PrivateFlow: Decentralised Information Flow Control in Event Based Middleware (Demo) *

I. Papagiannis*, M. Migliavacca*, P. Pietzuch*, B. Shand+, D. Eyers† , J. Bacon†
Imperial College London*, Clinical and Biomedical Computing Unit+, University of Cambridge†
{ip108, migliava, prp}@doc.ic.ac.uk, Brian.Shand@cbcu.nhs.uk, {dme26, jmb25}@cl.cam.ac.uk

ABSTRACT

Complex middleware frameworks are made out of interacting components which may include bugs. These frameworks are often extended to provide additional features by third-party extensions that may not be completely trusted and, as a result, compromise the security of the whole platform. Aiming to minimize these problems, we propose a demonstration of PrivateFlow, a publish/subscribe prototype supported by Decentralized Information Flow Control (DIFC). DIFC is a taint-tracking mechanism that can prevent components from leaking information. We will showcase a simple deployment of PrivateFlow that incorporates third-party untrusted components. In our demonstration, one of these components will try to leak sensitive information about the system's operation and it will fail once DIFC is activated.

1. MOTIVATION

Computer systems are frequently subject to attacks that not only stop their operation but also result in leaking inappropriate information to the attackers. These problems are often the outcome of poor system design, bugs in the application code or malevolent components destined to leak information. Event-based systems similarly suffer from the above issues that only get worse as systems increase in size, incorporate a greater number of third party extensions and manipulate sensitive data. While a third party extension that generates a financial statement for an incoming event might be generally acceptable, in a privacy-aware healthcare environment guarantees must exist that it will never leak irrelevant sensitive patient data from the event to other components. Puting poor design aside, we focus on protection from programming errors and rogue components.

Decentralised Information Flow Control [2] (DIFC) is becoming an attractive paradigm for enforcing security properties in a collaborative setting. DIFC labels the data exchanged between components and assigns clearance levels to components. Using these labels, the system can monitor all information flows and actively prevent any interactions that could leak information to untrusted components.

DIFC was introduced as an extension to traditional Information Flow Control systems by allowing the system's components to take part in the creation of labels and the assignment of clearance levels. Since its introduction, DIFC has been used as a paradigm to build a new generation of operating systems [3] or existing operating systems' extensions [1]. These systems have demonstrated how existing software platforms, such as web servers and their hosted applications, can benefit from DIFC by introducing guarantees for users' data confidentiality and integrity.

2. DEFCON

Decentralised Event Flow Control (DEFCon) is our framework for building extensible middleware systems. It uses DIFC to control the isolation between components. DEFCon applications are composed of a set of *units* that run within a processing *engine*. The engines track information flow using *labels* assigned to units and ensure that units' operations do not violate flow constraints. DEFCon's label model builds upon Flume [1]. Each label is a set of *tags* while each tag is a randomly generated integer that expresses information flow concerns. Units have the ability to generate arbitrary tags and thus, they can express flexible policies.

DEFCon events consist of a set of *parts* and each part is protected by an individual label. As a result, parts represent data that must be equally treated in terms of information flow. This approach permits configurations where a given unit can have only partial view of an event, a view that is controlled by the unit assigned labels. Once a unit has observed confidential data, the tags protecting the data will be attached to the unit's labels. From now on, the unit will have to obey any restrictions that these tags introduce, thus it will lose the ability to communicate with other components not tainted with these tags. The unit is effectively "jailed" by its engine. The privilege to access data is called *clearance* over a tag.

On the other hand, *declassification privileges* allow a unit to ignore restrictions over a given tag and effectively escape the "jail" that clearance introduces. Giving untrusted units particular clearance privileges allows such units to do useful work, while still respecting security requirements.

3. PRIVATEFLOW

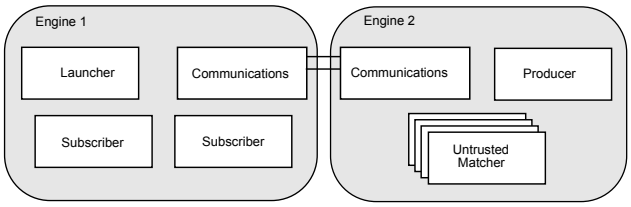
PrivateFlow is our prototype implementation of a simple topic-based, DIFC-enabled, publish/subscribe system built

*This work has been partially funded by ESPRC SmartFlow grants EP/F042469/1 and EP/F044216/1.

DEBS'09, July 6-9, Nashville, TN, USA.

on top of DEFCon. PrivateFlow's purpose is to demonstrate the possibilities that DIFC introduces for a publish/subscribe middleware system: to operate using untrusted components and yet guarantee the user's data confidentiality.

PrivateFlow is deployed as a set of units in two DEFCon engines. As illustrated below, the first engine includes a *launcher* unit to register new subscribers and a *communications* unit that connects the engine with a second engine. Label information is being preserved on that link. The second engine houses another *communications* unit as well as the message *producer*. Additionally, the second engine uses a third-party *untrusted matcher* that registers users' subscriptions and then matches them against publications.



PrivateFlow showcases two DIFC features. First, it protects the data within the messages from the matcher while avoiding intra-node encryption. Second, it protects the details of subscriptions so that the matcher is unable to communicate logs of the users' topic choices. Thus, both producers and receivers can use the untrusted matcher with confidence that it will not leak their sensitive information.

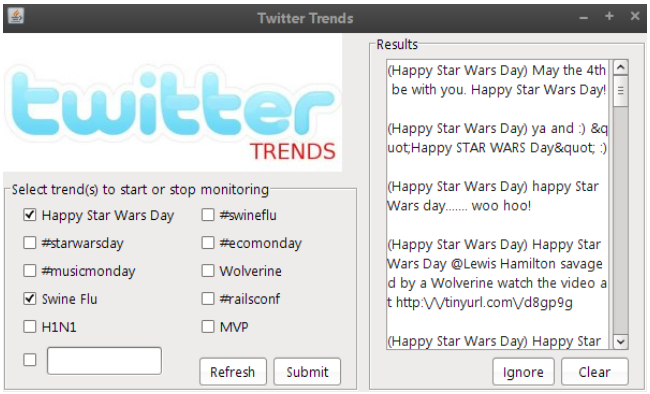
Publication data protection relies on DEFCon's multiple labeled event parts. Each published message is captured as a DEFCon event with three parts: a *type* part, a *topic* part and a *data* part. Each unit has different privileges over the tags protecting each part; the matcher specifically is prohibited to access the data part at all. The result is that the matcher is limited to only access the required topic part and it cannot leak other relevant information.

Subscription protection is achieved by introducing dynamically allocated per-subscriber tags. Unlike data part protection that can be enforced by a single statically allocated tag, each subscriber has to allocate its own tag at runtime. It uses this tag to protect its subscription and must delegate clearance over it to the matcher. PrivateFlow invokes a new instance of the matcher for each subscriber and taints each instance so that it is able to communicate only with the appropriate subscriber. This approach, imposed by DIFC, creates independent matcher states, so a matcher will never be able to send to a subscriber sensitive information (i.e. data and/or subscriptions) of another subscriber.

4. DEMONSTRATION OUTLINE

The demonstration will focus on presenting the two features described above that DIFC introduces to our publish/subscribe system. In order to do so, we will deploy two engines and some units, as described in the last section. When a new subscriber unit is instantiated, we are presented with the main subscriber interface.

The subscriber interface uses Twitter's search API to get a public preview of the current trend topics discussed in Twitter. It then populates a small form with possible subscription topics and waits for user interaction. Each user can select one or more topics to subscribe to.



In the second engine, the producer unit will do a similar job of fetching the trend topics and start publishing events on them to its engine. The demonstration will take place with two different settings: DIFC off and on.

DIFC turned off. The untrusted matcher will try and succeed in capturing both users' subscription information and the content of the messages that get delivered to them. Two subscriber instances will be instantiated and from each one of them, some subscriptions will be submitted. While the subscribers will be receiving their first events, a new output form will be generated by the matcher. The form will be dynamically updated with every subscription submitted by the subscribers and the data of each message that matched. Moreover, a third subscriber will be instantiated at engine 1 that will subscribe to a previously unknown topic named "hacker". This keyword will be understood by the matcher which, in response, will start to forward the above log to it. The subscriber's screen will then output a full log of other users' subscriptions, violating their privacy.

DIFC tuned on. We will repeat the steps of the previous case. Here the matcher will fail to generate an output form and a log message will be displayed stating that the matcher unit tried to violate DIFC. As more subscribers are instantiated, each corresponding matcher will fail to show output forms and it will correctly operate only with the corresponding user's subscriptions. Finally, another subscriber will be instantiated that will again issue the malevolent "hacker" subscription. Since the matcher instance that will receive it has no access to other units' subscriptions, the malicious subscriber will not receive any information.

5. CONCLUSION

DIFC provides a new paradigm for building secure event based systems. While the overhead that it introduces is yet to be explored, we believe that its advantages will enable more secure complex distributed applications.

6. REFERENCES

[1] M. Krohn, A. Yip, et al. Information flow control for standard OS abstractions. In *SOSP '07: Proceedings*, pages 321–334, New York, NY, USA, 2007. ACM.

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

[3] S. Vandebogart, P. Efstathopoulos, E. Kohler, et al. Labels and event processes in the Asbestos operating system. *ACM Trans. Comput. Syst.*, 25(4):11, 2007.