An Enhanced IFC Label Model to meet Application Policy Requirements.

Thomas F. J.-M. Pasquier[1], Olivier Hermant[2], Jean Bacon[1]

[1] University of Cambridge, United Kingdom, firstname.lastname@cl.cam.ac.uk
[2] MINES ParisTech, France, hermant@cri.ensmp.fr

Abstract. In recent projects we have investigated the use of Information Flow Control (IFC) for distributed and cloud computing. As reported elsewhere, we designed and implemented an Operating System (OS) kernel-loadable module for Linux (FlowK) to enforce IFC, and enhanced our SBUS middleware to be IFC-compliant (SBUS-IFC). FlowK's label model follows established practice for IFC in languages and OS, but unlike others and with cloud deployment in mind, FlowK's implementation aims to minimise the reengineering necessary in order to use IFC. We present the original FlowK model, then discuss difficulties we found in expressing and enforcing policies for certain applications, making the use of IFC impracticable for them. We then present a novel, enhanced label model (FlowK2) which subsumes that of FlowK. FlowK2 enables more concise and accurate expression of policy, according to the Principle of Least Privilege (PoLP). We have implemented FlowK2 and have carried out a comparative evaluation, showing similar overhead to current IFC implementations.

Keywords: Information Flow Control, Authorisation Policy

1 Introduction

Our recent projects have investigated the use of Information Flow Control (IFC) for distributed and cloud computing; for an overview see [1]. Concern about data leakage is holding back more widespread adoption of cloud computing by companies and public institutions alike. There is an increasing volume of legislation [15], but ensuring and demonstrating compliance with the legislation by cloud service providers and third parties is problematic. We believe that the deployment of IFC to augment traditional authentication and authorisation will make a substantial contribution to the security of distributed and cloud systems, both through enforcement mechanisms and demonstration of compliance through audit.

We designed and implemented an Operating System (OS) loadable kernel module FlowK [17] for Linux to enforce IFC. Note that we use the term IFC throughout to subsume Decentralised/Distributed IFC (DIFC) (see §2). FlowK's label model follows established practice for languages and OS (see §2), but unlike other implementations, and with cloud deployment in mind, we aimed to

minimise the reengineering necessary in order to use IFC. The design of FlowK follows the principle of policy-mechanism separation; the FlowK kernel module is concerned only with enforcing rules relating to entities' labels. Also, the Principle of Least Privilege (PoLP) should underly both application-level policy and IFC, as discussed in [2] and further in this paper.

Traditionally, access control is applied before data can be accessed by an entity, after which no further control is exercised on where that data flows in the system. IFC augments access control so that data flows are monitored continuously to enforce policy. This is achieved by associating labels with data and the entities that process them. Labels comprise a number of tags that describe the nature and/or source of the data such as healthcare, personal, government-information etc. Flows are permitted only if the labels match, as defined in §3. The challenge in integrating access control policy with IFC is to create labels that express policy accurately, concisely, naturally and efficiently, and in a way that makes policy changes easy to implement.

We have previously investigated how labels can be used to enforce certain laws and regulations, such as "data originating in the EU must not leave its boundaries, except to certain safe havens" [15]. A simple tag *EU* can be used for this purpose. We found access control policies to have more richness and complexity, and difficulties arise in representing some policies via IFC labels. Essentially, problems arise when some software can access all data items of a certain kind, for example to generate statistics, anonymise or encrypt, and other software is restricted to access only one such item. In work on Role-Based Access Control (RBAC), by ourselves [3] and others, parametrised roles were used in these circumstances to avoid proliferation of the number of required roles when following the PoLP.

In the work described here we have evaluated the original FlowK label model, which has equivalent expressiveness to other IFC models, from the perspective of carrying through application-level policy to runtime. We believe that some modifications are needed to improve the expressiveness of tags and to express correctly the trust that should be placed in entities that need to access a wide range of data, such as when generating aggregates of particular kinds of data. We propose two-component tags to represent the *concern* of data and a *specifier* for an item of that kind, for example $\langle medical, bob\rangle$ for Bob's medical record, and have implemented this in FlowK2. Such labels more closely reflect the policy maker's intent than previous models, and current single-component tags can easily be expressed in this way, such as $\langle location, EU\rangle$.

The first original contribution of this paper is the modification of standard IFC tag to allow better expression of policy §5. A second contribution of this paper is the ability to express conflict of interest for the first time in IFC implemented at the OS level §3.4 and §5.3.

§2 outlines IFC models and implementations and application level policy expression. §3 presents our original FlowK model. §4 gives some motivating examples underlying the design of FlowK2, showing how two-component tags solve certain problems. §5 presents the FlowK2 label model, extending that of

FlowK. In §6 we show that the modifications can easily be made and do not affect performance noticeably. §7 concludes the paper.

2 Related Work

This work is concerned with implementing authorisation policy in IFC via an OS-level mechanism. We first give an overview of IFC models. Since FlowK is an importable Linux kernel module, we then outline implementations of IFC within OS as components of distributed systems. We are not aware of any implementations of IFC in cloud computing environments.

We have found simple policies, for example on the permitted location of data, straightforward to express and enforce using the ubiquitous atomic tag model. We therefore focus on more complex and rich authorisation policy, in particular Role Based Access Control (RBAC), in which the most detailed application-level policy has been expressed.

2.1 IFC Models

It has long been argued that standard security techniques, such as firewalls and access control mechanisms, are not enough to prevent information leakage [8]. Indeed, it is beyond the scope of such mechanisms to determine whether, after the controls they impose, the information is used correctly. For example it is difficult to determine if the confidentiality of decrypted data is respected [19]. We therefore need to protect information flow end-to-end, in particular when information is transmitted within and between applications.

In 1976, Denning [8] proposed a Mandatory Access Control (MAC) model to track and enforce rules on information flow in computer systems. In this model, entities are associated with security classes. The flow of information from an entity a to an entity b is allowed only if the security class of b (denoted $\underline{b}$) is equal to or higher than $\underline{a}$. This allows the *no-read up, no-write down* principle of Bell and LaPadula [4] to be implemented to enforce secrecy. By this means a traditional military classification *public, secret, top secret* can be implemented. A second security class can be associated with each entity to track and enforce integrity (quality of data) during the permitted *reading down* and *writing up*, as proposed by Biba [5]. A current example might distinguish information from a government website in the *.gov.uk* domain from that from “Joe’s Blog”. Using this model we are able to control and monitor information flow to ensure data secrecy and integrity.

In 1997 Myers [16] introduced a decentralised IFC model (DIFC) that has inspired most later work. This model was designed to meet the changing needs of systems from global, static, hierarchical security levels to a more fluid system, able to capture the needs of different applications. In this model each entity is associated with two labels: a *secrecy* label and an *integrity* label, to capture respectively the privacy/confidentiality of the data and the reliability of a source of data. Each label comprises a set of tags, each of which represents some security

concern. Data are allowed to flow if the security label of the sender is a subset of the label of the receiver, and conversely for integrity. We describe in §3 the model we use in FlowK that derives from this general idea.

2.2 IFC in Operating Systems and Distributed Systems

Other research projects have implemented IFC constraints at the OS level, most notably in Flume [13]. Here, a model similar to Myers' original one [16] is used, and the entities considered are files, pipes, sockets and processes. In Flume, monitored (labelled) processes have access to a restricted set of system calls, and some (such as fork and pipe) are completely replaced by IFC-specific ones. This means that Flume applications running under IFC constraints need to be rewritten, even when they do not need to manipulate IFC labels during their life-cycles.

The Asbestos OS [10] implements the *send* and *receive* label paradigm, while also proposing a solution to improve the performance of IFC, which relies heavily on the fork operation to build applications. Asbestos is a rewritten OS rather than an imported module, requiring substantial changes in software that uses it. HiStar [25] extended Asbestos with security enhancements and a user-space library to emulate a Unix-like OS interface.

In Aeolus [7] and Laminar [18], an IFC aware operating system is used to enforce inter-process IFC constraints and a modified Java Virtual Machine ensures intra-process isolation via programming language objects. Again, it is necessary for application developers to be IFC-aware and to rewrite their Java programs.

DStar [26] enables IFC in distributed systems by leveraging the labelling mechanisms of IFC-compliant OS such as Flume and HiStar. DStar defines globally meaningful labels and each OS provides an *exporter* that maintains the correspondence between local representations and global labels, prroviding inter-process communication as message-passing. We have made a general middleware IFC-compliant (SBUS-IFC) [24], and are in the process of integrating it with FlowK2.

2.3 Role Based Access Control

RBAC is a mature, widely used access control scheme with a large literature and existing standards [11, 21].[3] Role definitions in RBAC tend to be functional in their scope, being application- or organisation-specific. Administrative roles are also included to capture the need to manage RBAC itself.

In work on RBAC by ourselves [3] and others [12, 14], parametrised roles were found to provide elegant expression of policy and avoid role explosion. For example, certain company software such as "Payroll" may need to access all employees' data whereas each employee can access only their own data record. To achieve this, a company either creates a role per employee e.g. *employee_smith* or parametrises a single role, for example *employee(smith)* etc. The Payroll software

[3] `http://csrc.nist.gov/groups/SNS/rbac/`

can then access *employee(*)*, where * indicates all employees. In this paper we argue that the IFC label model needs similar refinement in order to carry forward to runtime such aspects of application policy, thus following the PoLP.

Role parametrisation allows for relationships between parameters and for exclusions to be checked. For example the role *treating_doctor(doctorID, patientID)* allows the role to express the dynamic set of patients for each doctor for controlling access to records. An exclusion is expressed as a rule on a parameter value to indicate, for example, "all doctors can access my records except *doctorID*", see [3]. In §4 we show that such checks are most appropriately carried out at the authorisation point within the application, as are environmental checks such as "A Pharmacist can only authorise the issuing of drugs while on duty in the Pharmacy".

3 The FlowK Model

IFC augments authorisation by enforcing dynamically, end-to-end, that only permitted flows of information can take place. In this section we present our model for FlowK; please see [17] for further details and examples.

3.1 Enforcing Safe Flows via Labels

A tag within a label's set of tags represents a particular security concern for a category of data. In our IFC model two labels are associated with every entity A: a *secrecy label* $S(A)$ and an *integrity label* $I(A)$. The current state of these two labels (sets of tags) is the *security context* of an entity.

We now look at a flow of information from an entity A to an entity B, denoted $A \to B$. A flow is allowed if the following rules are respected:

$$A \to B, \text{ iff} \begin{cases} S(A) \precsim S(B) \\ I(B) \precsim I(A) \end{cases} \tag{1}$$

where, in FlowK, the preorder $\precsim$ denotes mere inclusion $\subseteq$ (it will be refined for FlowK2).

Consider the *read* and *write* functions of the Bell-LaPadula model [4] and the Biba model [5]. In the IFC world *read* is the equivalent of an incoming flow and *write* is the equivalent of an outgoing flow. In rule (1), the subrule concerning secrecy labels ensures that an entity only passes information to an entity that is allowed to receive it, thus enforcing the "no read up, no write down" policy of the Bell-LaPadula model. The subrule concerning integrity labels enforces quality of data during the permitted reading down and writing up, as proposed by Biba [5]. It is therefore possible to represent traditional security requirements as IFC constraints, although we use labels to represent more general security contexts.

When the information needs to flow in both directions the combination of $\precsim$ and $\succsim$ becomes an equivalence condition $\sim$, that for FlowK is equality:

$$A \leftrightarrow B, \text{ iff} \begin{cases} S(A) \sim S(B) \\ I(A) \sim I(B) \end{cases} \tag{2}$$

3.2 Creation of an Entity

We define $A \Rightarrow B$ as the operation of the entity A creating the entity B. An example is creating a process in a Unix-style OS by fork. We have the following rules for creation:

$$\text{if } A \Rightarrow B, \text{ then} \begin{cases} S(B) := S(A) \\ I(B) := I(A) \end{cases} \tag{3}$$

That is, the created entity inherits the labels of its creator.

3.3 Privileges for Managing Tags and Labels

Certain active entities (processes) have privileges that allow them to modify their labels. This is needed to support application management, see §3.5, and for declassification as described below. An entity has two sets of privileges for removing tags from its secrecy and integity labels (P_S^- for S and P_I^- for I), and two sets for adding tags to these labels (P_S^+ for S and P_I^+ for I). That is, for an entity A to remove the tag $t_s \in S(A)$, it is necessary that $t_s \in P_S^-(A)$, similarly to add the tag t_i to the label $I(A)$ it is necessary that $t_i \in P_I^+(A)$.

For an entity A, a label $X(A)$ and a tag t, a change of the label is authorised if the following rule is respected:

$$X(A) := X(A) \cup \{t\} \text{ if } t \in P_X^+(A) \text{ OR } X(A) := X(A) \setminus \{t\} \text{ if } t \in P_X^-(A) \tag{4}$$

For example, in order to receive information from an entity B, an entity A will need to set its labels (if it has the privilege) such that the flow constraints expressed by the tags associated with B are respected, that is such that the flow $B \rightarrow A$ respects the subrules in rule (1). We propose the following notation: for a process and its labels $(A, S, I) \rightsquigarrow (A, S', I')$ is the modification of the process labels following rule 4.

Declassification. An important example of changing security context is *declassification*. For example, plain-text data may have a *secret* tag whereas the same data when encrypted may flow more freely. A process that encrypts data must be trusted to have the privilege to *declassify* the derived encrypted data that is, to create encrypted data without the *secret* tag in its S label. In outline: the encrypting process starts off with the *secret* tag in S, reads the file with tag *secret* in its S, encrypts the data, changes its security context by removing the *secret* tag from S (for which it has the privilege) then writes the encrypted data, for example as public data.

Creation and Privileges. On creation, labels are automatically inherited by a created entity from its creator (per rule 3), but privileges are not. If the child is to be given privileges over its labels, they must be passed explicitly. We denote the flow generated by an entity A giving selected privileges $t^{\pm}_{X}$ to an entity B as $A \overset{t^{\pm}_{X}}{\hookrightarrow} B$ (for example allowing t to be removed from S, would be denoted $A \overset{t^{-}_{S}}{\hookrightarrow} B$). In order for a process to delegate a privilege to another process it must own this privilege itself. That is,

$$A \overset{t^{\pm}_{X}}{\hookrightarrow} B \text{ only if } t \in P^{\pm}_{X}(A) \tag{5}$$

3.4 Separation of Duty and Conflict-of-Interest Groups

A policy maker may need to specify a separation of duty (SoD) or conflict-of-interest (CoI) between principals and/or roles [6, 20]. An example of SoD is that an auditor may not audit their own actions. A CoI may arise when a principal could give professional advice to a number of competing companies. Separation of data access may be enforced by a Chinese Wall policy [6].

We believe CoI support in IFC is unique to FlowK. We define a set C of tags that represents some specified conflicting interests. In order for the configuration of an entity A to be valid with respect to C, rule (6) must be respected:

$$\forall C, \left| \Big(S(A) \cup I(A) \cup P^{+}_{S}(A) \cup P^{+}_{I}(A) \cup P^{-}_{S}(A) \cup P^{-}_{I}(A) \Big) \cap C \right| \leq 1 \tag{6}$$

That is, an entity is non-conflicting in this context if the set of its potential tags (past, present and future) contains at most one element from the set of tags within the related CoI group. In detail, by potential tags we mean the tags in its current S and I labels and those tags that it has the privilege to add to $S(A)$ (i.e. $P^{+}_{S}(A)$) and to $I(A)$ (i.e. $P^{+}_{I}(A)$) or that it may have removed from $S(A)$ (i.e. $P^{-}_{S}(A)$) and from $I(A)$ (i.e. $P^{-}_{I}(A)$). CoI rules should be checked every time a privilege is granted.

Suppose a conflict $C = \{Fiat, Ford, Audi, ...\}$ and some data is labelled $FiatData[S = \{Fiat\}, I = \emptyset]$ and $FordData[S = \{Ford\}, I = \emptyset]$. The CoI described ensures that it is not possible for a single entity (e.g. a process) to have access to both $FordData$ and $FiatData$ either simultaneously or sequentially, i.e. enforcing that $FordData$ and $FiatData$ are processed separately.

3.5 Application Startup

So far we have seen how active entities exchange information, create new entities, change their security context and delegate their privileges. We have not yet seen how tags and privileges are set up for the application instances.

We define a trusted entity to manage an application, denoted A^* for a trusted entity A. Such an A^* has, at startup, access to all the tags required for a given application and all the privileges to manage them. It spawns all application

instances, with appropriate security contexts, and sets up any CoI groups needed by the application. Note that A^* is the only entity entitled to create CoI groups for the tags it manages.

In our current implementation such trusted processes are configured at startup with the tags for the application's naming domain. Work is in progress on integrating FlowK with an IFC-aware middleware SBUS-IFC [22, 23], a function of which is to transfer tags as well as data.

In the example in 3.4, the application manager $CarManager^*$ is set up with the privileges to add and subtract all the tags in the naming domains for S and I labels for the application. For simplicity, consider $I = \emptyset$ for this application. Suppose the naming domain for S comprises the tags $\{Fiat, Ford, Audi, ...\}$. $CarManager^*$ creates a conflict of interest $C = \{Fiat, Ford, Audi, ...\}$ and then creates the processes to manipulate information for each car type.

4 Motivating Use Cases

In this section we motivate the use of two-component tags in FlowK2's label model. Our model is designed for a distributed system or cloud platform where there is likely to be a large amount of user data stored with persistent labels in files, databases, key-value stores etc. In a company context, data records may relate to individual employees; in a public health context, data may be the medical records of patients; in an educational context, data may relate to students, staff etc. Specifying and enforcing access to all, some specified subgroup or only one data record of a given type is a universal requirement, discussed in the literature on policy.

Suppose a principal is allowed access to a subset of records, e.g. doctors may be able to access only the records of the patients they are currently treating. Temporally separated instantiations of the application are likely, i.e. to one patient's records at a time. Each time, current authorisation policy is enforced and is translated into labels to ensure correct behaviour at runtime. Note that as a doctor's group of patients under treatment changes, a lookup of current patients at the authorisation point in the application will ensure that labels are created only for current patients, selected at runtime from the entire database.

The first two use cases arise from the need of certain software to perform computations on all records of a given type, whereas other software is authorised only to access records on behalf of a single individual. The third considers a system log containing records from all running applications relating to large numbers of principals. These problems are akin to the role proliferation that motivated role parametrisation in RBAC, as mentioned in §2.

We have solved these problems by making our tags more expressive to reflect the intended policy more accurately. Also, this new tag structure correctly represents the trust that should be placed in certain entities, according to PoLP. We use two-component tags, where a tag is of the form $\langle concern, specifier\rangle$. We use $\langle concern, *\rangle$ to indicate all tags of the specified concern and $\langle *, specifier\rangle$ for

tags of all concerns with the given specifier. $\langle *, * \rangle$ then indicates all tags in some naming domain. In §5 we present this label model for FlowK2 more formally.

4.1 Data Analysis

In the healthcare domain, statistical analysis of the medical records of patients is needed for various purposes including public health, environmental concerns, clinical practice etc. We are concerned with the privacy and confidentiality of medical data and will therefore discuss the construction of the IFC secrecy label for a statistical analysis program.

In the current FlowK model a tag represents a single security concern. To express the idea of Bob's medical data, we would use two tags *bob_data* and *medical_data*. The entity carrying out the statistical analysis of the medical data would then need to have not only the *medical_data* tag, but also a tag corresponding to every patient's data, for rule (1) to be satisfied. This makes the use of IFC infeasible for such purposes: (1) The performance implications are significant. We have shown [17] that the processing overhead induced by the number of tags in labels is not insignificant (especially at this scale). (2) Enumerating all tags would be prone to error as the database state changes, with records being added and removed. (3) This entity would be over-privileged by the PoLP, being able to receive any data labelled only with the tag *bob_data* although our intention was for it only to be concerned with *medical_data*. It would also be privileged to declassify, see §3.3, over all the patients' personal tags and trusted not to leak information about any patient. An alternative would be to define different tags for every category of user data such as *bob_medical_data*, *bob_tax_data*, *bob_home_data* and many more. The performance issues caused by huge numbers of tags would still hold. Also, an entity running on behalf of Bob could not simply have a label with a single tag *bob_data* but would have to enumerate all the particular types of Bob's data it could process.

The problems are solved by using two-component tags, where a tag is of the form $\langle concern, specifier \rangle$. Each record has an S label containing a tag such as $\langle medical, patientID \rangle$ and we express that an entity can access all medical data by including the tag $\langle medical, * \rangle$ in its S label, where $*$ indicates all records of concern *medical*. An entity E doing statistical analysis could be labelled $E = [S : \{\langle medical, * \rangle, \langle medical, statistics \rangle\}, I : \{\}]$ and would need to declassify over all patients to create useable output labelled $[S : \{\langle medical, statistics \rangle\}, I : \{\}]$. We show in §5 how declassification is represented in FlowK2's label model.

4.2 Log Audit

Consider a system log which several active entities are able to access. A process concerned with digital forensics requires access to the whole log to detect and investigate suspicious patterns of behaviour; applications, e.g. cloud tenants, and individual users should be able to audit the use of their own data.

With the current model an entity performing audit over all log records would need access to all relevant concerns and specifiers: in FlowK2's model $\langle *, * \rangle$. It

would need declassification privileges over all those data for the audit result to be of any use, see §5. An entity performing audit over data logs for specified applications would need $\langle application_name, *\rangle$ and again, the privilege to declassify over all specifiers. The tag needed by the entity performing audit on behalf of an individual over all their data (from any application) would be $\langle *, person_name\rangle$. The entity would need the privilege to declassify the output, see §5.

In this section we have given the intuition and motivating examples for the two-component label model of FlowK2. In the next section we present the formal notation, extending the FlowK model.

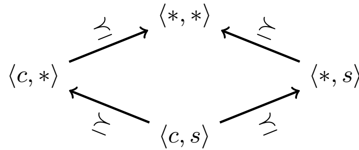
5 Two-Component Tags in FlowK2

After studying existing IFC label models [2] we designed and implemented FlowK. We have previously worked on authorisation policy specification and implementation [3] and the next phase of our current project was to integrate application-level policy with IFC. We had explored this in principle in [1] but the details of translating policy into labels for authorised principals have made us reconsider existing label models.

As motivated in §4 a major concern was to make our label model more expressive in order to reflect the intended policy accurately, while keeping its general simplicity (see §3). An important pragmatic concern is to limit the number of tags in a label, since this has a non-negligible effect on performance [17].

We propose to decompose a tag t into a pair $\langle c, s\rangle$ with c the concern of type $\mathscr{C}$ and s a specifier of type $\mathscr{S}$. For example, the pair $\langle medical, bob\rangle$ represent Bob's medical data. A statistical analysis over a set of patients' medical data is represented as $\langle medical, statistical_analysis\rangle$ and anonymised medical records as $\langle medical, anonymised\rangle$.

As we saw in §4, a major requirement is to be able to specify *all* data records of a certain kind without enumerating all possible tags, as required by current models. Therefore for any concern c and specifier s we establish the following subtyping relation:



That is, a tag $t = \langle c, s\rangle$ is a subtype of $t' = \langle c, *\rangle$ and $t'' = \langle *, s\rangle$ which are themselves subtypes of $t''' = \langle *, *\rangle$. For instance, $\langle medical, bob\rangle$ (Bob's medical data) is a subtype of $\langle medical, *\rangle$ (medical data) and a subtype of $\langle *, bob\rangle$ (Bob's data) which are each subtypes of $\langle *, *\rangle$ (all data in the current naming context).

5.1 Changes to Flow Constraints

To adapt rule (1) from §3.1 for the flow $A \rightarrow B$, we need only redefine the $\precsim$ binary relation between sets of tags X and Y as follows:

$$X \precsim Y \text{ iff } \forall t \in X\ \exists t' \in Y : t \preceq t' \tag{7}$$

Together with rule (1), this entails that a flow $A \rightarrow B$ is allowed if and only if for all secrecy tags of A there exists a super type in the secrecy tags of B and that for all integrity tags of B there exists a super type in the integrity tags of A. For example, an entity A labelled with $S(A) = \{\langle medical, bob\rangle, \langle legislation, EU\rangle\}$ is able to send information to an entity B with $S(B) = \{\langle medical, *\rangle, \langle legislation, EU\rangle\}$. For the bi-directional flow $A \leftrightarrow B$ in rule (2), the tag sets X and Y must be equivalent, since:

$$X \sim Y \text{ iff } \begin{cases} \forall t \in X\ \exists t' \in Y : t \preceq t' \\ \forall t \in Y\ \exists t' \in X : t \preceq t' \end{cases} \tag{8}$$

5.2 Changes to Privileges

Rule (4) from §3 becomes:

$$\begin{gathered} X(A) := X(A) \cup \{t\} \text{ if } \exists t' \in P_X^+(A) : t \preceq t' \\ \text{OR} \\ X(A) := X(A) \setminus \{t\} \text{ if } \exists t' \in P_X^-(A) : t \preceq t' \end{gathered} \tag{9}$$

We also add special privileges noted $\langle c, \Delta\rangle$, $\langle \Delta, s\rangle$ and $\langle \Delta, \Delta\rangle$ that allow removal only of the labels $\langle c, *\rangle$, $\langle *, s\rangle$ and $\langle *, *\rangle$ respectively.

For example a process A, with the privilege $P_S^-(A) = \{\langle medical, \Delta\rangle\}$ and the label $S(A) = \{\langle medical, *\rangle, \langle medical, anonymised\rangle\}$ is able to declassify to $S(A) = \{\langle medical, anonymised\rangle\}$. The use of Δ privileges allows the trust placed in a certain entity to be precise and is particularly useful when specifying declassifier privileges. Without it, we would have had $P_S^-(A) = \{\langle medical, *\rangle\}$ and no guarantee that the process would not declassify to $S(A) = \emptyset$ (also removing $\langle medical, anonymised\rangle$).
Privilege delegation rule(5), becomes:

$$A \overset{t_X^{\pm}}{\hookrightarrow} B \text{ only if } \exists t' \in P_x^{\pm}(A) : t \preceq t'$$

5.3 Changes to Conflicts of Interest

There are now three types of policy we must express: constraints applied to whole tags, to concerns and to specifiers. We define three operations on a tag's

pair, the projections π_1 in $\mathscr{C}$, π_2 in $\mathscr{S}$ and the identity function id:

$$\begin{array}{cc} \pi_1 : \mathscr{C} \times \mathscr{S} \rightarrow \mathscr{C} & \pi_2 : \mathscr{C} \times \mathscr{S} \rightarrow \mathscr{S} \\ \pi_1(\langle c, s\rangle) = c & \pi_2(\langle c, s\rangle) = s \end{array} \tag{10}$$

We extend these operations to sets, such that:

$$\begin{array}{cc} \pi_1 : \wp(\mathscr{C} \times \mathscr{S}) \rightarrow \wp(\mathscr{C}) & \pi_2 : \wp(\mathscr{C} \times \mathscr{S}) \rightarrow \wp(\mathscr{S}) \\ \pi_1(T) = \{\pi_1(t) \mid t \in T\} & \pi_2(T) = \{\pi_2(t) \mid t \in T\} \\ = \{c \mid \langle c, s\rangle \in T\} & = \{s \mid \langle c, s\rangle \in T\} \end{array} \tag{11}$$

For an entity A we note the union of its labels and privileges:

$$SU(A) = S(A) \cup I(A) \cup P_S^+(A) \cup P_S^-(A) \cup P_I^+(A) \cup P_I^-(A) \tag{12}$$

A conflict of interest is denoted $P_{CoI}(f, G)$ where f is π_1, π_2 or id and G is a set of conflicting tags. Rule 6 in §3.4 becomes (where C is a CoI group):

$$\forall P_{CoI}(f, C), |f(SU(A)) \cap C| \leq 1 \tag{13}$$

Here we note:

- $\{a, b, c\} \cap \{*\} = \{a, b, c\}$;
- $\{\langle a, b\rangle, \langle a, d\rangle, \langle c, d\rangle\} \cap \{\langle a, *\rangle\} = \{\langle a, b\rangle, \langle a, d\rangle\}$;
- $|\{*\}| = \infty$, $|\{a, *\}| = \infty$ and $|\{*, a\}| = \infty$.

The conflict of interest rule: $P_{CoI}(\pi_1, \{medical, private\})$ means an entity can handle the concern medical or private but not both. $P_{CoI}(id, \{\langle private, *\rangle\})$ means an entity can only ever manipulate the private data of a single user.

Consider the example used in 3.4 on isolating processes to manipulate car company data. A problem with the previous model is that if a new company was to use the application, a new tag would need to be added to the CoI group. For a rule applying to a more rapidly changing set this could prove problematic.

Using FlowK2, we have processes labeled $[S = \{\langle car, ford\rangle\}, I = \emptyset]$, $[S = \{\langle car, fiat\rangle\}, I = \emptyset]$ etc. and the CoI policy is expressed as: $P_{CoI}(id, \{\langle car, *\rangle\})$. This is simple to read and understand (i.e. a process can manipulate information for only one specifier of concern car) and this policy will not change over time.

5.4 Compatibility with Single-Component Tags

Certain tags in our original work create no problems as single-component tags. For example, $\langle user - input\rangle$ may be included in an Integrity label to indicate that input data should not be trusted. The tags $\langle EU - data\rangle$ and $\langle US - data\rangle$ have been used to enforce the geographical location of stored data to allow laws and regulations to be enforced. Such tags do not need to be two-component tags but can conveniently be expressed as such, e.g. $\langle input, network\rangle$ or $\langle location, EU\rangle$. Policy may or may not be conveniently expressed for such tags using $*$, but if $*$ is never used, our two-component tag model degrades gracefully to what is effectively a conventional one-component tag model, since both components must match for data to flow. Backwards compatibility with existing policies that have been defined for single-component tags is easily achieved by this means.

6 Implementation and Impact on Performance

We are building cloud and distributed systems around an IFC-aware operating system [17] and an IFC-aware middleware messaging system [24]. The approach has been explored for distributed systems in DStar [26] over IFC-aware OS [13, 25] (see §2), but with the need for reengineering software before using IFC.

In FlowK [17], we provide IFC to the OS through a kernel module that intercepts system calls [9] to enforce IFC constraints, while relying on the underlying system call. We assume that exchange of information between active entities (processes) occurs through passive entities (files, sockets and pipes) and that shared memory between processes is not allowed.

Our kernel module maintains a map between entity identifiers (processes, pipes etc.) and their respective labels and privileges. In system calls such as *write* or *send*, the security context of the process and of the entity associated with the provided *file descriptor* (i.e. socket, pipe or file) are retrieved, evaluated and compared to decide whether the call is authorised (and respectively for *read* and *recv*). System calls not generating information flow are left untouched.

Performance is first influenced by the cost of intercepting system calls. For FlowK we measured the interception cost on pipe read/write as around 10% per individual system call. The second factor is the cost of evaluating policy. For this example the additional overhead is negligible for up to 10 tags, rising to 25% with 100 tags. The number of tags used to express policy should therefore be minimised. The measured performance overhead for an application running on FlowK varies from 5% to 23% [17].

SBUS is a component-based messaging middleware. A component represents a process whose communication is managed by SBUS. Communication occurs through endpoints which are protected using IFC constraints. A component will have a number of endpoints, which can be connected (mapped) to endpoints of other components to enable communication [22,23]. SBUS supports client/server and stream-based interactions; both may be required for distributed systems. A *source* end-point can be mapped to multiple *sinks* to allow multicast.

The cost of system call interposition (as described in §6) is not affected by the policy being enforced so we focus on evaluating the cost of verifying IFC flow constraints. We compare the cost of evaluating the legality of a flow $A \to B$ between two entities A and B for FlowK (§3) and FlowK2 (§5). Further evaluation of FlowK can be found in [17]. For the messaging middleware layer, policy evaluation can be considered negligible in comparison with network traffic.

In FlowK, tags are represented by 64 bit integers and labels by ordered arrays. The worst case complexity of determining if a label is a subset of another is $\mathcal{O}(n)$. In FlowK2, a tag is implemented using two 64 bit integers, the worst case complexity for the 2D tag model is $\mathcal{O}(n^2)$.

In situations where the policy would be strictly identical as described in §5.4, FlowK would have a performance advantage over FlowK2 because of the increased complexity of tag comparison. However, as we showed in §4, the use of two-component tags allows us to simplify some policies, greatly reducing their complexity; situations that would have required a large number of tags

are avoided. We believe that in most cases the number of tags would be few and the performance similar to basic IFC, with improved performance in some cases.

7 Conclusion and Future Work

We have described part of our work to demonstrate the feasibility of providing IFC for cloud and distributed computing. We followed standard practice in designing our original label model and implemented it as a kernel-loadable module FlowK to enforce IFC. To increase the likelihood of adoption, FlowK requires no reengineering of the software that uses it.

This paper focusses on a novel enhancement of our label model (FlowK2) to meet application-level policy requirements. It is unlikely that IFC will be used in practice unless authorisation policy can be expressed correctly, elegantly and concisely, and carried through efficiently to runtime. We have shown that two-component tags solve significant problems in expressing policy and impose no more runtime overhead than existing IFC implementations.

Acknowledgement

This work was supported by the UK Engineering and Physical Sciences Research Council under grant EP/K011510 "CloudSafetyNet: End-to-End Application Security in the Cloud". We acknowledge the support of Microsoft for work on Cloud Law.

References

1. Jean Bacon, David Evans, David M. Eyers, Matteo Miglivacca, Peter Pietzuch, and Brian Shand. Big Ideas paper: Enforcing end-to-end application security in the cloud. In *ACM/IFIP/Usenix Middleware, LNCS 6452*, pages 293–312, 2010.
2. Jean Bacon, David Eyers, Thomas Pasquier, Jatinder Singh, Ioannis Papagiannis, and Peter Pietzuch. Information Flow Control for Secure Cloud Computing. *IEEE TNSM, Special Issue on Cloud Service Management*, 11(1):76–89, March 2014.
3. Jean Bacon, Ken Moody, and Walt Yao. A Model of OASIS Role-based Access Control and its Support for Active Security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, November 2002.
4. David E. Bell and Leonard J. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. Technical Report M74-244, The MITRE Corp., Bedford MA, May 1973.
5. K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report ESD-TR 76-372, MITRE Corp., 1977.
6. D. Brewer and M. Nash. The Chinese Wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, 1989.
7. Winnie Cheng, Dan R. K. Ports, David Schultz, Victoria Popic, Aaron Blankstein, James Cowling, Dorothy Curtis, Liuba Shrira, and Barbara Liskov. Abstractions for Usable Information Flow Control in Aeolus. In *Proc. USENIX Annual Technical Conference*, Boston, 2012.

8. Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.
9. Nitesh Dhanjani and Gustavo Rodriguez-Rivera. Kernel korner: Loadable kernel module programming and system call interception. *Linux Journal*, (82es), February 2001.
10. Petros Efstathopoulos and Eddie Kohler. Manageable fine-grained information flow. In *ACM EuroSys*, 2008.
11. David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST Standard for Role-based Access Control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, August 2001.
12. Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *Proceedings of the Second ACM Workshop on Role-based Access Control*, RBAC '97, pages 153–159, New York, NY, USA, 1997. ACM.
13. Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard OS abstractions. In *21st ACM SOSP*, pages 321–334, 2007.
14. Emil Lupu and Morris Sloman. Reconciling role based management and role based access control. In *Proceedings of the Second ACM Workshop on Role-based Access Control*, RBAC '97, pages 135–141, New York, NY, USA, 1997. ACM.
15. Christopher J Millard. *Cloud Computing Law.* OUP, 2013.
16. Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *Proc. 17th SOSP*, pages 129–142. ACM, 1997.
17. Thomas F. J.-M. Pasquier, Jean Bacon, and David Eyers. FlowK: Information Flow Control for the Cloud. In *under review.* IEEE, Submitted.
18. Indrajit Roy, Donald E. Porter, Michael D. Bond, Kathryn S. McKinley, and Emmett Witchel. Laminar: Practical fine-grained decentralized information flow control. *SIGPLAN Not.*, 44(6):63–74, June 2009.
19. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE JSAC*, 21(1):5–19, January 2003.
20. Ravi Sandhu. Separation of duties in computerized information systems. In *Database Security IV: Status and Prospects*, 1990.
21. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29(2):38–47, February 1996.
22. Jatinder Singh and Jean Bacon. On middleware for emerging health services. *Journal of Internet Services and Applications*, 5(6):1–19, May 2014.
23. Jatinder Singh and Jean Bacon. SBUS: A generic, policy-enforcing middleware for open pervasive systems. *University of Cambridge Computer Laboratory Technical Report TR*, 850, 2014.
24. Jatinder Singh, Jean Bacon, and David Eyers. Integrating Messaging Middleware with Information Flow Control. In *under review.* ACM.
25. Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in HiStar. In *Proc. 7th USENIX OSDI '06*, pages 19–19, 2006.
26. Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. Securing distributed systems with information flow control. In *Proc. 5th USENIX NSDI'08*, pages 293–308, 2008.