triangleq \{s \mid \lambda(o) \leqslant \lambda(s)\} \quad \text{(D1)}$$
$$W(o) \triangleq \{s \mid \lambda(s) \leqslant \lambda(o)\} \quad \text{(D2)}$$

From these definitions, it is intuitively clear that the set of users who can read o (write o) is captured by $R(o)$ ($W(o)$). For subjects, labelling is tricky. For a subject s, let o_1, o_2, ..., o_n be the objects it can read, and let o'_1, o'_2, ..., o'_m be the objects it can write. The situation is depicted in Figure 2.

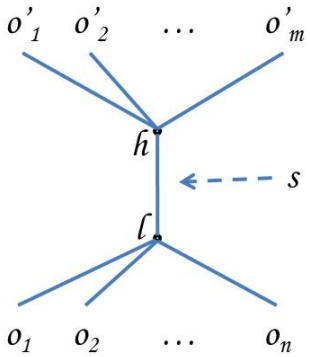

Figure 2: Intuition behind subject label

From the figure, it is clear that the label of the subject is lower bounded by the least upper bound of labels of o_1, o_2, ..., o_n (denoted by l in figure), and upper bounded by the greatest lower bound of labels of o'_1, o'_2, ..., o'_m (denoted by h in figure). Further note that, the label of s does not depend on the labels of readers of o'_1, o'_2, ..., o'_m and on the labels of writers of o_1, o_2, ..., o_n.

$$R(s) \triangleq S \cap (\bigcap_{\lambda(o) \leqslant \lambda(s)} R(o)) \quad \text{(D3)}$$

$$W(s) \triangleq S \cap (\bigcap_{\lambda(s) \leqslant \lambda(o)} W(o)) \quad \text{(D4)}$$

Intuitively, the fact that s should be able to read the objects it is allowed to read is captured by setting $R(s)$ to be the least upper bound[1] of the read labels of all objects it can read. Similarly, the fact that s should be able to write the objects it is allowed to write is captured by setting $W(s)$ to be the greatest lower bound[2] of the write labels of all objects it can write.

From $(D3)$ and $(D4)$, it easily follows that for all subjects s, $s \in R(s)$ and $s \in W(s)$. (P1)

Proof of (i) $\lambda(o) \leqslant \lambda(s)$ if and only if $R(s) \subseteq R(o)$ and $W(s) \supseteq W(o)$.

(**only if**). Given $\lambda(o) \leqslant \lambda(s)$ prove $R(s) \subseteq R(o)$ and $W(s) \supseteq W(o)$.

(1)	$\lambda(o) \leqslant \lambda(s)$	given
(2)	$R(s) \subseteq R(o)$	(1), (D3)

(1)	$s' \in W(o)$	assume
(2)	$\lambda(s') \leqslant \lambda(o)$	(1), (D2)
(3)	$\lambda(o) \leqslant \lambda(s)$	given
(4)	$\lambda(s') \leqslant \lambda(s)$	(2), (3)
(5)	$s \in W(o')$	assume
(6)	$\lambda(s) \leqslant \lambda(o')$	(5), (D2)
(7)	$\lambda(s') \leqslant \lambda(o')$	(4), (6)
(8)	$s' \in W(o')$	(7), (D2)
(9)	$s' \in W(s)$	(8), (D4)
(10)	$W(s) \supseteq W(o)$	(1), (9)

(**if**). Given $R(s) \subseteq R(o)$ and $W(s) \supseteq W(o)$ prove $\lambda(o) \leqslant \lambda(s)$.

(1)	$s \in R(s)$	(P1)
(2)	$R(s) \subseteq R(o)$	given
(3)	$s \in R(o)$	(1), (2)
(4)	$\lambda(o) \leqslant \lambda(s)$	(3), (D1)

Proof of (ii) $\lambda(s) \leqslant \lambda(o)$ if and only if $R(o) \subseteq R(s)$ and $W(o) \supseteq W(s)$.

(**only if**). Given $\lambda(s) \leqslant \lambda(o)$ prove $R(o) \subseteq R(s)$ and $W(o) \supseteq W(s)$.

(1)	$\lambda(s) \leqslant \lambda(o)$	given
(2)	$W(o) \supseteq W(s)$	(1), (D4)

(1)	$s' \in R(o)$	assume
(2)	$\lambda(o) \leqslant \lambda(s')$	(1), (D1)
(3)	$\lambda(s) \leqslant \lambda(o)$	given
(4)	$\lambda(s) \leqslant \lambda(s')$	(3), (2)
(5)	$s \in R(o')$	assume
(6)	$\lambda(o') \leqslant \lambda(s)$	(5), (D1)
(7)	$\lambda(o') \leqslant \lambda(s')$	(4), (6)
(8)	$s' \in R(o')$	(7), (D1)
(9)	$s' \in R(s)$	(8), (D3)
(10)	$R(o) \subseteq R(s)$	(1), (9)

(**if**). Given $R(o) \subseteq R(s)$ and $W(o) \supseteq W(s)$ prove $\lambda(s) \leqslant \lambda(o)$.

(1)	$s \in W(s)$	(P1)
(2)	$W(o) \supseteq W(s)$	given
(3)	$s \in W(o)$	(1), (2)
(4)	$\lambda(s) \leqslant \lambda(o)$	(3), (D2)

□

3.2 Illustrative Examples

In this section, we illustrate the encoding of Denning's policy in readers-writers policy using examples.

Example 3. Consider the policy λ_1 given in Example 2. $\lambda_1(s_1) = \lambda_1(o_1) = l_1$ and $\lambda_1(s_2) = \lambda_1(o_2) = l_2$. $s_1 \in R(o_1)$ because $\lambda_1(o_1) \leqslant \lambda_1(s_1)$ reduces to $l_1 \leqslant l_1$ which is true. $s_2 \in R(o_1)$ because $\lambda_1(o_1) \leqslant \lambda_1(s_2)$ reduces to $l_1 \leqslant l_2$ which is also true. Therefore $R(o_1) = \{s_1, s_2\}$. Similarly, we can derive the following labels on objects: $R(o_2) = \{s_2\}$, $W(o_1) = \{s_1\}$ and $W(o_2) = \{s_1, s_2\}$.

Notice that $\lambda_1(o_1) \leqslant \lambda_1(s_1)$ but $\lambda_1(o_2) \not\leqslant \lambda_1(s_1)$. Therefore, $R(s_1) = S \cap R(o_1) = \{s_1, s_2\}$. Similarly, $R(s_2) = S \cap R(o_1) \cap R(o_2) = \{s_2\}$, $W(s_1) = S \cap W(o_1) \cap W(o_2) = \{s_1\}$ and $W(s_2) = S \cap W(o_2) = \{s_1, s_2\}$.

Combining the above we get, $\lambda_{RW}(o_1) = \lambda_{RW}(s_1) = (\{s_1, s_2\}, \{s_1\})$ and $\lambda_{RW}(o_2) = \lambda_{RW}(s_2) = (\{s_2\}, \{s_1, s_2\})$. Note that $\lambda_{RW}(o_1) \leqslant_{RW} \lambda_{RW}(o_2)$.

[1]Note that the readers lattice is ordered by reverse inclusion and hence the least upper bound is defined by $\cap$

[2]Note that the writers lattice is ordered by inclusion and hence the greatest lower bound is defined by $\cap$

The original and the inferred policies are depicted in Figure 3.

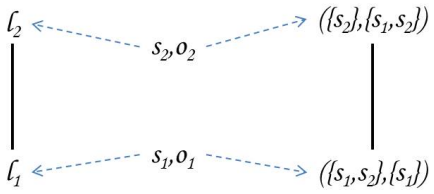

Figure 3: Denning's policy and corresponding readers-writers policy inferred in Example 3

□

Example 4. Let us consider a policy defined on the lattice SC_2 of Figure 1. Let $S = \{s_1, s_2\}$, $O = \{o_1, o_2\}$, and $\lambda_3(s_1) = l_1$, $\lambda_3(s_2) = l_4$, $\lambda_3(o_1) = l_2$ and $\lambda_3(o_2) = l_3$.

In this case, the labelling in RWFM comes out to be $\lambda_{RW}(o_1) = \lambda_{RW}(o_2) = (\{s_2\}, \{s_1\})$, $\lambda_{RW}(s_1) = (S, \{s_1\})$ and $\lambda_{RW}(s_2) = (\{s_2\}, S)$. The following ordering among labels is satisfied: $\lambda_{RW}(s_1) \leqslant_{RW} \lambda_{RW}(o_1) \leqslant_{RW} \lambda_{RW}(s_2)$. The original and the inferred policies are depicted in Figure 4.

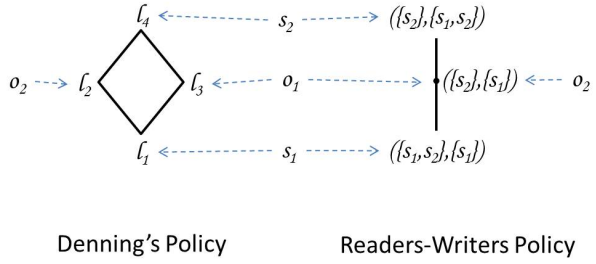

Figure 4: Denning's policy and corresponding readers-writers policy inferred in Example 4

□

Note that, in Example 3, the inferred readers-writers lattice is the same as the original lattice. However, in Example 4, the lattice resulting from the translation has strictly fewer elements than the original one. Moreover, o_1 and o_2 which had different labels in the original policy have same labels in the readers-writers policy. Still, Theorem 2 guarantees that the accesses allowed by both the policies will be the same. This is possible because, even though the original policy assigned two different labels to o_1 and o_2, the labels assigned to subjects are such that as far as reading and writing are concerned, the objects are indistinguishable i.e., readable and writable by the same set of subjects.

Examples 3 and 4 clearly illustrate that readers-writers policies are simpler to understand.

3.3 Properties of RWFM

In this section, we present some important properties of the readers-writers policy inferred from a lattice policy.

Proposition 1. *Let DFM with λ be a Denning's flow policy, and let R and W denote the corresponding labelling in the readers-writers flow model as given in Theorem 2. For any subject s and object o, the following holds:*
(i) $R(o) \supseteq R(s) \Rightarrow W(o) \subseteq W(s)$
and
(ii) $W(s) \subseteq W(o) \Rightarrow R(s) \supseteq R(o)$
where $\Rightarrow$ denotes logical implication.

Proof. We provide a proof of (i).

(1)	$R(o) \supseteq R(s)$	given
(2)	$s \in R(s)$	(P1) in Thm 2
(3)	$s \in R(o)$	(1), (2)
(4)	$\lambda(o) \leqslant \lambda(s)$	(3), (D1) in Thm 2
(5)	$W(o) \subseteq W(s)$	(4), **only if** part of (i) in Thm 2

Proof of (ii) is similar and the details are omitted. □

Definition 5 says that, a subject s can be allowed to read an object o if both $R(o) \supseteq R(s)$ and $W(o) \subseteq W(s)$ are satisfied. Proposition 1 simplifies this check to $R(o) \supseteq R(s)$. Similarly for writing.

In the proof of Theorem 2, we argued that intuitively $R(o)$ and $W(o)$ capture the set of subjects allowed to read and write o respectively. This is formalized in the proposition below.

Proposition 2. *Let DFM with λ be a Denning's flow policy, and let R and W denote the corresponding labelling in the readers-writers flow model as given in Theorem 2. For any subject s and object o, the following holds:*
(i) $s \in R(o) \Rightarrow R(o) \supseteq R(s)$
and
(ii) $s \in W(o) \Rightarrow W(s) \subseteq W(o)$
where $\Rightarrow$ denotes logical implication.

Proof. We provide a proof of (i).

(1)	$s \in R(o)$	given
(2)	$\lambda(o) \leqslant \lambda(s)$	(1), (D1) in Thm 2
(3)	$R(o) \supseteq R(s)$	(2), (D3) in Thm 2

Proof of (ii) is similar and the details are omitted. □

Proposition 2 further simplifies the access check to $s \in R(o)$ for s to read o, and $s \in W(o)$ for s to write o. Thus, *the model provides for intuitive specifications and also simplifies the algorithm for making access decisions*.

In section 4.1, we prove that not all discretionary access policies are information flow policies. In this context, propositions 1 and 2 provide us conditions to verify whether a given access policy is an information flow policy.

3.4 Discussion on Merits of RWFM

In an information system, information flow can only happen between a subject and an object. There are only two directions in which information can flow, either from subject to object or from object to subject. Although we have been using terms read and write because of their familiarity which aids intuition, note that read is representative of the class of operations which make information to flow from object to subject, and write is representative of the class of operations which make information to flow from subject to object. Thus, *the model is general and can be applied to any system by appropriately modelling the direction of information flow caused by its basic operations*.

Although Denning's lattice model can be used to study any information flow properties, it is most widely applied for studying confidentiality and integrity policies. Since Theorem 2 is applicable to any lattice policy, it readily captures the typical confidentiality and integrity policies. Further, *a combination of confidentiality and integrity policies can be modelled as a readers-writers policy, which is more intuitive and allows easy reasoning*.

In this context, it is important to observe that even when one wants to study only confidentiality or integrity, both readers and writers play a vital role, which is brought out clearly by our approach. This is a subtlety, which is mostly ignored. For example, in [28] and [40], authors roughly equate secrecy/confidentiality as controlling reading and integrity as controlling writing. At this point, it is important to make explicit the assumption we make in this argument: information systems that allow information flow in both directions. In the case of a server, whose purpose is only to provide information, confidentiality can be equated to controlling reading. Similarly, in the case of an opinion poll, for example, where information only flows into the system, integrity can be equated to writing. Note that in both the cases above, there is "information stagnation" and the system reaches a "dead-end" as far as usage of its information is concerned. Information systems that have to permit information flows in both directions, thus allowing "information circulation", are encountered in practice. Even in this case, it is uninteresting to consider a disjoint set of readers and writers i.e., some only "give" information to the system and some only "take" information from it. *In practice, information systems encountered frequently are such that there is a set of users who can both give and take information from the system. It is in this case, that both readers and writes play a vital role for both confidentiality and integrity, and in fact for any information flow property/policy*.

4 Application of RWFM to Secure Operating System Design

In this section, we demonstrate that the use of readers-writers policies for securing operating systems addresses some of the challenges and complements the enforcement mechanisms of current DIFC systems.

Some of the major challenges in the design and development of DIFC enabled operating systems are:

1. Specification
   (a) How easy is it to specify policies and manage them?
   (b) How well does the policy complement the existing discretionary policy?
2. Implementation
   (a) What additional infrastructure needs to be created for enforcing the given policy?
   (b) How automated is the algorithm for making access decisions?
   (c) Is the trusted computing base (TCB) small?
   (d) What is the performance overhead?
   (e) Do the existing applications continue to work without much change?

In the context of an operating system, processes and threads are the subjects, and filesystem, process memory, external devices etc are the objects.

In discretionary systems employed by current operating systems, access decisions are made based on the UID and GID of the process (subject). This means that what ever operations the user could have performed, are made available to all the processes executing on his behalf. This makes discretionary systems vulnerable to Trojan horses. Mandatory access control systems overcome this weakness by making a distinction between a user and a subject operating on his behalf. While the users are trusted, subjects operating on his behalf are not. For enforcing

information flow policies which are mandatory in nature, the system keeps track of the information accessed by a process so far to decide which further accesses should be allowed.

We start by noting that RW security classes are very similar to the existing discretionary policies. Indeed, the permission bits and ACLs of $Linux$ are specified in terms of which users are allowed to read and write the files/directories. The difference between these specifications is that while RW classes are associated with subjects, the discretionary policy associates permissions with users. However, note that for DIFC systems targeted towards a wider usage and acceptance, forcing users to specify policies in terms of subjects is unsuitable. Hence, we decide to work with policies specified in terms of users. In this case, we can automatically derive/infer security classes for objects based on the semantics of the discretionary policy of the system.

Example 5. Let F be a file in a standard Linux system. Let F be owned by user u, and let g_F denote its group whose members are users $u_1, .., u_i$. Further, let U denote the set of users in the system.

If F's permission bits be set to $rw-r--r--$, then the RW class of F is $(U, \{u\})$.

If F's permission bits be set to $-w-rw-r--$, then the RW class of F is $(U - \{u\}, \{u, u_1, ..., u_i\})$. □

Information content accessed by a process is denoted by an RW class (R, W) intuitively as "the information which is readable by users in R and written by users in W". Thus in comparison to labels of current DIFC systems, RW security class presents this information in a more intuitive manner.

4.1 Are all discretionary policies information flow policies?

In the following, we present an example of a discretionary policy which is not an information flow policy.

Example 6. Consider the following discretionary policy. File F_1 is readable by u_1 and u_2, and writable by u_1 but not by u_2. File F_2 is readable by u_1 and u_2, and writable by u_2 but not by u_1. This policy is not an information flow policy.

Proof. Assume the contrary. Let λ be a labelling which denotes the policy. The following conditions must be simultaneously satisfied.

(1)	$\lambda(F_1) \leqslant \lambda(u_1)$	F_1 readable by u_1
(2)	$\lambda(F_1) \leqslant \lambda(u_2)$	F_1 readable by u_2
(3)	$\lambda(u_1) \leqslant \lambda(F_1)$	F_1 writable by u_1
(4)	$\lambda(F_2) \leqslant \lambda(u_1)$	F_2 readable by u_1
(5)	$\lambda(F_2) \leqslant \lambda(u_2)$	F_2 readable by u_2
(6)	$\lambda(u_2) \leqslant \lambda(F_2)$	F_2 writable by u_2
(7)	$\lambda(u_1) \leqslant \lambda(u_2)$	(3), (2)
(8)	$\lambda(u_2) \leqslant \lambda(u_1)$	(6), (4)
(9)	$\lambda(u_1) = \lambda(u_2)$	(7), (8)

Which leads to a contradiction, because if $\lambda(u_1) = \lambda(u_2)$, then it cannot be the case that u_1 can read F_1 while u_2 cannot. □

However, note that the discretionary policy can be thought of as defining an upper limit on the permissible accesses. In a particular execution, a subject executing on behalf of a user has access to only a subset of the accesses permitted by the discretionary policy, and importantly this also depends on the order in which requests are made. Thus information flow control policies inherently support the *principle of least privilege* [34], by forcing the subjects to not gather more information than required for a computation. If this principle is not followed by a subject, information flow policy ensures that the information resulting from its computations becomes less useful by placing it in a higher security class.

4.2 Information Flow Tracking/Controlling

In the context of operating systems, it also helps to keep track of the owner of an information, particularly, for sensitive operations like downgrading. Hence, we make small technical changes to the labels as defined below.

Definition 6. *RW class is a triple $S \times 2^S \times 2^S$, where the first component denotes the owner of the object/subject, second component denotes the set of readers, and third component denotes the set of writers.*

Definition 7. *Information is allowed to flow from RW class (o_1, R_1, W_1) to class (o_2, R_2, W_2) if (i) $R_1 \supseteq R_2$, (ii) $W_1 \subseteq W_2$, (iii) $o_2 \in R_1$ if the flow is due to a read operation, and (iv) $o_1 \in W_2$ if the flow is due to a write operation.*

Subjects could be labelled statically or dynamically. In static labelling, user specifies the RW class with which a process is to be executed and all access decisions are based on this class. Note that relabelling to a higher security class is always safe. We use this idea for dynamic relabelling of processes.

Dynamic labelling of processes and objects:

Next, we describe how the information flow is tracked and controlled by dynamic relabelling based on operations that cause information flow.

1. When a process is created by a user u, it starts with the least security class possible i.e., $(u, S, \emptyset)$, where S is the set of users in the system.

2. If a process is created/spawned by an already executing process, the created process shall inherit the current security class of the creating process.

3. When a process executing on behalf of user u, and having a current security class of (u, R_1, W_1) makes a request to read a file labelled (u', R_2, W_2), this access is permitted (i) if $u \in R_2$ i.e., the access is permitted by the discretionary policy, and (ii) after relabelling the process to $(u, R_1 \cap R_2, W_1 \cup W_2)$, reflecting that the process plausibly inferred new information.

4. When a process executing on behalf of user u, and having a current security class of (u, R_1, W_1) makes a request to write a file labelled (u', R_2, W_2), this access is permitted (i) if $u \in W_2$ i.e., the access is permitted by the discretionary policy, and (ii) if $R_1 \supseteq R_2$ and $W_1 \subseteq W_2$ i.e., the access is permitted by the information flow policy.

5. When a process executing on behalf of user u, and having a current security class of (u, R, W) creates a new file, the file is labelled $(u, R, W \cup \{u\})$.

6. Labels of run-time objects like pipes and sockets used for interprocess communication float with the processes current label.

Our subject floating label system is inspired from [11, 24, 47, 42, 41]. In this context, note that the lowest and the highest classes that a subject executing on behalf of a user u can have are $(u, S, \emptyset)$ and $(u, \{u\}, S)$.

Root is assigned $(r, \emptyset, \emptyset)$. Further, root processes are **always trusted** and **exempted** from the access and labelling rules i.e., *label of a root process always remains* $(r, \emptyset, \emptyset)$.

Network is assigned the class (n, S, S). The security class of network is justified because:

- a subject with label (u, R, W) can send information to the network (write) if $R \supseteq S$ and $W \subseteq S$. $W \subseteq S$ is always satisfied, while $R \supseteq S$ forces $R = S$, which means that the subject has only read information which is world readable.
- a subject with label (u, R, W) can receive information from the network (read) if $R \subseteq S$ and $W \supseteq S$. $R \subseteq S$ is always satisfied, while $W \supseteq S$ forces $W = S$, which indicates that the subject has been influenced (contaminated) by information of all users.

4.3 Illustration of End-to-End Security and Downgrading

In this section, we illustrate the features of our approach using the web-tax example from [28].

Bob wishes to prepare his tax form with the help of an application $Web - Tax$, which is provided by $Preparer$. Further, Bob wishes to protect the privacy of his tax data and does not want it to flow to the network. $Preparer$ also provides a proprietary database used by the $Web - Tax$ application for preparing Bob's final tax form. $Preparer$ is interested in the security of the database and does not want its information to flow to the network. We use the following notation for this example: b stands for Bob, p stands for $preparer$, n stands for network, TD stands for Bob's tax data, DB stands for the proprietary database, IR stands for the intermediate results during computation and FF denotes the final tax form. The set of subjects $S = \{b, p, n\}$ and the set of objects $O = \{TD, DB, IR, FF\}$.

Various entities of the web-tax example and permissible flows between them are depicted in Figure 5.

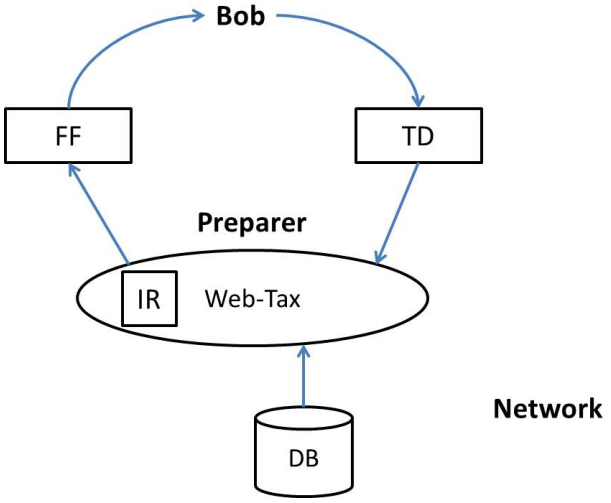

Figure 5: Permissible flows between entities of web-tax example

For expressing the desired security for various objects, the labelling is as follows: Bob labels TD as

$(b, \{b, p\}, \{b\})$ to denote that b owns it (first component), b and p can read it (second component), and b has written it (third component). Similarly, $Preparer$ labels DB as $(p, \{p\}, \{p\})$.

Note that because the network is labelled (n, S, S) neither TD nor DB can be sent to it because $(b, \{b, p\}, \{b\}) \not\leqslant (n, S, S)$ and $(p, \{p\}, \emptyset) \not\leqslant (n, S, S)$.

IR is labelled $(p, \{p\}, \{b, p\})$, because both TD and DB flow into it, second component is the intersection of second components of TD and DB, and third component is the union of third components of TD and DB. Note that Bob cannot read the intermediate results.

For the final form to be readable by Bob, an information flow which is against the permissible flows is needed. This is a recurring theme in practical information flow systems and is referred to as *downgrading*. In case of confidentiality policies downgrading is called *declassification*, and in the case of integrity policies it is called *endorsement*. As this is a sensitive operation, several conditions need to be imposed on its success. These are discussed below.

Definition 8 (Downgrading Rule 1)**.** *A subject s executing on behalf of user u can downgrade an object labelled (u', R, W) by relabelling it to (u, R', W) only if (i) he owns the object i.e., $u = u'$ and (ii) the object contains only his information i.e., $W = \{u\}$.*

This rule does not suffice for our situation and a slightly weaker rule is required as defined below.

Definition 9 (Downgrading Rule 2)**.** *A subject s executing on behalf of user u can downgrade an object labelled (u', R, W) by relabelling it to (u, R', W) only if (i) he owns the object i.e., $u = u'$ and (ii) he adds all those subjects whose data appears in it as readers i.e., $R' - R = W$.*

For the final form to be readable by Bob, $preparer$ downgrades (using Rule 2) the final form by relabelling it to $(p, \{b, p\}, \{b, p\})$. Label of the final form conveys the following: (i) p has prepared it (first component), (ii) both Bob and $preparer$ can read it (second component), and (iii) both Bob's data (TD) and $preparer$'s data (DB) went into preparing it.

Remark. Note that in the readers-writers policy, the writers component depicts two things: (i) who can write to this object and (ii) whose data was used in preparing the object. The second aspect also plays an important role. For example, when Bob finally submits his tax information, the tax authorities may need an endorsement from the preparer certifying that he prepared the form. This is readily reflected in our model.

In the decentralized label model (DLM) [28], label is a set of policies, each of which is of the form $o_i : r_{i_1}, ..., r_{i_n}$ which is to be understood as "owner o_is information is in the object and he stipulates that only $r_{i_1}, ..., r_{i_n}$ can read it". The effective set of readers of an object is the intersection of the reader sets of all owners. For example, Bob's tax data in the above example is labelled $\{b : b\}$ indicating that Bob owns it and only Bob can read it. Note that this information is readily conveyed by the readers-writers label.

Disjunction category (DC) labels [40] are tuples (S, I), where the first component protects privacy by controlling reading and the second component protects integrity by controlling writing; and both are represented as CNFs (negation-free propositional formula) over principals. Permissible flows are defined using logical implication. For example, the intermediate results in the above example are labelled $(\{b \wedge p\}, \{b \vee p\})$, denoting that only a subject authorized by both Bob and $preparer$ may read this information (first component), and either Bob's or $preparer$'s consent is required for writing into it.

A comparison of the labelling of the web-tax example in DLM [28], DC labels [40] and our approach (RW) is as given below.

	DLM	DC	RW
TD	$\{b : b\}$	$(\{b\}, \{b\})$	$(b, \{b, p\}, \{b\})$
DB	$\{p : p\}$	$(\{p\}, \{p\})$	$(p, \{p\}, \{p\})$
IR	$\{b : b; p : p\}$	$(\{b \wedge p\}, \{b \vee p\})$	$(p, \{p\}, \{b, p\})$
FF	$\{b : b\}$	$(\{b\}, \{b \vee p\})$	$(p, \{b, p\}, \{b, p\})$
n	$\{\}$	$(True, True)$	$(n, \{b, p, n\}, \{b, p, n\})$

DLM label of FF, $\{b : b\}$, says that Bob owns this information and only Bob can read it. We observe that although for the purposes of this example, the DLM label of FF is sufficient, if further computation was involved using FF, the DLM label does not convey complete information. This can be easily overcome if the $preparer$ had declassified IR by relabelling it to $\{b : b; p : b, p\}$ instead of $\{b : b\}$. The label $\{b : b; p : b, p\}$ while allows Bob to read it, does not allow it to be read by others. In particular, note that if Bob wishes, he may declassify $\{b : b\}$ to permit $Alice$ to read FF. Since the $preparer$ has interest in protecting his secret database whose information was used in preparing FF, relabelling it to $\{b : b; p : b, p\}$ permits this protection.

DC label of FF, $(\{b\}, \{b \vee p\})$, says that both Bob's information and $preparer$'s information are contained in it and only Bob can read it. RW label of FF, $(p, \{b, p\}, \{b, p\})$, says that $preparer$ owns this information, which can be read by both Bob and $preparer$, and this contains information from both Bob and $preparer$.

In this particular example, DC labels and RW classes

provide more accurate information than DLM. A full comparison of these models along with their expressive power will provide a good basis for the design of DIFC information systems.

4.4 Related Work

Insufficiency of discretionary policies (DAC) and the need for mandatory controls (MAC) for protection of operating systems is well understood [23]. Flask [39] is a classical system which supports multiple security policies for protection of operating systems. The Flask security architecture was later extended in [22] for use in standard Linux operating systems giving rise to SELinux [37].

SELinux currently supports both RBAC and MLS policies. We believe that `RWFM` presents a more intuitive policy specification, and it also becomes possible to easily verify policies in a system where a combination of several policies are used. Consider the case of collaboration between two organizations, where one organization's policy is specified in terms of RBAC and the others policy is in terms of information flow. In this case, `RWFM` provides a common framework for enforcement of interests of both the parties and also simpler verification of properties.

Before we consider the challenges of the DIFC operating systems, let us recollect the six design principles for usable access control systems as envisaged in [20].

1. Provide "good enough" security with a high level of usability, rather than "better" security with a low level of usability.
2. Provide policy, not just mechanism.
3. Have a well-defined security objective.
4. Carefully design ways to support exceptions in the policy model.
5. Rather than trying to achieve "strict least privilege", aim for "good-enough least privilege".
6. Use familiar abstractions in policy specification interface.

Usable Mandatory Integrity Protection (UMIP) [20] and Information Flow Enabled DAC (IFEDAC) [24] describe notable results which successfully combine MAC with DAC to protect the integrity of systems in the face of network attacks.

On the other hand, information flow control, which is a particular kind of mandatory policy, has gained a lot of prominence and attention in recent times because of its suitability for a wide class of applications. Subsequently, several DIFC enabled operating systems like Asbestos [42], HiStar [47, 48], Flume [18] and Laminar [32] have been developed.

Current DIFC operating systems are in conformance with the first five principles, but fail the sixth principle. This motivated us to work towards a specification interface which enables wider acceptance of the otherwise advantageous DIFC systems. Note that the label models of DIFC operating systems impose considerable burden on the end user to learn unfamiliar concepts, and the process of translating his security notions into this new language is error prone. On the other hand, `RWFM` policies are specified in terms of concepts that the users are acquainted with for several decades, which helps gain their acceptance and greatly reduces the possibility of errors that may arise in the translation process.

From an implementation perspective, the policy described in Section 4.2 could be implemented using the standard system call interposition [31] techniques. It could also be implemented using the useful abstractions developed exclusively for tracking/controlling information flow. Asbestos OS [42] provides the abstraction of *event processes* and tracks information flow at this level of abstraction. HiStar OS [47, 48] provides the abstraction of *containers* and five other object types, and tracks information flow at this level. Flume OS [18], which is actually a user-level library, presents an abstraction called *end points* and tracks all information flow at this level.

We believe that a simple initial implementation using system call interposition will help in systematically blocking information channels one after the other, while carrying out a performance and usability evaluation at the same time. Further, because of similarity of the policy concepts of `RWFM` with the existing operating system practices, we also believe that the underlying operating system architectures (like LSM [45], for example) could be leveraged to achieve an efficient implementation.

From a label generation and maintenance perspective, note that while other DIFC systems require mechanisms for generating labels (usually using cryptographic primitives) and their management, `RWFM` does not require any special effort on this. Further note that, because labels are sets of users, they could be denoted by bit vectors, and for computing with labels, we use only subset, superset, intersection and union operations which could be very efficiently implemented using "bit-wise and" and "bit-wise or" on currently hardware.

In summary, we conclude that `RWFM` contributes by simplifying the specification of DIFC policies and complements the current research efforts in this direction.

5 Conclusions and Future Work

RWFM presented in this paper is a unified model for information flow control, where information flow labels can be automatically inferred from an underlying discretionary policy. Application of RWFM to operating systems security is described which accepts a discretionary policy as specification and enforces the implied information flow. Thus, RWFM also forms a unified basis for specifying combination of DAC and MAC (information flow) policies.

An implementation of the proposed policy for operating systems security and its evaluation (performance and usability) remains to be carried out. The structure imposed by information flow policies on the entities of a system needs to be explored in detail. A basic framework for studying (label algebra) and comparing (embedding) the expressive power of information flow labels is developed in [25]. We wish to study the algebraic properties of readers-writers labels in this framework. In Section 4.3, we compared our labels with label models DLM and DC using an example. A more thorough comparison with DLM, DC and other well studied label models will provide insights into which model is suitable for what applications etc.

In the context of operating systems security, we wish to model our approach in a process calculus setting and formally establish its correctness even in the case of concurrent processes, similar to the work in [17, 14]. We opine that application of the techniques developed in this paper yields significant advantages for securing distributed systems, and in particular the cloud computing environment [44, 43]. In the distributed systems context, sound mechanisms for generating immutable labels that yield succinct representation while allowing efficient computations becomes a challenge. We believe that exploration of the suitability of SPKI names [9] for this purpose would provide tangible solutions.

References

[1] O. Arden, M. D. George, J. Liu, K. Vikram, A. Askarov, and A. C. Myers. Sharing mobile code securely with information flow control. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 191–205, Washington, DC, USA, 2012. IEEE Computer Society.

[2] A. Askarov and A. Myers. A semantic framework for declassification and endorsement. In A. D. Gordon, editor, *ESOP*, volume 6012 of *Lecture Notes in Computer Science*, pages 64–84. Springer, 2010.

[3] D. Bell and L. La Padula. Secure computer systems: Unified exposition and multics interpretation. In *Technical Report ESD-TR-75-306, MTR-2997, MITRE, Bedford, Mass*, 1975.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system version 2, 1999.

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE SP '96*, page 164, Washington, DC, USA, 1996. IEEE Computer Society.

[6] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust management system. In *FC '98: Proceedings of the Second International Conference on Financial Cryptography*, pages 254–274, 1998.

[7] D. F. Brewer and M. J. Nash. The chinese wall security policy. *IEEE Symposium on Security and Privacy*, 0:206, 1989.

[8] D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.

[9] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. RFC 2693: SPKI certificate theory. IETF RFC Publication, September 1999.

[10] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[11] T. Fraser. Lomac: Low water-mark integrity protection for cots environments. In *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, pages 230–245, 2000.

[12] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[13] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *IEEE Symposium on Security and Privacy*, pages 75–87, 1984.

[14] W. R. Harris, N. A. Kidd, S. Chaki, S. Jha, and T. Reps. Verifying information flow control over unbounded processes. In *Proceedings of the 2Nd World Congress on Formal Methods*, FM '09, pages 773–789, Berlin, Heidelberg, 2009. Springer-Verlag.

[15] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, Aug. 1976.

[16] K.Biba. Integrity considerations for secure computer systems. In *Technical Report ESD-TR-76-372, MITRE, Bedford, Mass*, 1976.

[17] M. Krohn and E. Tromer. Noninterference for a practical difc-based operating system. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 61–76, May 2009.

[18] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard os abstractions. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 321–334, New York, NY, USA, 2007. ACM.

[19] B. Lampson. Making untrusted code useful: Technical perspective. *Commun. ACM*, 54(11):92–92, Nov. 2011.

[20] N. Li, Z. Mao, and H. Chen. Usable mandatory integrity protection for operating systems. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 164–178, May 2007.

[21] J. Liu, M. D. George, K. Vikram, X. Qi, L. Waye, and A. C. Myers. Fabric: A platform for secure distributed computation and storage. In J. N. Matthews and T. E. Anderson, editors, *SOSP*, pages 321–334. ACM, 2009.

[22] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In C. Cole, editor, *USENIX Annual Technical Conference, FREENIX Track*, pages 29–42. USENIX, 2001.

[23] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell. The inevitability of failure: The flawed assumption of security in modern computing environments. In *In Proceedings of the 21st National Information Systems Security Conference*, pages 303–314, 1998.

[24] Z. Mao, N. Li, H. Chen, and X. Jiang. Combining discretionary policy with mandatory information flow in operating systems. *ACM Trans. Inf. Syst. Secur.*, 14(3):24:1–24:27, Nov. 2011.

[25] B. Montagu, B. C. Pierce, and R. Pollack. A theory of information-flow labels. In *Proceedings of the 2013 IEEE 26th Computer Security Foundations Symposium*, CSF '13, pages 3–17, Washington, DC, USA, 2013. IEEE Computer Society.

[26] A. C. Myers. Jflow: Practical mostly-static information flow control. In *POPL*, pages 228–241, 1999.

[27] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *SOSP '97*, pages 129–142, New York, NY, USA, 1997. ACM.

[28] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol.*, 9(4):410–442, Oct. 2000.

[29] A. C. Myers, N. Nystrom, L. Zheng, and S. Zdancewic. Jif: Java information flow. In *Software release. http://www.cs.cornell.edu/jif*, July 1992.

[30] P. Orbaek and J. Palsberg. Trust in the λ-calculus. *J. Funct. Program.*, 7:557–591, November 1997.

[31] N. Provos. Improving host security with system call policies. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 18–18, Berkeley, CA, USA, 2003. USENIX Association.

[32] I. Roy, D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel. Laminar: Practical fine-grained decentralized information flow control. *SIGPLAN Not.*, 44(6):63–74, 2009.

[33] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[34] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[35] R. Sandhu. The typed access matrix model. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 122–136, May 1992.

[36] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29:38–47, 1996.

[37] *SELinux*. http://selinuxproject.org/page/Main_Page.

[38] V. Simonet and I. Rocquencourt. Flow caml in a nutshell. In *Proceedings of the first APPSEM-II workshop*, pages 152–165, 2003.

[39] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The flask security architecture: System support for diverse security policies. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 11–11, Berkeley, CA, USA, 1999. USENIX Association.

[40] D. Stefan, A. Russo, D. Mazières, and J. C. Mitchell. Disjunction category labels. In *Proceedings of the 16th Nordic Conference on Information Security Technology for Applications*, NordSec'11, pages 223–239, Berlin, Heidelberg, 2012. Springer-Verlag.

[41] D. Stefan, A. Russo, J. C. Mitchell, and D. Mazières. Flexible dynamic information flow control in haskell. In K. Claessen, editor, *Haskell*, pages 95–106. ACM, 2011.

[42] S. Vandebogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris, and D. Mazières. Labels and event processes in the asbestos operating system. *ACM Trans. Comput. Syst.*, 25, December 2007.

[43] C. Wang, K. Ren, W. Lou, and J. Li. Toward publicly auditable secure cloud data storage services. *Network, IEEE*, 24(4):19–24, July 2010.

[44] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou. Toward secure and dependable storage services in cloud computing. *Services Computing, IEEE Transactions on*, 5(2):220–232, April 2012.

[45] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux security modules: General security support for the linux kernel. In *Proceedings of the 11th USENIX Security Symposium*, pages 17–31, Berkeley, CA, USA, 2002. USENIX Association.

[46] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers. Secure program partitioning. *ACM Trans. Comput. Syst.*, 20(3):283–328, 2002.

[47] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in histar. In *Proceedings of the 7th USENIX OSDI*, pages 19–19, 2006.

[48] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in histar. *Commun. ACM*, 54(11):93–101, 2011.

[49] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières. Securing distributed systems with information flow control. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 293–308, Berkeley, CA, USA, 2008. USENIX Association.