MoonFog: Policy-driven Monitoring of Fog Applications

Draft

Enrico Russo, Luca Verderame, Alessio Merlo
DIBRIS - University of Genoa
Viale F. Causa, 13, I-16145, Genoa Italy
name.surname@unige.it

***Abstract*—This paper introduces a proposal aimed at defining a novel methodology for run-time monitoring of Fog applications which is both *policy-driven* and *app-agnostic*. The first feature grants the possibility to define security policies that are enforced at run-time on a single or a set of Fog applications. The latter allows to enforce the security policies independently from the execution environment of the Fog applications (e.g., Virtual Machine, Container, PaaS, ...). The paper also discusses a PoC implementation on Cisco IOx.**

***Index Terms*—Fog Computing, Dynamic Analysis, Cisco IOx**

I. Introduction

Fog Computing is an emerging paradigm which adds novel computing resources to the edge of the network, thereby overcoming the current limitations of Cloud Computing. More in detail, the Cloud provides extended computing power to analyze data retrieved from sensors, mobile and edge devices. Unfortunately, the *distance* between such devices and the Cloud requires to move a lot of data from the edge to remote data centers in the Cloud, in order to be analyzed and take decisions. This fact produces delays that may not be tolerated by applications with specific time constraints. To this aim, Fog Computing introduces an intermediate computing level (see Fig. 1) provided by specific routers (hereafter, Fog nodes), that allows to analyze data just where they have been produced, thereby reducing delays, decentralizing computation and granting seamless activities.

A promising killer application for Fog Computing is Internet of Things (IoT), where a lot of small, task-specific devices produce a lot of data which, on one hand, do not require a huge amount of computing power to be analyzed, but, on the other hand, they have strict time constraints (i.e., they need to be analyzed as soon as they have been produced). To this aim, each Fog node interacts and retrieves data only from a subset of IoT devices (e.g., those belonging to its local network), and host domain-specific or environment-specific applications able to analyze data produced by that subset of IoT. From an holistic perspective, Fog Computing has a decentralized, heterogeneous and domain-specific nature which strongly differs from the centralized approach of Cloud computing.
From a security standpoint, Fog Computing introduces novel

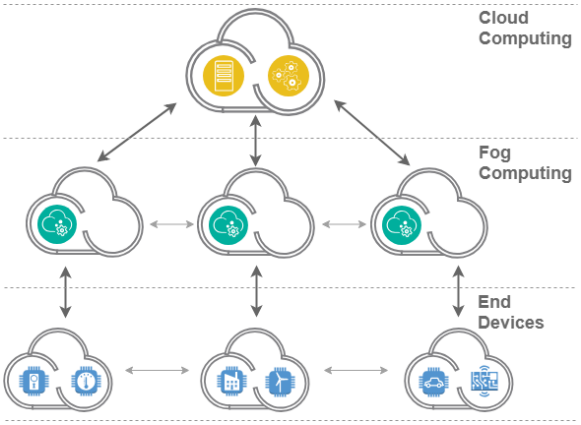

Fig. 1. The Fog Computing paradigm.

security challenges which are far different from the Cloud ones. In this paper we focus on Fog application security. In a nutshell, each Fog node has a multi-programmed OS (e.g., Cisco IOx[1]) which allows executing several applications at a time. Currently, Fog OSes offer basic security mechanisms to applications, but they do not provide any way to systematically analyze their behavior, nor to restrict their interactions with other applications and the underlying Fog OS. Therefore, it is very hard now to define and enforce any kind of user-defined *security policy* on a single or a set of Fog applications.
To overcome such limitation, in this paper we propose a novel policy-driven run-time monitoring methodology for Fog applications which supports the following features:

1) *Turing completeness*. The definition of the security policies is carried out through a simple (i.e., human-understandable) and Turing complete policy specification language.
2) *Technology neutrality*. The run-time behavior of Fog applications is analyzed independently from their underlying technology (i.e., Virtual Machines, Linux Container, PaaS, ...).
3) *Reduced invasiveness*. The run-time monitoring environment is implemented on existing Fog OSes requiring (at

[1]https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html

most) a minor customization of the OS.

Organization of the paper. The rest of the paper is organized as follows: Section II discusses the related work, while Section III introduces some background on Fog Computing applications. Section IV presents the proposed methodology, while Section V discusses its implementation on Cisco IOx. Finally, Section VI concludes the paper, pointing out some future extensions of the work.

II. Related Work

Since Fog Computing is an extension of the Cloud paradigm, it inherits many of its security and privacy challenges **roman2018mobile**, **alrawais2017fog**. Among them, the security of the interaction among Fog applications has driven the attention of the scientific community, mostly from an *access control* point of view **zhang2018survey**. For example, Salonikias et al. **salonikias2015access** presented a distributed RMAC scheme for Fog Computing. In this scheme, security policies and attributes are preserved in a distributed policy information point (PIP) and the policy decision point (PDP), that is in charge of making decision according to the access control policy, is implemented on Fog devices; the policy enforcement point (PEP), that enforces the access decisions, are instead implemented on the edge of the network. Furthermore, Fan et al. **fan2017secure** point out that Ciphertext-policy attribute-based encryption (ABE) can help to achieve data access control in fog-cloud systems. Hence, the authors propose an access control scheme based on a verifiable outsourced multi-authority.

However, one of the most relevant examples is the policy-based resource access control framework proposed by Dsouza et al. **dsouza2014policy**. Such framework adopts the eXtensible Access Control Markup Language (XACML) to define formalized and refined operational, security and network policy specifications. However, the proposal is just a preliminary framework which does not include a lot of details regarding how to i) build the policy repository, ii) identify a user, iii) take a decision, and iii) protect the identity and grant data privacy. Furthermore, this work requires the inclusion of additional dedicated resources within Fog nodes that may introduce some operational latency and are subjective to DoS attacks **Khan2017**.

Similarly, Nguyen et al. **nguyen2018security** propose a Model-Driven Security policy enforcement framework, named MDSioT, for IoT tenant apps deployed at the Edge layer. The framework allows generating security enforcement code, called gatekeepers, for different kinds of IoT tenant apps, and deploying a tenant app with its corresponding gatekeeper, that acts as a local PDP-PEP, in the edge server. Although promising, MDSioT requires substantial architectural changes, e.g., the introduction of an intermediate layer to intercept requests using the gatekeeper patterns, and it is mainly focused on architectural designs, policy modeling, and engineering approach without a definition of the run-time monitor solution. To the best of our knowledge, this work is among the first proposals for the definition of a methodology for the run-time monitoring of Fog applications based on user-defined security policies.

III. Background and motivation

The main aim of Fog Computing is to extend the Cloud functionality towards the field devices. In this respect, some interconnected nodes (e.g., switches, routers, industrial controllers) become providers for computing, storage, and network connectivity. Fog applications represent the consumers of such new resources. A typical pattern of an application running in the Fog is i) monitoring and acquiring data from network-connected things (e.g., IoT devices), ii) analyzing the collected information, and, iii) starting an activity as a consequence of the results of the analysis phase. These actions involve machine-to-machine (M2M) communications, e.g., opening a valve, locking a door or changing a device setting, and human-machine interaction (HMI), e.g., creating a graphical chart or sending an alert.

Due to the heterogeneity of such activities, Fog nodes allow executing applications in different environments which also depend on the manufacturer and the class of hosting devices. In particular, Fog devices typically support at least the execution of the following packaged applications:

- *Virtual Machine (VM) packaged applications*. They consist of a traditional virtual machine containing an operating system, libraries and application code. Fog devices can host a hypervisor to run such packaged applications.
- *Platform as a Service (PaaS) style applications*. They are self-contained programs developed using high-level languages, e.g., Java, Python, and Ruby. Fog devices provide them with the execution environment, provided as a service.
- *Container applications*. They depend and are designed to execute directly on the operating system of the Fog node. This solution, as opposed to PaaS style, depends on the features of the Fog operating system, but it leaves the complete flexibility to developers on the choice of the programming language, as well as the full framework stack. Fog devices often leverage the Linux Container (LXC) paradigm **ber14containers**.

Each application, regardless of the packaging method, also contains a standard descriptor file containing the hardware requirements that the application needs from the Fog node, in order to execute properly (e.g., computing and network resources).

Beyond some specific features related to the execution environment, a Fog application is made of a set of standard components. To this aim, Figure 2 depicts such components as well as their interaction with the Fog environment. In particular, a main component, namely the *Field Device Connector*, interacts with edge devices, e.g., IoT or field devices, using standard protocols stacks, e.g. CANbus, Modbus, and I2C, and acquire raw data. Moreover, further data can be retrieved directly from the Fog platform itself, e.g., from services like GPS or other connected serial devices, using a *Device Connector* component. A *Data Processing* component is in charge

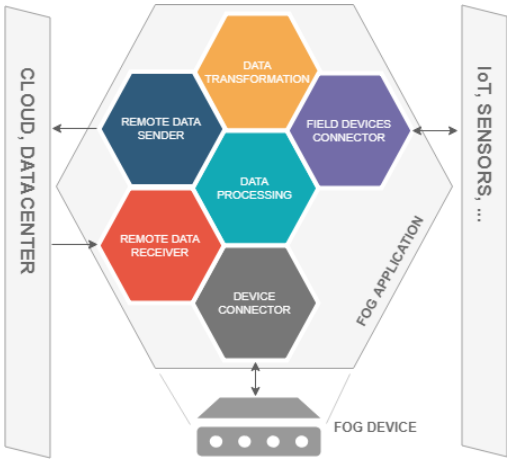

Fig. 2. Common components of a Fog application.

to elaborate the above inputs in order to apply configurable rules of filtering, reduction, and analysis. The output is then processed by a *Data Transformation* component which applies some custom business logic to render acquired data using a standard representation format, like e.g., JSON or XML. This transformation allows a *Remote Data Sender* to send the processed data to a centralized application, hosted in the cloud or in an external data-center, using a web-friendly protocol, like e.g., HTTP and MQTT. Finally, a *Remote Data Receiver* provides a web interface, e.g., a RESTful web service, which allows the Cloud or an external data-center to interact with the Fog application itself.

Since Fog applications are made by a set of modular components, each with specific functions which interact one another through standard protocols, the Fog Computing paradigm is actually implemented as a service-oriented middleware **AlJaroodi2012ServiceorientedMA**. In details, the Fog Computing commonly relies on a *microservices* approach **namiot2014micro**, which is likewise adopted in the IoT environment **Jarwar2018MicroservicesIW**, also.

As a consequence, a Fog application internally implements specific tasks and leverages reusable components, named *services*, for granting basic functionalities. For instance, some examples of core services available to all applications are: IoT and field devices protocol handler, access to Fog device resources, data processing and transformation facilities, and cloud connectors. As Fog applications, also services specify their requirement and dependencies with a description file and, in addition, expose their capabilities through a service contract. All in all, the Fog model relies on applications and services, running in different containers, interacting with one another and with the field devices. Such interaction is enabled by a *Message Broker* providing a shared and authenticated communication channel for each involved entity.

A. Fog Computing: Security considerations

On one hand, the adoption of Fog Computing provided a set of novel functionality. However, on the other hand, it also introduced a novel attack vector (i.e., the Fog application), thereby extending the existing attack surface of the Cloud paradigm. In particular, attack vectors can include:

- *Malicious applications*. A third-party application, possibly even digitally signed, is installed for exploiting the functionalities specified in the package descriptor. In this scenario, the application interacts with unknown applications or services and can executes malicious code.
- *Unexpected verbs or input*. An exploited application interacts with other scheduled applications or services submitting unexpected data. For example, a write command can be sent to a device which expects to receive only read inputs.
- *Violation of logic flaws*. Typically, an action is the result of previously happened events. For example, a service can be triggered to lock a door only if an application read a given value from a sensor and only after sending an alarm and receiving a human acknowledge. In this scenario, an attacker could interact directly with the door service, thereby bypassing the expected flow.
- *Unrestricted data domains*. Some Fog services, e.g., the Remote Data Sender, can communicate with public clouds or, in general, with remote endpoints hosted in less secure networks. In a scenario with a weak segmentation, an attacker can exploit such services in order to create a bridge aimed at exfiltrating data from sensitive devices towards external networks.

All the above scenarios represent a violation of the principle of least privilege, i.e., each Fog service or application must be able to access, with the expected methods, only the data and resources that are required for its legitimate purpose.

We argue that such issues can be addressed (and, in the general case, the attack surface of Fog Computing can be reduced) by bounding the runtime behavior of applications to user-defined security rules. In fact, from a security standpoint, the main limitation of Fog Computing in the actual deployments is that the security mechanisms applied to applications is still limited, as discussed in Section II. As a possible solution, in the following we propose a methodology that allows to enforce complex security policies to Fog applications at runtime.

IV. Methodology

We present here a run-time monitoring methodology that is able to evaluate the interaction among Fog application components according to a set of security rules, i.e., *policies*. In detail, a policy rule describes the security-relevant behavior in the interaction among *(i) applications*, *(ii) services*, and *(iii) devices* according to:

- *Message content*. The rule can enforce the usage of a specific verb, encryption method or data format. For instance, a Fog service may require the use of HTTP methods for HTTP REST APIs or a particular range of values for typed data included in the message.
- *Scope*. Each component can be labeled with identifiers, called *scopes*. Scopes are a logical way to group

components according to common properties (e.g., the usage of sensitive resources or the hosting in a specific zone). For example, a device that deals with not-aggregated data (and thus labeled with *sensitive_resource* scope), cannot communicate with components that can access to an external network (i.e., labeled with *external_networks_access* scope).

- *System Context*. The usage of context-aware rules allows for the definition of complex behaviors that depend on the overall state of the Fog node. For instance, an app is allowed to send an alert to HMI iff one of its sensor has a specific state value.

Each Fog component can expose security requirements to the policy monitor through a *Security Contract*. The security contract extends the package description of standard Fog components by including the required security rules.
Moreover, our methodology allows for the definition of global policies that extends the components security contracts with system-wide rules. Such rules may include requirements from the node manufacturer, field devices, and standard regulations.

Figure 3 summarizes the proposed monitoring solution. A runtime monitoring module (the **Fog Monitor**), embedded in the Message Broker of the Fog node, accesses all messages exchanged among those Fog components which expose individual *Security Contracts*. For each incoming message, the Fog Monitor determines if the communication request is allowed according to *i)* the security contract of the source, *ii)* the security contract of the destination, and *iii)* the global policies contained in the **Policy DB**. Furthermore, the Fog Monitor maintains the set of the Security Contexts of the Fog node inside the **Context DB** that are used for the policy evaluation.

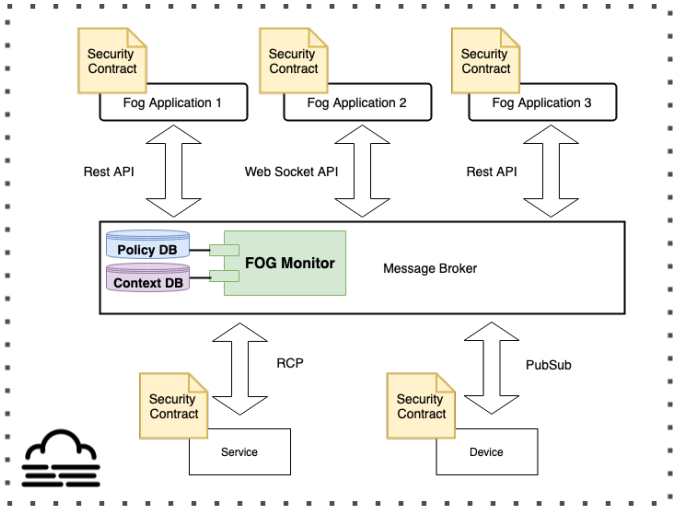

Fig. 3. Runtime Monitoring Solution

V. Notes on Implementation

We developed a prototype implementation of our monitoring methodology for *Cisco IOx*[2] Fog nodes. Cisco IOx, which is the leading application environment for Fog devices, is based on Cisco IOS and Linux OS and provides hosting capabilities for Fog applications across a network infrastructure.
Cisco IOx supports the microservices approach and allows the Fog components, i.e., applications, services and devices, to interact one another through a global message bus called *North Bound Interface* (NBI) Gateway. Since the NBI represents the single interface point for components communication, we embed the FOG Module inside the Gateway to execute the runtime monitoring.
The interaction between components relies on standard protocols like REST, Web Sockets, RPC and Cloud Pub/Sub and communication supports JSON serialized data to provide language agnostic exchange of data types. For this reason, the monitoring module is able to decode protocols and take decisions based also on message content and typed data.
Applications and services expose package description files and service contracts using the YAML format. We developed components Security Contracts by extending the above files with a dedicated section for security rules.
Security policies are written in ConSpec (Contract Specification Language) **aktug2008conspec**, which we already adopted for specifying both global policies and components contracts for Android applications **Armando2014219**, **computer14**, **Armando2013176**. The current implementation requires to be extensively tested in the wild. However, it allowed to prove the viability of the approach as well as the possibility to implement a run-time monitoring solution for Fog applications with a reduced invasiveness on the underlying Fog OS.

VI. Conclusion and Future Work

This paper introduced a novel methodology for the run-time monitoring of Fog applications, as well as the enforcement of user-defined security policies during the execution of the application. This project is still on-going. We identify three main extension of this work: 1) a complete implementation of the methodology on Cisco IOx, so that it can be ported and executed on a set of current Fog-enabled Cisco devices; 2) the definition of a set of promising security policies for the actual ecosystem of Fog applications, and, 3) the validation of the methodology in the wild, by deploying the methodology on actual Fog devices and by analyzing an extensive set of actual Fog applications. As a side effect of this latter activity, we expect to better understand the boundary of the attack surface of actual Fog Computing deployments.

[2]https://developer.cisco.com/docs/iox/