6-2017

P4SINC – An Execution Policy Framework for IoT Services in the Edge

Phu Huu Phung
University of Dayton, pphung1@udayton.edu

Hong-Linh Truong
Vienna University of Technology

Divya Teja Yasoju
University of Dayton

P4SINC – An Execution Policy Framework for IoT Services in the Edge

Phu H. Phung[1], Hong-Linh Truong[2], and Divya Teja Yasoju[1]
[1] Intelligent Systems Security Lab,
Department of Computer Science, University of Dayton, USA
http://academic.udayton.edu/PhuPhung/
[2] Distributed Systems Group, TU Wien, Austria
truong@dsg.tuwien.ac.at

***Abstract*—Internet of Things (IoT) services are increasingly deployed at the edge to access and control Things. The execution of such services needs to be monitored to provide information for security, service contract, and system operation management. Although different techniques have been proposed for deploying and executing IoT services in IoT gateways and edge servers, there is a lack of generic policy frameworks for instrumentation and assurance of various types of execution policies for IoT services. In this paper, we present P4SINC as an execution policy framework that covers various functionalities for IoT services deployed in software-defined machines in IoT infrastructures. P4SINC supports the instrumentation and enforcement of IoT services during their deployment and execution, thus being leveraged for other purposes such as security and service contract management. We illustrate our prototype with realistic examples.**

I. Introduction

To support on-demand IoT sensing and distributed analytics in the edge together with large-scale computation and analytics in clouds, mobile-edge computing, fog computing, and edge computing models [1], [2], [3] have been developed for several applications in building management, crowdsensing, geosports, to name just a few. The basic tenet of these models is that software components – performing sensing, analytics or controls – will be deployed from centralized clouds to IoT and edge resources, such as IoT gateways, edge servers, and micro-data centers. State-of-the-art tools allow us to easily deploy such software components, called *IoT units* in this paper, which are part of *IoT services*. Furthermore, several works have been focused on IoT service and application composition, such as [4]. However, few works have been developed to support *generic execution policy* that enables service contracts and trustworthiness aspects of IoT services deployed in IoT and edge infrastructures. Researchers have emphasized widely that enforcing execution policies for IoT is a crucial point [5].

A. Motivation

Let us consider a real-world scenario where a Telco company has several Base Transceiver Stations (BTSs) whose equipment need to be monitored: HVAC, backup electricity systems, and electricity generators. The Telco has deployed an IoT infrastructure for thousands of BTSs. In each BTS, there is an IoT gateway running Raspberry PI; this gateway has connected to sensors and actuators interfacing to equipment in the BTS. IoT software units read data from hardware sensors and send the data to the cloud through a MQTT (Message Queuing Telemetry Transport) broker. From the cloud, management services send commands to the equipment via IoT software units through the broker.

In our case, the Telco company has outsourced the maintenance of the HVAC for third-party companies. Any third-party maintenance company will deploy its IoT service which includes a thousand instances of (the same) IoT units in IoT gateways. This service not only monitors the HVAC but also controls HVAC in each BTS. If HVAC has a problem, the performance of the BTS will be impacted.

In this real-world scenario, the outsourced IoT units must be granted appropriate access to the gateways to function properly. When deployed on IoT gateways/edge servers, IoT units of the IoT service can perform any action/request. However, if an IoT unit contains bugs or unknown vulnerabilities, it can affect the security and performance of the BTS. For example, the IoT unit might read data from some sensors that it should not be allowed, might read data from allowed sensors too often, or might send data to a cloud service in an untrusted or non-compliant destination. In current tools, there is no control or mechanism to enforce such policies.

As we increasingly deploy software components of IoT services from the cloud to IoT gateways and edge servers, it is important to provide generic mechanisms to enable multi-purpose service contracts, security, and access controls for such IoT services. However, IoT infrastructures nowadays have multiple types of gateways, sensors/actuators with different protocols. Thus, to assure the execution policy, it is very challenging, if not impossible, to focus on individual types of gateways and sensors/actuators. Similarly, edge servers also have different capabilities. In this paper, we envisage the software-defined machine (SDM) [6] abstracting various underlying IoT and edge resources to be developed and adopted widely [7]. SDMs simplify the interface to Things via a set of APIs for data, control, execution and connectivity and we believe that execution policies support should focus on SDMs which encapsulate underlying functionalities of IoT gateways and edge servers for IoT units. Basically, it calls for

978-1-5386-2011-3/17 $31.00 © 2017 IEEE
DOI 10.1109/IEEE.ICIOT.2017.23

a new approach to deal with generic execution policies for IoT services deployed on SDMs.

B. Contributions

We introduce an execution policy framework – called P4SINC (Policy for Servicing IoT, Network Functions, and Clouds) – that covers

- *Generic policy model and specifications for SDMs*: Due to the diversity and complexity of underlying IoT devices and software units, policies need to be generic enough to support high-level interactions in IoT services. Our execution policy specification covers the generic functionality of SDMs that enable different purposes, such as security and service contract management.
- *Management of diverse types of policy utilities*: in IoT, we need to utilize different *utilities* to enforce security policies due to the diversity of IoT units. We propose and implement an integrated framework to allow instrumentation and deployment of IoT units using different policy utilities for different types of policies and units.
- *Dynamic enforcement mechanisms*: enforcement must be dynamic due to the change of IoT deployment and execution. We provide enforcement mechanisms to enforce policies per IoT unit, SDM and IoT service as a whole.

The rest of this paper is organized as follows: Section II describes our P4SINC framework. Examples and experiments are given in Section III. We discuss related work in Section IV, and present the conclusion and future plan in Section V.

II. P4SINC Execution Policy Framework

A. Machine Profile for Execution Policy

Fig. 1 presents stakeholders and their needs with regard to the execution policies. We have IoT services and units developed and deployed by the *IoT Service Provider* who uses SDM Profile (and its APIs) exposed by the *IoT Infrastructure Provider*. The *IoT Service User* utilizes IoT services so it just concerns with the APIs offered by the *IoT Service Provider* and other service constraints. The IoT Infrastructure Provider and the IoT Service Provider care about the SDM APIs utilization and they monitor SDM APIs to support, e.g., security and service contract, and profiling. To enable different purposes for these stakeholders, we have to capture the SDM Profile and use the profile for policy instrumentation and enforcement.

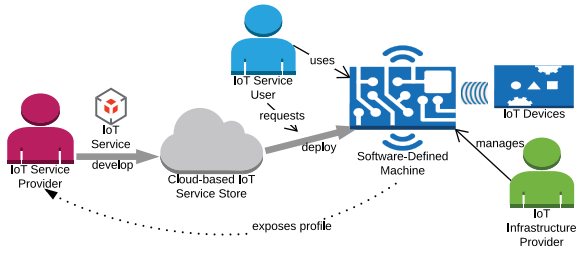

Fig. 1. Stakeholders with IoT services and infrastructures.

We use the main concepts of Software-Defined Machine (SDM) in [6] to abstract underlying resources for developing suitable execution policies. Each SDM is associated with a profile. Conceptually, a profile of an SDM (pSDM) includes a set of APIs categorized into different categories of functionality. The `pSDM` model allows the IoT services to be developed and monitored IoT resources, independent of each particular platform. In our framework, `pSDM` is an input for the instrumentation service to instrument the IoT services. In this paper, however, we assume that policies are given to us in defined forms that we can parse and perform according to instrumentation and monitoring. In this sense, we focus on the low-level policies and leave the high-level policy specification for the future work. We use JSON (http://www.json.org/) format for `pSDM`.

B. Framework Overview

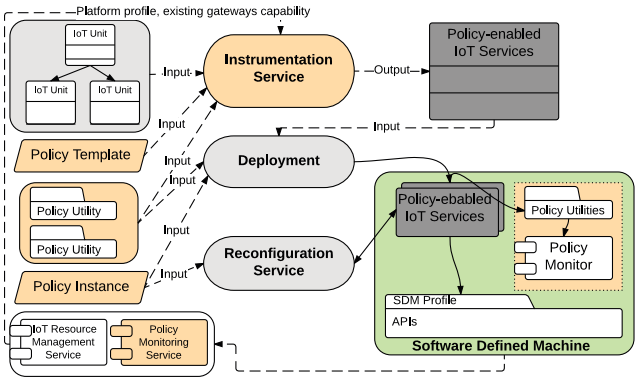

Fig. 2. P4SINC Framework Architecture

An IoT unit can be deployed and executed in different gateways/edge servers. As such, our proposed framework is designed to support for multiple gateways/edge servers architecture. This means that each instance of IoT services might be enforced by different policies for different gateways.

Fig. 2 depicts the overview of our proposed framework. IoT services and their units are stored in external repositories (e.g., Github) and marketplaces. An IoT service can be described in terms of a topology of IoT units. Before deploying IoT services, their IoT units are instrumented based on the *policy specification*, which describes types of policies that should be enforced at runtime. The policy specification is provided by our framework (see § II-C) and defined by the SDM owner.

Since there are different types of IoT units and policies, in our conceptual framework, we provide different types of utilities (see § II-D) to handle appropriate policy enforcement. When instrumenting IoT services, the instrumentation service will find suitable Policy Utilities for defined policy based on the specification. Depending on the situation, the instrumentation process will insert calls of utilities into the IoT units and modify the topology of IoT services to make sure that the topology also includes suitable utilities so that they can be deployed or activated for the IoT units at runtime.

After instrumentation, policy enforcement code is injected into IoT services, called policy-enabled IoT services. These policy-enabled IoT services and possible policy utilities can be deployed to IoT gateways as usual by an external *Deployment Service*. In our design, we use SALSA [8] for service deployment; IoT services, their units and other artifacts, such as required Policy Utilities, are described in dependency topologies using TOSCA.

Each IoT unit will be controlled by a local (inlined) reference monitor resided inside the unit. The local *Policy Monitor* maintains local policy specifications for each single unit and all units in the same SDM. Policy Monitor will communicate with a (global) *Policy Monitoring Service* which enforces the IoT services as a whole. The local monitor must be invoked when an IoT unit starts and can start, stop, suppress or replace an action in an IoT unit. In general, the monitor can stop the execution of the whole IoT service if the service violates a policy.

C. Execution Policy Specification

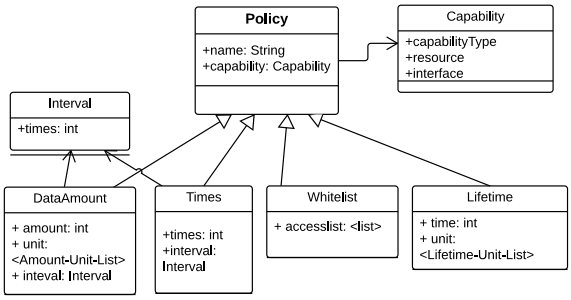

Fig. 3. Execution Policy Model

One goal of our framework is to provide a general and extensible policy specification that can be used (enforced) on different platforms represented through SDMs. To this end, we define a high level execution policy specification, which is based on the capability of SDM profile. Fig. 3 depicts our conceptual policy model for the specification. In this model, a policy is mapped with an each action, which is a capability provided by SDMs. There are several metrics that are associated with each policy. In this work, we consider the following metrics for policies: data amount (the amount of data that an IoT service can read or write), times (the number of times an action in an IoT service can be invoked), whitelist (a list of string values that are allowed to for a particular action), and lifetime (time to live of an IoT service). More metrics of execution policy can be extended. There are two parts in our policy specification: (1) a policy template that maps a capability in SDM to a low-level API call and its policy metrics in our policy model, and (2) a policy instance that IoT unit users can specify concrete policy values to be enforced at runtime. We use JSON to define the both specification as it can be extensible and platform-independent. A policy template example is illustrated in Listing 1. In this

Listing 1. A simple policy template example for data access through capability DataPoint.get.

```
"PolicyTemplate" :
 [{"action": "DataPoint.get",
   "policies": [{"DataAmount":
          {"description":"Limit the data amount access",
               "unit":"Bytes"} },
          {"Whitelist":
              {"description":"",
                    "unit":"string"}},
          {"Times":{"description":"",
                                "unit":"number",
                                "sub-metric":"interval"
          }}
         ]
    }]
```

example, we illustrate several execution policies with different metrics for a single action "DataPoint.read", which is mapped to specific low-level and platform-dependent APIs in a SDM profile. To perform the enforcement of execution policy, each item in a policy template must be provided with concrete values, forming a policy instance. A policy instance example of the above template is shown in Listing 2, which specifies a policy that limits an IoT service to read a certain device, and to read a maximum amount of data per day in a BTS. We leave formal specification with extensible metrics for future work.

Listing 2. A simple policy instance example that limits the data access.

```
{"policies": [{"action": "DataPoint.get",
               "metrics": [{"DataAmount":"100"} ,
                           {"Whitelist":["list1","list2"]}
                           {"Times":{"value":"2"}}
                          ]
              },
              {..}]
}
```

D. Policy Utilities

In runtime enforcement, to support runtime policy checks, there are policy states and values. For example, in a policy such as "limit the data read amount to 10MB", 10MB is the policy value. During the execution, the enforcement mechanism has to keep track of the amount of data read, which is a policy state. Whenever the data read event happens, the policy state will be checked with the policy value for violation. If the violation happens, the runtime checks might suppress the event, otherwise it might update the state.

Runtime policy enforcement using inlined reference monitors normally combines policy states and values (to check policy violation at runtime) normally by program variables within the application. This kind of approach is platform specific as the variables must be implemented in the application. To support the requirement of platform independence, we propose to separate the code from the policy states and values. Policy values provided from the user-specific policy and the states for the policy will be encoded by separate entities, called *software-defined policy utilities*. As our framework supports different gateways/sensors in various platforms, we need an instance of an utility for a particular platform. Moreover, our policy utilities also support reporting messages to cloud automatically for policy monitors.

We propose software-defined policy utilities that handle runtime policy checks for different policy metrics such as data

amount, times, execution lifetime, whitelist on a particular platform. The operations include checking policy violations and updating and transitioning policy states for later checks. In future work, we will develop a platform independent policy utility profile as a software-defined concept, which can be used for any SDM IoT infrastructure.

The set of policy utilities consists of two parts: in-service policy utilities and gateway policy manager. In-service policy utilities are implemented as services to control metrics such as amount, times, permissions and also control the execution of the corresponding IoT service, i.e., stop, suppress, truncate an action. When an IoT service starts, an in-service policy controller is invoked. This controller will establish a connection to a gateway policy manager to register the service and check policy at gateway level when needed.

E. Instrumentation Service

This service takes a policy template and instance in above specification, policy utility meta-data, an IoT service and its topology as inputs, and instrument the code of the IoT service to produce a new IoT service topology with instrumented IoT units. The key algorithm in this service includes finding an appropriate policy utility to inject to the IoT service and invoke an instrumentation module to perform the code transformation to embed the policy and utilities into the code of IoT service as a monitor so that corresponding actions in the IoT service can be controlled and the policy can be enforced at runtime.

Given an IoT service, SDM and policy utilities profile, and policy specification, the core of this proposed framework is to *instrument* the IoT service to inject the policy code into the service to transform it into a policy-enabled IoT service. The conceptual model of this process is depicted in Fig. 2. In our considered IoT scenarios, IoT services are developed and stored in the cloud and are ready to be executed on an SDM. Therefore, the source code of the service might not be available. The instrumentation service should deal with binary or intermediate languages such as bytecode to inject the policy checks. We adopt the aspect-oriented programming (AOP) paradigm [9] to implement this instrumentation process. AOP is a programming paradigm that allows the separation of cross-cutting concerns by adding additional code (advice) to a program without modifying the program code itself. To date, AOP is supported for almost all programming languages.

Within the Instrumentation Service, we have developed an instrumentation tool that has a transformation module to transform the high-level policy specification into a low-level AOP language. The transformation is based on (1) SDM profile, (2) Policy template, and (3) policy instance as described in the previous steps. After the transformation, the instrumentation tool will automatically invoke an appropriate utility (based on the description in the SDM profile) to perform the instrumentation to inject policy code into IoT units.

The deployment and execution of the policy-enabled IoT services are as the same as original IoT ones, as we embed the policy utilities and necessary runtime libraries for a specific platform into the IoT services. A policy instance is also provided for a particular policy-enabled IoT service at the deployment phase so that it can be accessible at runtime.

Listing 3. Pseudo code of runtime enforcementat at a capability execution point:

```
1 //policyutil is an instance of PolicyUtils for the IoT unit
2 //capability is the signature of the capability
3
4 foreach metric m within
5         policyutil.getMetrics(capability)} do {
6   //check if the metric is defined in the policy instance
7   if policyutil.exist(m, capability){
8      policyutil.enforce(m, capability);
9   }else skip;
10 proceed(); //execute normally
```

F. Runtime Enforcement

IoT units are deployed in different IoT gateways and each unit might be executed by different users. An IoT unit executed by a user needs user-specific policy inputs so that the unit can be monitored and enforced by the policy. User-specific policy inputs are expressed in a policy instance associated with a particular IoT unit instance. As depicted in Fig. 2, a policy instance, together with a policy-enabled IoT unit, is one of the inputs for the Deployment Service to deploy the secure IoT unit to an IoT gateway. When being executed, an inlined reference monitor in a policy-enabled IoT unit connects to the policy to enforce the defined policy.

1) Policy Instance: A policy instance contains user-specific policy value inputs expressed in a policy file associated with a policy-enabled IoT unit. When the unit is executed, the policy utility inlined in it will check the values in the policy instance to enforce the policy. Similar to the policy template introduced in Listing 1, a policy instance is encoded in a JSON format with a list of policy object, which contains a capability action and a list of policy metrics and values. Listing 2 illustrates a simple policy instance that limits the data access to the amount of 100 MB.

In each policy-enabled IoT unit, all the policy checks and enforcement code are implemented in a policy utility (`PolicyUtils`), which has been introduced in Section II-D. The code first checks if a metric has policy input value defined the policy instance and if so invoke the policy utility to enforce the value. The `enforce` function maintains and compares the runtime parameters versus the policy input values in the policy instance. If there is any policy violation, the execution will be suppressed, otherwise, it proceeds normally. The pseudo code in Listing 3 illustrates this enforcement process.

G. Extensibility

Our policy enforcement framework is extensible for new capabilities. When a gateway or edge computing system provides a new capability or instruction for IoT units to function, its corresponding SDM profile is updated. With such a new update, there is no modification in the Instrumentation Service. A policy template for an IoT unit with the new capability will have the same specification, only a new object is added to the template. Using the policy template, the Instrumentation Service can instrument the IoT unit regularly. The deployment phase is the same as before. A policy instance is provided

when an IoT unit is requested to be deployed and executed. The runtime enforcement mechanism is also the same as before.

The process of deployment on a new gateway is straight forward. Using a deployment service such as SALSA [8], each policy-enabled IoT unit embedded with the policy utilities, runtime libraries, and a policy instance is deployed as the same method and can be executed normally.

III. Implementation and Experimental Results

P4SINC is an open source: code and experimental examples are continuously updated at https://github.com/SINCConcept/P4SINC. We used an open source implementation of SDM for IoT gateways[1] APIs to evaluate our framework. These APIs have been implemented as a set of Java libraries providing programmatic access to sensors and actuators through a daemon process that directly interfaces the underlying hardware. Although we perform experiments on Java, our conceptual model is platform-independent. In this work, we leverage the AspectJ tool, an aspect-oriented tool for Java, to perform the instrumentation process. We note that this instrumentation process is also platform-independent as the Instrumentation Service can invoke a similar tool for a specific platform.

A. Implementation and Experimental Set Up

We have developed and reused a number of IoT units with various functionalities, including read data from several data points, implemented in the above-mentioned SDM prototype for IoT gateways. The applications are packed into `jar` files and deploy them on a gateway/edge computing system.

We utilize a Raspberry PI to emulate a gateway. In our experiment, when the user wants to request an IoT unit to be executed in the gateway, she specifies a policy instance and the link in the cloud of the IoT unit, and invoke our framework to perform the request. We have developed several execution policies representing the design of our policy specification presented previously including limiting data amount access, the number of times an action can be invoked and a whitelist of IDs that an IoT service can access.

In addition to the utilities to handle different metrics for execution policies, we have also developed a message communication functionality as a part of policy utilities. In this implementation, any messages from the policy enforcement will be posted to the cloud via a MQTT broker so that the IoT service provider can see the activities. We use an MQTT broker implemented with https://www.cloudmqtt.com/ in our emulation and the Google BigQuery for storing IoT data.

We have developed a web-based system implementing our Instrumentation Service. The system allows the users to provide an IoT service packed in a Java `jar` file, SDM profile and policy instance in JSON files. The output of this system is the instrumented `jar` with policy enforcement code and corresponding policy utilities and necessary runtime Java libraries. In future, we will extend this system as a cloud-based service that allows the users to compose policies to perform the instrumentation process on the cloud. This cloud-based Instrumentation Service can be easily combined with the Deployment Service to form a complete eco-system for execution policy enforcement.

[1]https://github.com/tuwiendsg/SoftwareDefinedGateways

B. Evaluation

We have evaluated our implementation by testing original IoT services and instrumented ones (policy-enabled IoT services). We ran the original and then policy-enforced IoT services with some particular policy instances by deploying and executing them in different systems including a virtual machine and a Raspberry PI. As mentioned earlier, the policy checks at runtime have been logged on the MQTT broker, through a function of policy utilities. We have modified the original IoT services to violate the policies and we have verified that the violation messages are logged on the MQTT server and the violated actions are suppressed from execution (while the IoT service is still running).

To illustrate how this instrumentation module and the proposed framework work in practice, let us reconsider the motivated scenario mentioned in the introduction, where the Telco company allows to deploy an IoT service from the outsourced HVAC company run on Telco's SDMs. Assume that the IoT service contains an unknown vulnerability that allows attackers to inject malicious code to e.g., steal data and to abuse the resource by reading too much data. As discussed earlier, standard security mechanisms cannot prevent the attack because it resides inside the service (and its units) and unknown. In our proposed framework, before an IoT service can be deployed, the IoT infrastructure provider defines a policy template and some policy instances e.g., limit the data read, and then requests the instrumentation service to weave the policy code to the IoT units. There are some policy values specified by the user when she executes the service, in this example, the destination of data to be sent (which is implemented as a whitelist policy utility). Thus, data read and send behaviors are monitored at runtime by the security checks and policy utilities injected by the Instrumentation Service so that they comply with the defined policies. As a result, even though this cannot prevent the code injection due to the unknown vulnerability, the monitor can ensure that no data leak nor resource abuse happen thanks to the policy enforcement.

IV. Related Work

Runtime policy enforcement techniques have been explored widely in the literature. One such technique is to use a *reference monitor* that mediates the interactions of the application on the host environment to enforce runtime policies, e.g., [10], [11], [12]. While the approaches can enforce fine-grained security policies, they are not applicable to the dynamic and complex execution environments of IoT infrastructures as we are investigating in this research. In [13], an enforcement of generic security policy is introduced. However, the policies only support on input/output of a program therefore it is not applicable for IoT infrastructures. Some frameworks

allow code running in mobile devices/gateways but do not support runtime policy enforcement. For example, ThinkAir [14] allows mobile code to be executed in its environment, and provides a profiler to monitor execution of mobile code but no generic execution policy has been discussed and presented. In this work, we aim to provide a generic policy model that can enforce execution, security/privacy or service contract policies. Several present techniques allowing policies and code integrated within a program is presented in e.g., [15]. Their solutions are generic so in principle such techniques can be applied for IoT software components. However, these techniques are not suitable as IoT applications that might be used for different IoT infrastructures.

Policy Languages: Several works introduce policy languages and related tools that can enforce security policies for software, however, the languages are platform-specific. Moreover, these languages do not support IoT infrastructures. Standard policy languages such as WS-SecurityPolicy, Role-Based Access Control, or Attribute-Based Access Control only support coarse-grained access control policies, which are inadequate to address application-level attacks and generic execution policies.

IoT Security: As the IoT has been deployed widely in practice, IoT security is a big concern in society [16]. Big industry players such as ARM, Symantec are cooperating to build standard for the IoT. Cisco has proposed a security framework for IoT infrastructures [17]. In addition, several works in the literature (e.g., [18], [19], [20]) proposed security solutions for IoT infrastructures. However, these proposals focus on security and privacy for a specific IoT platform. In P4SINC, we investigated on enforcing generic execution policies that include security and privacy, and also service contract management crossing multiple IoT platforms.

V. Conclusions and Future Work

Generic execution policies are strongly needed for enabling IoT services deployed in IoT infrastructures. We have presented the P4SINC framework which includes a generic execution policy specification and a set of tools to enable the instrumentation and enforcement of policies through the deployment and execution of IoT services. Initial results have shown that our framework can help to enable different types of policies typically seen in security and service contract management. Our experiments and examples are currently conducted in a small-scale that need to be improved. We are currently concentrating on the implementation of the proposed framework for various complex IoT services and infrastructures.

Acknowledgments: This work was partially supported by the University of Dayton Research Council Seed Grant and the European Commission in terms of the U-Test H2020 project (H2020-ICT-2014-1 #645463). We would like to thank Vishal Panwar at University of Dayton for implementing the Instrumentation Service.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the first edition of the MCC Workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[3] "Mobile-Edge Computing – Introductory Technical White Paper," https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf, September 2014.

[4] P. Persson and O. Angelsmark, "Calvin – Merging Cloud and IoT," *Procedia Computer Science*, vol. 52, pp. 210 – 217, 2015, the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

[5] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.

[6] H.-L. Truong and S. Dustdar, "Principles for Engineering IoT Cloud Systems," *Cloud Computing, IEEE*, vol. 2, no. 2, pp. 68–76, Mar 2015.

[7] A. Brring, S. Schmid, C. K. Schindhelm, A. Khelil, S. Kbisch, D. Kramer, D. L. Phuoc, J. Mitic, D. Anicic, and E. Teniente, "Enabling IoT Ecosystems through Platform Interoperability," *IEEE Software*, vol. 34, no. 1, pp. 54–61, Jan 2017.

[8] D.-H. Le, H.-L. Truong, G. Copil, S. Nastic, and S. Dustdar, "SALSA: A Framework for Dynamic Configuration of Cloud Services," in *Proceedings of IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom), 2014*, Dec 2014, pp. 146–153.

[9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," in *ECOOP*, 1997, pp. 220–242.

[10] R. Joiner, T. Reps, S. Jha, M. Dhawan, and V. Ganapathy, "Efficient runtime-enforcement techniques for policy weaving," in *Proceedings of Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, 2014, pp. 224–234.

[11] K. Havelund and G. Rosu, "Efficient monitoring of safety properties," *Int. J. Softw. Tools Technol. Transf.*, vol. 6, no. 2, pp. 158–173, 2004.

[12] P. H. Phung and D. Sands, "Security Policy Enforcement in the OSGi Framework Using Aspect-Oriented Programming," in *Proceedings of 32nd Annual IEEE International Computer Software and Applications Conference*, July 2008, pp. 1076–1082.

[13] M. Ngo, F. Massacci, D. Milushev, and F. Piessens, "Runtime Enforcement of Security Policies on Black Box Reactive Programs," in *Proceedings of POPL 2015*. ACM, 2015, pp. 43–54.

[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 945–953.

[15] T. H. Austin, J. Yang, C. Flanagan, and A. Solar-Lezama, "Faceted execution of policy-agnostic programs," in *Proceedings of the Eighth ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS '13)*. ACM, 2013, pp. 15–26.

[16] Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, and S. Shieh, "IoT Security: Ongoing Challenges and Research Opportunities," in *Proceedings of The 7th IEEE International Conference on Service-Oriented Computing and Applications*, Nov 2014, pp. 230–234.

[17] "Cisco Security Research & Operations. Securing the Internet of Things: A Proposed Framework," Online: http://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html, July 2016.

[18] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the Internet of Things," *Mathematical and Computer Modelling*, vol. 58, no. 56, pp. 1189 – 1205, 2013.

[19] E. Bertino, K.-K. R. Choo, D. Georgakopolous, and S. Nepal, "Internet of Things (IoT): Smart and Secure Service Delivery," *ACM Trans. Internet Technology*, vol. 16, no. 4, pp. 22:1–22:7, Dec. 2016.

[20] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms," in *Proceedings of The Network and Distributed System Security Symposium 2017 (NDSS'17)*, 2017.