Information Flow Control for Strong Protection with Flexible Sharing in PaaS

Thomas F. J.-M. Pasquier, Jatinder Singh and Jean Bacon
Computer Laboratory, University of Cambridge
Cambridge, United Kingdom
Email: firstname.lastname@cl.cam.ac.uk

***Abstract*—The need to share data across applications is becoming increasingly evident. Current cloud isolation mechanisms focus solely on protection, such as containers that isolate at the OS-level, and virtual machines that isolate through the hypervisor. However, by focusing rigidly on protection, these approaches do not provide for *controlled sharing*. This paper presents how Information Flow Control (IFC) offers a flexible alternative. As a data-centric mechanism it enables strong isolation when required, while providing continuous, fine grained control of the data being shared. An IFC-enabled cloud platform would ensure that policies are enforced as data flows across all applications, without requiring any special sharing mechanisms.**

I. Introduction

Since the inception of PaaS, a major concern has been to isolate tenants' applications from each other to guarantee first, confidentiality of their data and, to a lesser extent, its integrity. Isolation was originally achieved by running tenants within separate *virtual machines (VMs)*, sharing hardware and the hypervisor. More recently, strong isolation has been achieved through *containers* [1], which isolate tenants over a shared OS. Access to resources, such as storage, is often restricted through tenant-based access control.

Applications, however, are increasingly required to share data. The emergence of the Internet of Things (IoT), and IoT's use of PaaS services [2] to achieve open interoperability, is making support for protected data sharing between PaaS tenants increasingly urgent. Meanwhile, research on data sharing between isolated containers [3], [4] has proposed complex and often relatively inefficient (application-specific access control) schemes.

We believe that what is needed is a generic mechanism to finely control the exchange of information between parties running applications on a PaaS cloud, that provides for both *protection* (isolation) and *sharing*. Sharing should be supported, by permitting data flows according to application policy. Strict isolation should be enforced when flows are not permitted. A mechanism is needed to enforce policy that defines which information to share and with whom.

As such, we propose *Information Flow Control (IFC)*, which enables **granular, data-centric isolation**. Current containment approaches focus on strict tenant and/or infrastructure isolation, with no means for controlling the subsequent use of data that has been shared. IFC provides mandatory enforcement and discretionary, application-specific policy specification. Furthermore, IFC guarantees continued enforcement of the specified policy after the data has been exchanged.

We begin by providing an overview of IFC, before describing our prototype implementation and its relevance to cloud platforms. We then compare IFC with containers—the contribution of this paper—to highlight the benefits of the approach.

II. IFC Model

The purpose of IFC is to prevent data leakage through controlled information exchange. Our model accords with the general IFC guarantees of secrecy (*no read up, no write down* [5]) and integrity (*no read down, no write up* [6]). Further details of our model, with examples, are given in [7].

A. Tags and Labels

In IFC, tags are tokens that represent some security concern regarding secrecy or integrity which are used to regulate flows throughout the system. Tags are associated with entities that include application instances (processes), messages, sockets or any mechanism allowing data exchange.

Every entity in the system has two labels, each comprising a set of tags: an entity A has a secrecy label $S(A)$ and an integrity label $I(A)$. The current state of these labels is the entity's *security context*.

Definition 1. *A flow of information $A \to B$ is safe if and only if:*

$$A \to B, \textit{ iff } S(A) \subseteq S(B) \land I(B) \subseteq I(A)$$

The secrecy label S enforces privacy/confidentiality and the integrity label I enforces data quality, using tags. For example, a hospital patient Bob may be issued with a heart monitor device to use when at home. Data from the device must flow to a process in his hospital for analysis and storage. The hospital process is only allowed to receive data from hospital-issued devices, so has $I =$[hospital-issued]. Bob's device and the data flowing from it must also include the tag $I =$[hospital-issued] for the data to be accepted. Because Bob's data is private, the heart monitor and data are tagged $S =$[medical, bob]. The hospital process may only receive the data if it has tags which include $S =$[medical, bob].

An example from Fig. 1b, is that Carl's application instance B can only communicate with Bob's application instance B' if $S(B) \subseteq S(B')$.

B. Decentralised Privileges and Security Contexts

Sometimes it may be necessary to make data more accessible (removing a secrecy tag), e.g. when a document becomes ready for publication. This is known as declassification. It is also the case that certain applications may require sanitised/verified input data. This means that a process will be charged with analysing new inputs and labelling data as sanitised (adding an integrity tag), to enable the data to propagate further. This is termed endorsement.

An application (and the principal on behalf of whom the application runs) requires privileges to undertake endorsement and declassification operations. If an application A has a privilege to add t to its secrecy label, we denote this $t \in P_S^+(A)$, and to remove t from its secrecy label: $t \in P_S^-(A)$ (and similarly $P_I^+(A)$ and $P_I^-(A)$ are the privileges for integrity).

Definition 2. *A label change noted $A \rightsquigarrow A'$ is safe if and only if all additions or removal of tags respect (where X stands for either S or I):*

$$X(A) := X(A) \cup \{t\} \text{ if } t \in P_X^+(A)$$

OR

$$X(A) := X(A) \setminus \{t\} \text{ if } t \in P_X^-(A)$$

IFC policy is therefore *simple to express and enforce*, expression involving assigning tag metadata to entities and enforcement checking the presence of tags in flow endpoints' metadata. Moreover *IFC policy is enforced continuously*, unlike traditional access control.

III. Cloud Provider Enforced IFC

We have developed a prototype IFC implementation to demonstrate the feasibility of an IFC-enabled cloud platform. For want of space, here we provide only a brief overview, for more detail see [7]. Data exchange across machines is done through an IFC-aware message passing middleware, described in [8].

A. Kernel Enforced IFC

FlowK operates to enforce IFC within an OS, and thus easily integrates into cloud services. It is implemented in the kernel to ensure consistent enforcement across all applications. We assume that passive entities (files, sockets and pipes) are used to exchange data between active entities (processes), and we prevent shared memory between processes.

Our implementation uses system call interception through a Linux Kernel Module [9]. Although this method has some limitations [10], [11], it is sufficient for a proof-of-concept exploration of IFC functionality. We are currently investigating a Linux Security Module IFC implementation to help address some of these concerns.

B. User Space Helpers

The development of IFC-aware applications, including services forming part of an IFC-aware PaaS, requires mechanisms for user-space processes to interact with the kernel-level IFC functionality. For this, we define User-Space Helper processes, which have three functions: 1) persisting an application's security context across multiple executions; 2) saving the correspondence between local and global tag representations; 3) assisting in inter-process message-passing.

C. A Prototype for PaaS with Decentralised IFC

We now describe application-level integration through a framework for web applications that run above an IFC-aware OS. This has been implemented and detailed in [7]. In this implementation, the workers are Ruby on Rails instances and we use a Redis data-store.

End-users are able to specify the IFC security context in which they want their request (and therefore the application) to be executed. This is managed through a web interface. An end-user is able to: define new tags to reflect his policy requirements; share tags with other end-users to allow data sharing; and dynamically define under which IFC security context the remote cloud application should be executed. For example, a security context can be set up such that a nurse can access the medical information pertaining to one of their patients.

The end-users attach[1] to each request credentials specifying the security context in which the request should be executed. When requests are received they are routed towards an application instance running within the specified security context. If no such instance exists, a new one is created to meet the user's security requirements. To improve performance, application instances (processes) no longer in use can be recycled (reused) through checkpoint techniques that ensure no data leaks between security contexts [12].

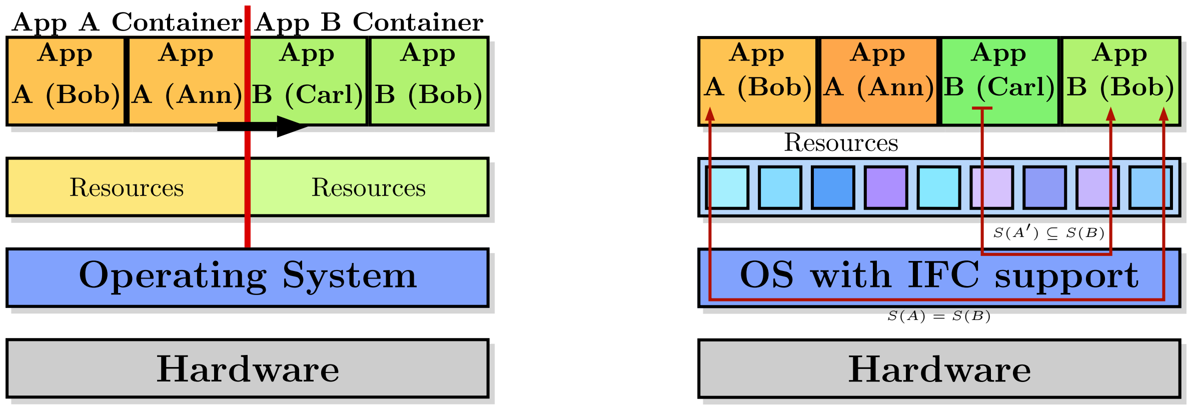
Application instances are constrained to work within a well defined security context as specified by the end-user request. This security context cannot be changed directly by the application and its enforcement is guaranteed by the OS as described in §II. The resources, data items (within a store) and messages an application instance is able to read/send or write/receive are (tagged) in accordance with the IFC constraints.

In terms of protection, application bugs or misconfigurations cannot compromise end-users' data. Furthermore, data produced by an application (e.g. a smart watch) could be used by another application (e.g. a diet coaching application) provided the security contexts of the applications are compatible.

IV. Containment compared

Virtual machines (VMs) are well established as a mechanism for tenant isolation, albeit somewhat heavyweight since each tenant is allocated a separate VM (executing its own OS, software-stack, etc.). More recently, containers have emerged as a more efficient (performance-wise) form of isolation [1], [13], in which tenants share the same OS.

[1]This is similar to a single sign-on system, i.e. the user authenticates with the security context provider and receives credentials corresponding to the desired security context.



(a) Application isolation via containers.

(b) Resource and data sharing with IFC constraints.

Fig. 1: Container Isolation vs IFC aware OS

Both of these approaches focus on protection, essentially imposing a 'silo' around/between tenants and/or resources. Although applications can contrive to share data across isolation boundaries, sharing is either unsupported by the platform, or tends to target situations where multiple applications belong to a single/small groups of tenants, specified statically. For example, Google Pods[2] allows applications to exchange information and share common resources, but focuses on enabling common logging, content management, backup services, etc., for the range of applications that a tenant runs.

There is an increasing need for *managed* data sharing across a range of tenants and application boundaries. Developments in 'Internet of Things' (IoT), for example, are towards breaking application/tenant silos to allow large-scale data sharing in order to build more sophisticated applications [2], including dynamically. There are also requirements emerging regarding the isolation of data and applications for individual end-users (c.f. isolating at the granularity of a tenant), for example, to offload power-constrained devices' computations [3], while allowing data to be shared when appropriate.

Such sharing must be controlled. The rationale of existing mechanisms is that once data access is authorised, there is no further control over how the data is subsequently used. The benefit of IFC is to control information propagation continuously, not only at the point of authorisation.

In Fig. 1, we illustrate the differences between an IFC and container managed cloud. For containers, application instances run as though they were alone in the machine, and all application instances within the same container share resources. IFC allows more fine grained non-interference policy, making it possible for inter-tenant data sharing (which is complicated when using containers) and intra-tenant isolation (which would require a large number of containers, and increase management complexity). In short, *IFC enables not only isolation, but also flexible data sharing.*

In Fig. 1b, we see a user, Bob, sharing data between two applications, and another user, Carl, who shares data with Bob. Using containers (Fig. 1a) any data exchange policy would be application-centric (i.e. App A can interact with App B). Further, once the data is shared there is a definitive loss of control: Bob has no guarantee that if he shares his data with application B, Carl cannot access it. As discussed earlier, it is possible for each user/application pair to run its own container, but this brings issues of scale. In contrast, with IFC, the data is protected by the cloud platform. Even when shared between applications, data can only flow to application instances with compatible security contexts (as defined by their S and I labels' tag sets). Carl cannot access Bob's data even if they both execute instances of application B, except by specific IFC-enforced policy. Although one application instance per security context appears heavyweight, it is far less so than the alternative of running a complete container per user. Furthermore, as mentioned, application instances can be re-used through checkpointing techniques in order to reduce overhead.

Another consideration is that container technology does not currently extend to the protection of data storage. IFC can be enforced within the OS, and also in cloud provided services such as data storage (protecting files) or caching. A technique explored is to provide an IFC-aware trusted interface between the data-store and the cloud application [14] or through an IFC-aware FUSE interface [7], leveraging the OS IFC enforcement mechanism discussed in §III. IFC has been integrated into the PostgreSQL database [15], and a messaging middleware SBUS-IFC [8], that also enables managed database queries.

Managed data sharing is only possible if the cloud provider is able, and trusted, to guarantee that the data-bound policies are enforced. As part of this enforcement, audit logs can be generated. A verifiable audit log provides transparency as to whether data is properly used and managed, i.e. according to the provider-tenant contract. Further, as these logs are naturally data-centric [16] they allow for more meaningful audit than traditional system logs, and are able to capture data-specific constraints; such as those imposed by legislation dictating

[2]http://tinyurl.com/ml5kux4

the geographic locales to which particular information may flow [17], [18]. Audit can also raise the level of trust end-users have in the behaviour of cloud applications. For example, end-users could verify that no data leakage occurred, or that their data are not being shared with undisclosed third parties.

V. Conclusions & Future Work

Cloud security has always been a major issue. PaaS security design traditionally focussed on protection, isolating tenants via VMs and later via OS co-resident containers. The need for data-sharing between tenants was not a major consideration, but has increasingly become a requirement. Even with traditional applications we can envisage, say, local government having many separate applications such as social services, public transport, electoral roll and many more, relating to the same citizens and needing a degree of interworking.

With the emergence of IoT, the need for sharing becomes compelling. The philosophy of IoT is open interoperability, with data sharing able to occur dynamically, possibly in unforeseen ways. Cloud services are seen as an essential part of an overall IoT architecture which may include traditional cloud applications as well as 'things'. Examples of 'things' include the many personal monitoring devices whose data need to be composed and correlated for emergency detection, but also integrated with traditional healthcare, including professional care services and health record databases.

The contribution of this paper is in the case for isolation approaches that also provide for managed data sharing. We believe the rigour and flexibility of IFC makes it a strong contender for meeting the future needs of PaaS clouds for both strong protection and flexible sharing as required.

So far, we have built a proof-of-concept system from a Linux kernel module (FlowK) to an IFC-enabled web-service application framework [7]. We have integrated an IFC-aware messaging middleware with FlowK [8].

There are remaining challenges in deploying IFC in PaaS clouds but we believe it is feasible and has great potential. Discussion on and progress towards meeting these challenges can be found in [19], [7], [8], [2]; in outline:

(1) IFC can be implemented at different levels in the cloud stack, giving rise to cross-cutting concerns. For example, policy concerning structured objects operates at a higher-level than the OS. As initial work towards this, we have structured messages with individually labelled attributes as part of middleware integration [8].

(2) For a general architecture including multi-clouds and end-systems, IFC can be extended to operate end-to-end, provided a global naming scheme is defined for tags.

(3) Uniquely among IFC models and implementations, we have aimed for maximum transparency of IFC. An application instance can use IFC without reengineering if it does not need to change its security context.

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] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, 2007, pp. 275–287.

[2] J. Singh, T. Pasquier, J. Bacon, , H. Ko, and D. Eyers, "20 Cloud Security Considerations for Supporting the Internet of Things," *submitted, IEEE Internet of Things Journal*, 2015.

[3] S. Lee, E. L. Wong, D. Goel, M. Dahlin, and V. Shmatikov, "πBox: A Platform for Privacy-Preserving Apps." in *10th USENIX Symposium on Networked System Design and Implementation*, 2013, pp. 501–514.

[4] B. Viswanath, E. Kiciman, and S. Saroiu, "Keeping information safe from social networking apps," in *Proceedings of the 2012 ACM workshop on Workshop on online social networks*. ACM, 2012, pp. 49–54.

[5] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," The MITRE Corp., Bedford MA, Tech. Rep. M74-244, May 1973.

[6] K. J. Biba, "Integrity Considerations for Secure Computer Systems," MITRE Corp., Tech. Rep. ESD-TR 76-372, 1977.

[7] T. F. J.-M. Pasquier, J. Bacon, and D. Eyers, "FlowK: Information Flow Control for the Cloud," in *6th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, Dec 2014.

[8] J. Singh, T. Pasquier, J. Bacon, and D. Eyers, "Integrating Middleware with Information Flow Control," in *International Conference on Cloud Engineering (IC2E)*. IEEE, 2015.

[9] N. Dhanjani and G. Rodriguez-Rivera, "Kernel Korner: Loadable Kernel Module Programming and System Call Interception," *Linux Journal*, no. 82es, Feb. 2001.

[10] T. Garfinkel, "Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools." in *NDSS*, vol. 3, 2003, pp. 163–176.

[11] R. N. Watson, "Exploiting Concurrency Vulnerabilities in System Call Wrappers." *WOOT*, vol. 7, pp. 1–8, 2007.

[12] B. Niu and G. Tan, "Efficient user-space information flow control," in *Proc. 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013, pp. 131–142.

[13] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in *26th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2012, pp. 15–22.

[14] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information Flow Control for Standard OS Abstractions," in *21st ACM Symposium on Operating Systems Principles*, 2007, pp. 321–334.

[15] D. Schultz and B. Liskov, "IFDB: Decentralized Information Flow Control for Databases," in *8th ACM European Conference on Computer Systems (Eurosys)*. ACM, 2013, pp. 43–56.

[16] R. K. Ko, M. Kirchberg, and B. S. Lee, "From System-centric to Data-centric Logging-accountability, Trust & Security in Cloud Computing," in *Defense Science Research Conference and Expo (DSR), 2011*. IEEE, 2011, pp. 1–4.

[17] K. Hon, C. Millard, C. Reed, J. Singh, I. Walden, and J. Crowcroft, "Policy, Legal and Regulatory Implications of a Europe-Only Cloud," Queen Mary University of London, School of Law, Tech. Rep., 2014. [Online]. Available: http://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID2527951_code1577160.pdf

[18] T. Pasquier and J. Powles, "Expressing and Enforcing Location Requirements in the Cloud using Information Flow Control," in *International Workshop on Legal and Technical Issues in Cloud Computing (Claw'15)*. IEEE, 2015.

[19] J. Bacon, D. Eyers, T. Pasquier, J. Singh, I. Papagiannis, and P. Pietzuch, "Information Flow Control for Secure Cloud Computing," *IEEE TNSM SI Cloud Service Management*, vol. 11, no. 1, pp. 76–89, March 2014.