THORIN: an Efficient Module for Federated Access and Threat Mitigation in Big Stream Cloud Architectures

Luca Davoli
University of Parma

Laura Belli
University of Parma

Luca Veltri
University of Parma

Gianluigi Ferrari
University of Parma

In order to make cloud services attractive for several IT organizations, it is necessary to provide access control and implement safe and reliable mechanisms of Identity and Access Management (IAM). This article focuses on security issues and challenges in the design and implementation of cloud architectures and, in particular, for the management of Big Stream applications in Internet of Things (IoT) scenarios. The proposed work introduces a new set of modules allowing a federated access control policy for cloud users. We present an analysis of possible threats and attacks against the proposed Big Stream platform, investigating the system performance in terms of detection and elimination of malicious nodes. In particular, we propose a new module, denoted as Traffic Handler Orchestrator & Rapid Intervention (THORIN), which is very efficient in counteracting botnet-based threats.

Cloud computing can be described as an infrastructure computing paradigm which consists of a collection of interconnected computing nodes, servers, and other hardware, as well as software services and applications, that are rapidly and dynamically provisioned among end users with minimal management efforts. Services are delivered over the Internet, private networks, or through their combination, and are accessed on the basis of their availability, performance, and Quality of Service (QoS) requirements.[1]

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 ©2018 IEEE

Nowadays, cloud computing is constantly growing and so are concerns about data security and privacy that give rise to additional cloud-specific vulnerabilities (i.e., the possibility that a component will be unable to resist against the actions of a threat agent). Following the public, shared, and virtualized nature of the cloud computing paradigm, the migration of assets (e.g., data and applications) from an administrative control environment to a shared environment where many users are co-located, highly increases security concerns.[2–3]

From an infrastructure point of view, the services offered by the cloud paradigm are automatically managed following a multi-tenant model, exploiting virtualization technologies on one or more physical servers. Similarly to other paradigms, cloud computing needs to manage the digital identity of its users. A possible solution is represented by Identity Management (IdM),[4] corresponding to a set of capabilities (such as maintenance, administration, authentication and policy enforcement) used to ensure identity information and guarantee security. According to this perspective, since all data are outside the domain of the consumer, cloud computing opens new scientific challenges about access control, security, and privacy. In order to make cloud services attractive for organizations, it is first necessary to provide access control and implement safe and reliable mechanisms of Identity and Access Management (IAM). Moreover, some IAM systems allow management of federations, which are groups of organizations establishing trust among themselves in order to have safe business cooperation and adopt identities denoted as federated. In this type of IAM systems, once a user is authenticated by an organization of the federation, he/she can use all its services as well as those of another federated organization, without the need to repeat the process of authentication. This optional behavior is not trivial, because it requires interoperability between different systems and security technologies. In fact, each organization could have specific authentication and IAM mechanisms, making interoperability challenging.

Another relevant trend rapidly growing in recent years is the Internet of Things (IoT), corresponding to a worldwide "network of networks" involving billions of different devices (nodes) connected to the Internet.[5] The IoT allows new forms of interaction among things and people, generating huge amounts of data, which can be processed and employed to build services for consumers in several scenarios (e.g., Smart Cities, monitoring and surveillance applications, e-health). Due to its scalability, robustness and cost-effectiveness, the cloud is, on one hand, the natural collection environment for data retrieved by IoT nodes; and, on the other hand, the ideal service provider for consumers' applications.

In our previous work,[6] we proposed a graph-based cloud architecture for the IoT, describing in detail the modules of the framework and the technologies employed for an open source implementation. This preliminary work does not take into account any aspect related to security. In another work,[7] we introduced new modules (denoted as In-Graph Security (IGS) and Outdoor Front-end Security (OFS)) to the architecture, aiming at increasing the platform security. In particular, we addressed the problem of "core" security inside the graph.[7]

We extend this platform architecture here to take into account also the access control problem (or "outer security") and the management of malicious nodes in the graph. In particular, we propose a new module, denoted as "Traffic Handler Orchestrator & Rapid Intervention" (THORIN), which is very efficient in counteracting botnet-based threats. Our results show that the proposed architecture can efficiently react against attacks of a malicious node, "sanitizing" the graph and removing compromised nodes in a transparent way, while other nodes remain unaware of the changes occurred.

RELATED WORK

In order to allow access control in open environments, IdM can be implemented in different ways: (i) in-house implementation: identities are issued and managed by single companies, which have complete responsibility on them; (ii) managed implementation: the Identity-as-a-Service (IDaaS) concept is adopted and IdM is outsourced to external companies. Regardless of the chosen configuration, an IdM System (IMS) controlling the platform identities always involves three main types of entities: (i) the user; (ii) the Identity Provider (IdP), responsible for issuing and managing user identities and credentials; and (iii) the Service Provider (SP), responsible for

providing services based on users' identities/attributes. An IMS must always provide the following functions: (i) provisioning of identities related to the different types of accounts adopted by the organization (e.g., administrator, IT consultant, developer, end user); (ii) authentication and authorization functions, identifying individuals through various mechanisms (e.g., username/password, X.509 certificates, OTP, biometrics) and providing them different access levels for different operations within a computing infrastructure; (iii) grouping SPs into a federation and, then, establishing a trust that allows sharing of identities' information among them.

Even if the topic is not fully explored, some works have recently appeared in the literature to address the problem of applying security mechanisms to the IoT-cloud environment. The work of Lake and colleagues[8] analyzes security issues in an IoT-based e-health scenario, providing a framework mainly focused on vulnerabilities associated with: (i) endpoint access; (ii) cloud services; and (iii) partners and providers. The work of Suciu and colleagues[9] explores a Smart Cities scenario, focusing on the problem of securing cloud APIs for sensing and actuation in an open sensor platform. Regardless of the selected use case, the most relevant technologies employed for implementing an IMS can be summarized as follows.

- Active Directory (AD): a directory service created by Microsoft for Windows domain networks.
- Security Assertion Markup Language (SAML): an XML-based open standard for exchanging authentication and authorization data between security domains, i.e., between an IdP and a SP.
- Single Sign-On (SSO): a mechanism for access control of multiple (related, but independent) software systems, in which a user logs in only once and gains access to all related systems without the need to log in again.

Storing and managing the identities present crucial security concerns for cloud providers, because stored information can be tampered or modified by malicious or unauthorized users. Several technologies implement federated identity, such as the Shibboleth system.[10] Shibboleth is a widely employed Authentication and Authorization Infrastructure (AAI) based on SAML that allows to create a safe structure that simplifies the management of identities and provides the user a SSO layer for different organizations belonging to the same federation, in which identity information is shared. A Shibboleth system is built on two main entities, the IdP and the SP, and on an optional one, the Where Are You From (WAYF) module. The Shibboleth IdP manages users' authentication, maintaining credentials and attributes. The SP is located where the resources are stored and accessed by the user. The optional WAYF component allows an association between a user and multiple organizations. When trying to access a resource, the user is forwarded to an interface that asks him/her to choose the organization which he/she belongs to; after this choice, the user is redirected to his/her correct IdM interface to start the authentication process.

The growing diffusion of IoT applications, based on large-scale sensor networks, has spurred research efforts in studying data streams processing and distribution mechanisms. However, several aspects related to data streams' security still need to be investigated. Puthal and colleagues[11] address the end-to-end data stream security problem (e.g., integrity and authenticity) for risk-critical applications scenarios. Their proposed approach, called dynamic key length-based secure framework (DLSeF) is based on symmetric key cryptography, with a common shared key that is generated by exploiting synchronized prime number. The framework aims at avoiding excessive communication messages for the rekey process between data sources and the centralized collection point, called Data Stream Manager (DSM). By reducing the overall communication overhead, the DLSeF module is able to perform all security verification processing activities on-the-fly, avoiding data stream buffering. This solution is suitable for stream management applications where the both the endpoints of the communication are known, and cannot be adopted when data streams can be dynamically combined, without fixed source and destination points.

Another relevant issue is given by the authentication and privacy problem in the continuous sensing scenario. Emerging technologies, such as Google Glass and Microsoft Kinect, rely on continuous recording of audio or video to provide services to users (i.e., voice command or gestures); however, these capabilities raise serious privacy and security concerns. Roesner and colleagues[12] propose a framework for controlling access to sensor data on multi-application

continuous sensing platforms. This approach is based on world-driven access control, which allows real-world objects to explicitly specify access policies. In the authors' vision, users themselves do not specify any policy, but devices automatically detect policies broadcasted by real world's objects through a particular certificate called passport. The proposed approach can be extended and is suitable to scenarios where there is an explicit interaction between humans and things. It does not cover other possible applications related to the IoT scenarios.

THE BIG STREAM CLOUD ARCHITECTURE: CONCEPT AND BASIC IMPLEMENTATION

Several relevant IoT applications (e.g., alerting and monitoring systems[13]) require real-time performance or, at least, a predictable (and consistent) latency. On one hand, the large number of IoT data sources globally generate data at a high rate (even though a single IoT service generates a limited amount of data); on the other hand, the low-latency constraints call for innovative cloud architectures, able to efficiently handle such massive amount of information. A possible solution is given by Big Data approaches, which address the need to process extremely large amounts of heterogeneous data for various purposes. However, these techniques typically have an intrinsic inertia and focus on the data itself, rather than providing real-time processing and dispatching. For these reasons, Big Data approaches might not be the right solution to manage the dynamicity of IoT scenarios and, thus, Belli and colleagues[6] proposed a shift from the Big Data paradigm to a new one, denoted as "Big Stream." A cloud-oriented graph-based architecture, explicitly designed to fit Big Stream IoT scenarios, has also been proposed and implemented.[6–7] In order to minimize the delay between the instant of raw data generation (e.g., at the sensors) and the instant at which properly processed information is provided to the final consumer, the proposed platform adheres to the publish/subscribe model and is based on the concept of "listener." More in detail, the fundamental components of the graph-based cloud architecture are the following.

- Nodes: processing units which process incoming data, not directly linked to a specific task. A graph node can be a listener of one or more streams and a publisher of a new stream for other nodes. Nodes in the graph are logically organized in layers, characterized by an increasing degree of complexity.
- Edges: streams linking together various nodes, allowing complex processing operations.

Although the Big Stream architecture outlined above can be deployed on a single server, exploiting it on the cloud allows to improve the system's scalability and to manage the workload with a huge number of data sources and processing nodes. Another important benefit, brought by the use of cloud, is that it provides a common platform in which data streams can be shared among several actors (e.g., IoT data sources, developers or consumers), in order to build useful services for different consumers. Therefore, this architecture is mainly intended for developers interested in building applications based on data originated by IoT networks, with real-time constraints and low overhead. The platform provides the following services for developers: (i) processing nodes upload/deletion; (ii) internal stream status; (iii) external data source upload/deletion (i.e., a new IoT provider). Developers authenticated on the platform can thus customize paths in the graph through the definition and deployment of new processing nodes. It is important to observe that, by accessing the Big Stream architecture, each developer can operate on data streams coming from IoT networks which he/she does not own.

SECURITY IN THE BIG STREAM ARCHITECTURE

In the proposed Big Stream architecture, security was originally implemented in two clearly separated levels[7]: (i) inner security, through the IGS module, and (ii) outer security, with the OFS module. As shown in Figure 1, the IGS module manages the secured streams, defining a set of rules indicating which actors are authorized to send/receive data to/from each single processing node. To accomplish this task, the IGS module works with the Application Register (AR) module, which is composed by the following components (shown in Figure 1):

- the Graph State Database (GSDB) and the Node Registration and Queue Manager (NRQM) modules, which cooperate to manage the authorization policies adopted by the graph nodes;
- the Policy Manager and Storage Validator (PMSV) module;
- the Graph Rule Authorization Notifier (GRAN) module; and
- the Persistent Security Storage Container (PSSC) module.

The OFS module operates at the borders of the platform, after receiving input data from IoT networks, as well as before pushing out streams to final consumers. OFS has a crucial role, since it authorizes the interactions between the external entities and the frontier of the cloud platform. Since OFS should support both "open" and "secured" communications through different protocols (e.g., HTTP, CoAP and MQTT), in this module some security mechanisms have to be implemented (e.g.: at the application layer through S/MIME or OAuth; at the transport layer through TLS/DTLS). The IGS module allows to define paths in which some segments are "secured" and some others are "public." Beside the basic functionalities outlined above, security features in the core part of the platform itself need to be improved, adopting policies (e.g., access control and federated authentication) like those explained in the following and, in particular, those provided by the THORIN module.

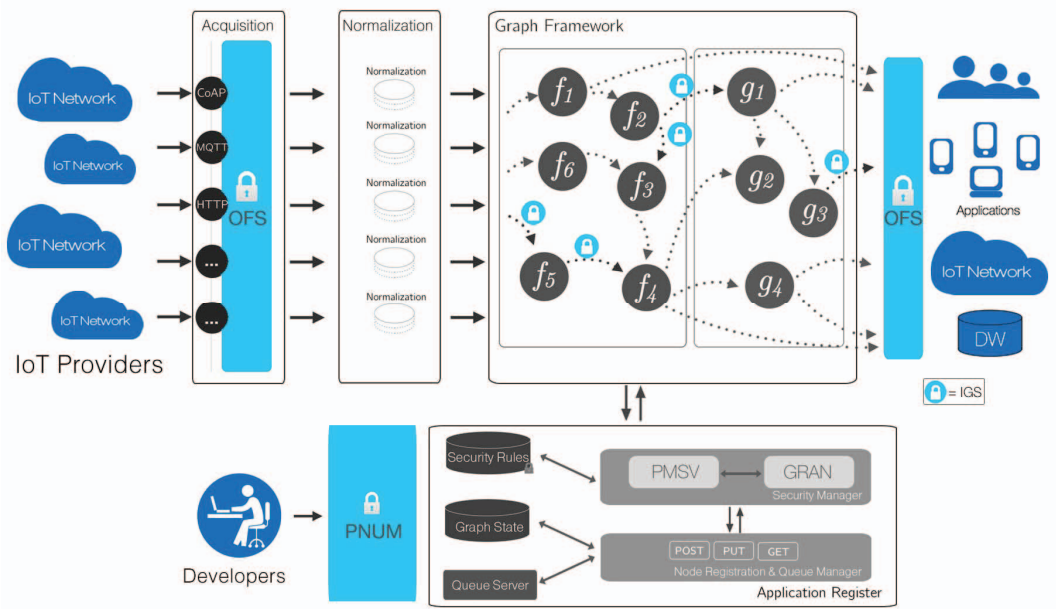

Figure 1. Details of the security modules embedded in the proposed Big Stream architecture.

Federated Access Control in Big Stream

Following the SaaS paradigm, the Big Stream architecture gives external providers the possibility to upload into the graph custom nodes containing executable (software) code. Although this functionality is an added value for the platform, it clearly opens several security vulnerabilities. A possible countermeasure is the adoption of an encrypted channel on which a developer can upload its nodes via a transfer protocol (e.g., using SFTP). Another one is represented by a federated authentication system. This approach allows to create a federation of trusted entities aiming at accessing to the Big Stream platform, through a federated access paradigm (e.g., Shibboleth) and to share their own nodes. Both these solutions are handled by the Processing Node Upload Manager (PNUM) and are depicted in Figure 2.

The adoption of a federation-based access paradigm allows to get rid of the need to store and maintain an account for each developer, reducing system vulnerabilities and making the architecture more scalable and accessible through the Web. This type of authentication mechanism can

also be adopted for the output stage of the Big Stream platform, in order to authenticate a final consumer requesting a stream coming from the higher layer nodes. By exploiting the capabilities of the proposed Big Stream platform, together with its acquisition module, it is possible to define policies to maintain a sufficiently high QoS for external providers. This can be carried out through the arrangement of the following special communication channels for secured streams.

- Reservation of acquisition nodes, providing a clone for each requested communication protocol (e.g., CoAP, HTTP): this is possible in a limited time (e.g., on the order of the startup time of a Java process, between 1 s and 5 s), because of the highly dynamic nature of the system and the specificity of the nodes that have to be cloned.
- Transfer of the processing nodes, that have to be deployed into the platform, encapsulating them into a secure communication tunnel (e.g., through IPSec) after federated authentication.
- Encryption of the transferred data through a public/private encryption mechanism, using previously negotiated keys between the Big Stream platform and the IdM of the organization which the external provider belongs to.

As previously mentioned, the same solution can be adopted at the output stage of the Big Stream platform, if a final consumer needs some additional streams. These security measures allow to control and manage the access requests, denying the injection of malicious nodes. Moreover, in the case of heavy attacks (e.g., if the platform is under requests bombing), the acquisition nodes at the input stage of the platform can be replaced in a limited time (e.g., on the order of the startup time of a Java process, between 1 s and 5 s) and the authorized traffic hijacked to the newly activated nodes.

There are several aspects that make the proposed Big Stream platform adhering to the cloud paradigm. First, it can be deployed on different commercial cloud services (e.g., Amazon EC2 and Microsoft Azure). Moreover, the internal graph organization platform can be assimilated to a wide cloud interconnection of systems, in which each single cloud entity corresponds to a single processing node. However, the Big Stream platform is vulnerable to a botnet-based attack, in which a multitude of systems get infected and become zombie machines, remotely controlled by Command-and-Control (C&C) servers, often hidden behind a Tor-based network. The success of this type of attack can provide the attackers with a high firepower, as the number of nodes running inside the graph of the cloud-oriented Big Stream architecture is large. For this reason, it is fundamental to deny the attackers any possibility (i) to catch the command of any processing node and (ii) to inject malicious nodes exploiting the cohesion of the Big Stream platform.

In order to protect the Big Stream platform against this botnet-based threat, we now propose a new security module, denoted as Traffic Handler Orchestrator & Rapid Intervention (THORIN), deployed in the graph platform and connected to the Application Register module.

THORIN

THORIN monitors the traffic on the overall platform, analyzing in real-time the behavior of the different components, with the help of heuristics and optimization techniques. The joint action of Application Register and THORIN modules aims at:

- controlling the interactions between the external providers and the acquisition front-end nodes of the Big Stream platform;
- analyzing the behavior of the internal graph of the platform, looking for suspicious activities carried out by processing nodes;
- reacting to malicious activities and protecting the other active nodes, allowing them to safely continue their operations.

A cloud-oriented infrastructure has to provide countermeasures against security threats related to both the interactions with external entities and the inner communications among graph processing nodes. Concerning the outside interactions, a security challenge is represented by the

need to guarantee access control, providing services only to authorized entities (i.e., IoT providers, developers, final consumers). This can be done, as previously highlighted, in a centralized way, as well as by delegating to a trusted third-party entity the management of the security policies (such as that adopted in our proposed platform, that allows to increase the number of collaborating entities and, at the same time, avoids the centralization of security management). In particular, THORIN implements a Shibboleth instance that allows different providers (linked to a wide federation) to connect and provide their processing nodes to the Big Stream platform. Considering communication between processing nodes, a challenge faced by the proposed architecture is its reactivity in the presence of security threats. Once authenticated, a federated user can use the architecture for its purposes: the presence of malicious users is thus a tangible security issue and proper considerations are needed to evaluate the security degree of the proposed platform. An authenticated malicious user can affect the system in different points of the infrastructure, as shown in Figure 2.

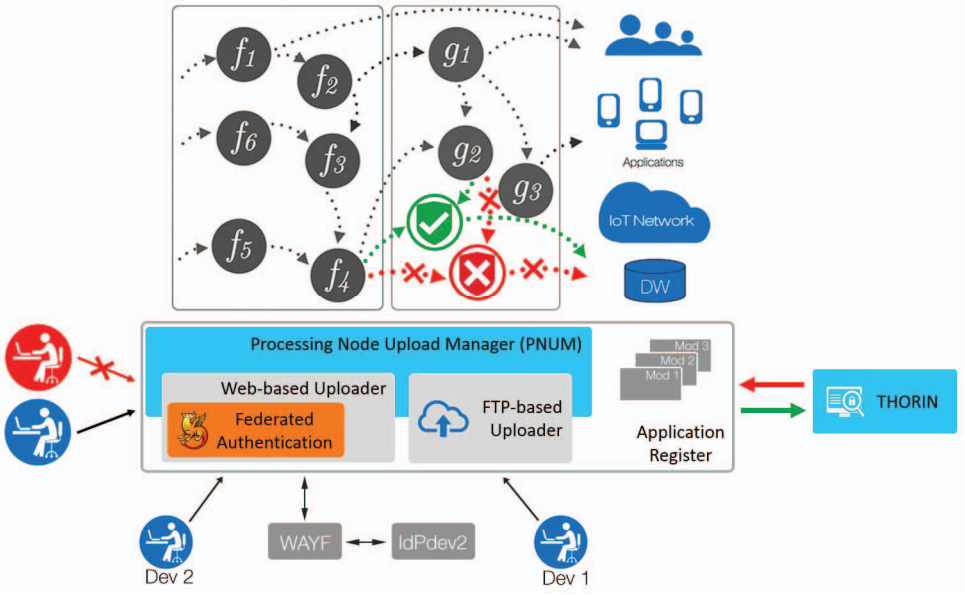

Figure 2. Big Stream platform's behavior to counteract internal and external security threats.

Thanks to its internal graph structure and the presence of THORIN, the Big Stream platform can withstand the attacks outlined in the previous paragraph better than a "standard" cloud-oriented architecture. We can summarize the motivations behind this robustness as follows.

First, being based on a publish/subscribe paradigm, the Big Stream implementation has some communication entities (namely, the "brokers") that can be clustered to improve the redundancy of functionalities and the system resilience. An attack against one of these brokers may be thwarted in a short time, removing it from the cluster and replacing it with a new one. In case of a "standard" cloud infrastructure, the reaction time increases, as a new virtual machine (VM) must be activated to replace the affected one that worked as broker before the attack, and its activation time is not negligible (in the order of hundreds of seconds). In particular, THORIN contains a particular Java module that replaces the broker under attack with a new (honest) one.

Second, when a federated node is recognized to have malicious behaviors (i.e., acting as DDoS attacker or as e-mail spammer), the THORIN module immediately reacts as follows: (i) shuts off the external provider of malicious data; (ii) removes it from the front-end interface; (iii) reports its behavior to the proper IdM; and (iv) prevents it from connecting to the acquisition interfaces of the Big Stream platform. This is shown on the left side of Figure 2, in which a malicious developer is banned from the architecture. Conversely, a "basic" cloud-oriented infrastructure requires the adoption of an external control software that handles all these operations. In particular, when a federated entity is admitted to be a provider/consumer of the Big Stream platform, THORIN transparently creates a hidden channel h_x (e.g., a RabbitMQ queue) that is linked and

used to communicate with the federated entity. In case of malicious behavior, THORIN (through its internal Java modules) immediately shuts-off and disconnects the federated entity and, through h_x, notifies the federated entity about the detected problem. Then, it avoids further connections from the ex-federated entity and deletes h_x, saving computational resources.

Finally, handling the security into the inner layers of the infrastructure, THORIN can intercept malicious behavior of the processing node (i.e., a zombie device in a botnet-based scenario) and can recover and restore, in a short time (between 1 s and 20 s), the full functionalities of the graph. THORIN does so through the following steps:

1. Isolating the internal malicious processing nodes, which are remotely managed by a malicious controller (e.g., a C&C server).
2. Protecting the other running nodes, allowing them to continue their execution safely.
3. Changing the routing keys associated with the streams under attack, to isolate the malicious nodes in the graph, denying the reception of further information by them and preventing them from sending malicious processed data to the broker working in their current graph layer (e.g., the red node in Figure 2).
4. Replacing the isolated malicious nodes with new clones (e.g., the green node in Figure 2), in order to reestablish the correct operational behavior of the internal graph.

Conversely, a "basic" cloud-oriented architecture needs more time than the one requested by the Big Stream platform, to power off the VM hit by a security attack and to replace it with a newly activated one, on the order of more than a minute.[14]

Regarding the monitoring activity of the inner layers, THORIN operates according to the following heuristic principles, namely: (i) running an internal Java module that aggregates and performs statistics on data flows, as well as (ii) interacting with external heuristic services and (iii) analyzing behaviors and comparing URLs used by inner processing nodes with publicly available dangerous blacklists (e.g., spam URLs, malware and ransomware URLs, darknet onion-like URLs, etc.).

SYSTEM EVALUATION ON A REAL USE-CASE

In order to prove the efficiency of the inner security approach guaranteed by the proposed architecture, the time required to isolate a malicious node (e.g., a node that tries to inject spam in a public news feed system) can be evaluated. Considering a real use-case, we assume that THORIN, during its runtime streams analysis, identifies a malicious Java node, denoted as M, injecting spam news containing dangerous URLs (e.g., targeting sites in the darknet). The node M has n Java "listener" nodes, each of them receiving q different streams produced by the malicious node. After the node M has been identified, THORIN should remove and replace a total of $q \times n$ streams, which, in our implementation, correspond to RabbitMQ queues. In Figure 3, the time (dimension: [s]) required to isolate the node M and its q streams by the other safe Java nodes $\{f_j\}_{j=1}^{n}$ is shown as a function of n, considering various values of q (from 1 stream/node to 50 streams/node). In the same figure, the reference attacked portion of the graph is shown. The obtained results show that, in the worst case, with 2500 streams to be shut down (namely, $n = 50$ listening nodes with $n = 50$ streams/node), the system reacts in less than 17 seconds.

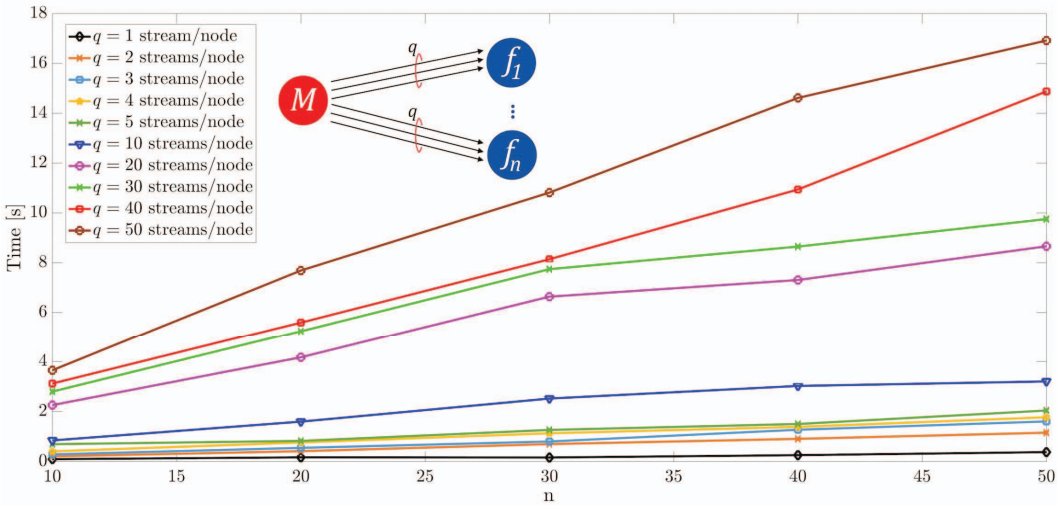

Figure 3. Performance results (in terms of reaction time) emulating an attack conducted by a malicious node *M* that affects *n* different listening nodes, each of them accepting incoming data on *q* streams.

The different steps involved in the deletion of the malicious node *M* by THORIN are the following:

1. isolating the malicious Java node *M*, removing its input/output RabbitMQ queues from the platform;
2. generating and binding new incoming RabbitMQ queues for the listener Java nodes $\{f_j\}_{j=1}^{n}$ to their proper RabbitMQ brokers, restoring the graph topology;
3. updating the modified queues details, related to Java nodes $\{f_j\}_{j=1}^{n}$ and to the malicious node *M*, into the SQL-based GSDB module, to maintain a consistent representation of the graph;
4. notifying each sanitized Java node $\{f_j\}_{j=1}^{n}$ of the updated consistent state, through reserved RabbitMQ queues exclusively binded between the same node and the Application Register module.

It is important to underline that: (i) the malicious Java node *M* is immediately isolated after step 1, executed in a very short time (in our evaluation, unbinding a RabbitMQ queue requires tens of milliseconds), while the rest of the time is needed to update the state of the graph in a consistent way; and (ii) the other Java processing nodes are unaware of the occurred changes, since all the previous steps will be orchestrated by the THORIN module and executed in a totally transparent way.

CONCLUSIONS AND FUTURE WORK

In this work, we have analyzed security issues and access control policies in cloud architectures, with particular attention to the management of Big Stream applications in IoT scenarios. We have defined the characteristics of the Big Stream paradigm and described the details of a recently proposed graph-based Big Stream architecture for IoT. The earlier implementation of the Big Stream architecture already takes into account inner security through the use of specific modules (with IGS, PMSV, GRAN, and PSSC modules). The current work has focused on the architecture's outer security and access control, introducing a new module, denoted as THORIN, which allows a federated access control policy for cloud users. An analysis of possible challenges and attacks has highlighted that the proposed architecture implementation, thanks to its graph-based structure and the consequent introduction of the THORIN module, can manage and