IN SUPREMÆ DIGNITATIS

1343

UNIVERSITÀ DI PISA

FogFaas

A faas extension of SecFog

Laboratory for Innovative Software

A. Y. 2019/2020

1 Introduction

The main objective of the project is to provide an extension of SecFog which solves the Components Deployment Problem adding some functionalities to meet the Faas paradigm, i.e. providing a way to deploy services on nodes of an infrastructure in such a way that some security requirements are met. In this setting, services are intended to be compositions of functions: the main point is thus to effectively and securely deploy functions on the given infrastructure.
To ensure a high level of security, we provide a type system which takes into account several kinds of implicit flows, togheter with further checks on the nodes. A labelling predicate must be defined by the user, in such a way that different security constraints can be easily defined.
Furhtermore, labels are intended to be ordered, so that we get a (semi)lattice which can be exploited to compute the security level of functions and, subsequently, of services.
As we model the Fog environment, we also provide constructs to allow communication between services and operations on resources. Moreover, we model triggers which allow clients to invoke a given service.
In the following, we cover the design choices and we provide the language syntax and semantics, along with some examples and use cases.

2 In class work

The in class work mainly focused on the deployment of functions. As said, the main functionality provided by FogFaas is the secure deployment of functions on an infrastructure: to get this, we provide a *placeApp* predicate. As an app is a list of services, the predicate call internally the *placeService* predicate, which in turn calls the *placeFunctions* one. Here (parallel or sequential) functions are deployed, according to the resources and the security guarantees of the nodes. The codebase also embed the trust model of SecFog and can be customized with different trust models. Except for this, basic functionalities of FogFaas don't require a probabilistic reasoning (while, as we'll see in the following, probability plays a key role in modelling communication).
In this preliminary work, we had a quite simple type system and few language constructs. The language has been later extended with several instructions, as well as for the type system which provides some new rules.

2.1 Example: testing FogFaas basic functionalities

As a main example, we provide a set of services and a complete lattice (in particular a strict total order) as a labelling, as shown in fig. 1.

This can be easily defined by the user by providing a suitable definition of the *labelF* predicate as follows:

```
labelF(ann, Args, ts).
labelF(ann, Args, s) :-
```

$$
\begin{array}{c}
\{l\} \\
| \\
\{s\} \\
| \\
\{ts\}
\end{array}
$$

Figure 1: Default lattice

```
                    findall(X, ts(X, Args), []).
labelF(ann, Args, l) :-
                    findall(X, notPublic(X, Args), []).
```

As we'll see, more complicated structures can be provided in a similar way: more on this later on.
In this case, we just state that

- any function can be top secret (in some sense, low level information can always be promoted to a higher level),
- a function can be secret iff it does not have top secret arguments,
- a function can be low iff it does not have top secret or secret arguments.

3 Extensions

3.1 Communication between services

We consider the problem of representing synchronous communications between services in a fog environment. In this setting, a deployment of the application is possible if the nodes hosting two communicating services are actually connected by a physical link which meet some security requirements as, for istance, some constraints on the geographical location of the traversed nodes.
In order to effectively represent communicating services, then, we have to represent the physical links and to provide a language construct for synchronous communications. To tackle the problem, we opted for representing physical links as bidirectional links. This adds a level of complexity in finding a solution to the routing problem, as we have to solve it on a non-directed, cyclic graph, but it seemed a reasonable representation of a physical network.
However, as said, we are representing synchronous communications between services, thus we provide a single language construct *send(Args, Service)* to this end. Furthermore, since synchronous calls can lead to starvation, we assume the instruction to be provided with a timeout, hence having two cases:

- the send operation succeeds: then the sent value is stored by the receiving service;

- the send operation fails for exceeding timeout.

The possibility of failure is taken into account exploiting probability. Thus the user can provide the probability of timeout exceeding using the *responseTime* predicate, which in turn is used inside the *ctx* predicate. If the communication succeeds, then the security context of the receiving service is updated according to the security level of the received value, otherwise it stays unchanged.

3.2 Extension of Tau

3.3 Type system

3.4 Trigger modelling

4 Putting it all togheter

5 Conclusions and future work