Protecting the Internet of Things with Security-by-Contract and Fog Computing

Alberto Giaretta
Centre for Applied Autonomous Sensor Systems (AASS)
Örebro University, Sweden
alberto.giaretta@oru.se

Nicola Dragoni
DTU Compute
Technical University of Denmark, Denmark
and *AASS, Örebro University, Sweden*
ndra@dtu.dk

Fabio Massacci
Department of Information Sciences and Engineering
University of Trento, Italy
fabio.massacci@unitn.it

***Abstract*—Nowadays, the Internet of Things (IoT) is a consolidated reality. Smart homes are equipped with a growing number of IoT devices that capture more and more information about human beings lives. However, manufacturers paid little or no attention to security, so that various challenges are still in place. In this paper, we propose a novel approach to secure IoT systems that combines the concept of Security-by-Contract (S×C) with the Fog computing distributed paradigm. We define the pillars of our approach, namely the notions of IoT device contract, Fog node policy and contract-policy matching, the respective life-cycles, and the resulting S×C workflow. To better understand all the concepts of the S×C framework, and highlight its practical feasibility, we use a running case study based on a context-aware system deployed in a real smart home.**

***Index Terms*—security-by-contract, Fog computing, IoT**

I. Introduction

Accordingly to Gartner Hype Cycle for Emerging Technologies [1], Internet of Things (IoT) surpassed the so-called peak of disillusion, headed to an established role within society. But all the problems are far from being solved and, the more pervasive the IoT becomes, the harder it is to manage. In particular, IoT security is one of the biggest cybersecurity challenges, and one of its most embarrassing failures [2]. Traditional cybersecurity solutions have proven to be ineffective for IoT due to a number of technical and operational challenges. First, IoT devices are highly heterogeneous, with huge differences across tiers, languages, OSes, and networks. Also, the IoT lacks a common security framework, and standards are still not settled. Often times security is not a manufacturers' (nor IT admins') core competency, and may not be even considered part of the IoT product development process.

A typical smart home can easily use dozens of IoT sensors, whereas industrial applications can scale up to hundreds of IoT devices. This introduces a number of problems, such as maintaining and monitoring the IoT devices, allowing and disallowing communication protocols, and overseeing what kind of information can be shared under defined conditions.

A context-aware (CA) system (e.g., a smart home) is responsible for collecting raw data from deployed sensors, and inferring information from combination of simple observations. If a person is moving in the kitchen and the room temperature is rising (with respect to the other rooms temperature), the CA is capable of inferring that a resident is cooking and reacting accordingly. Taking the cooking example, as soon as the CA infers that the resident is cooking, it might be programmed to power on the ventilation. It is clear to see that prompt reactions require low latency, sometimes hard to achieve by means of Cloud computing. A partial answer to this, Fog computing can act as a middle layer between IoT devices and Cloud computing. A CA system can appoint Fog nodes to carry on real-time computation, and delegate more cumbersome tasks (e.g., processing historical data) to the Cloud.

Following the CA system example, security is often neglected in IoT smart homes: even though the whole network might be well protected, single devices might be vulnerable. For instance, an attacker might get access to a few of them and, through an external context reasoner, infer important information about the residents. Stuffed toys, smart thermostats, networked light bulbs, baby monitors are just a few examples of IoT devices that have been hacked in the recent years, representing a threat to the security, privacy and in some case safety of the residents [2]. Moreover, such IoT devices might be used to carry on DDoS attacks, as already happened in the past with Mirai and many other DDoS-capable malwares [3], [4]. Therefore, we need a disciplined way to gather sensors data and govern communication flows, according to devices specific requirements.

Contributions of the Paper: In this paper, we propose a novel approach to secure IoT devices by combinationf of two key concepts: Security-by-Contract (S×C) and Fog computing. The S×C framework has been applied to secure software and devices in a number of domains, mainly mobile applications [5], [6] and multi-application smart cards [7]. We claim that S×C, combined with Fog computing, can be adapted and applied to protect IoT systems. A running case study based on a Fog-based smart home is used throughout the paper to illustrate the main concepts of our proposal, as well as its feasibility in a real-world setting.

Paper Outline: Section II presents the case study used throughout the paper, to explain the key concepts of the S×C approach. Sections III briefly introduces Fog computing. Section IV describes the key concepts of the S×C approach. Section V briefly summarizes the relevant literature. Finally, Section VI draws conclusions.

978-1-5386-4980-0/19/$31.00 ©2019 IEEE

II. Running Case Study: E-care@home

E-care@Home is a Swedish interdisciplinary distributed research environment that pulls together competences in Artificial Intelligence, Semantic Web, IoT and Sensors for Health [8]. Ängen, a real smart home composed by several IoT devices, has been created within the E-care@home initiative (Fig. 1 shows Ängen layout). In each room, motion sensors are present. Bed, sofa, and chairs are all equipped with binary pressure sensors to detect whether someone is sitting or not. In the kitchen, the oven has an on/off sensor. A smart light bulb and a smart voice assistant are installed in the living room. Last, every sensor communicates with a context reasoner, which runs on a laptop.

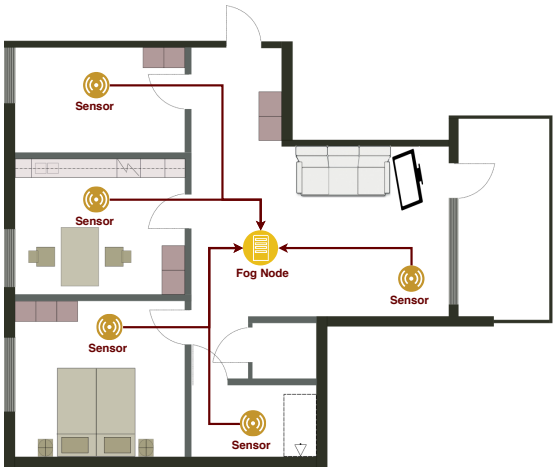

Fig. 1. Ängen context-aware smart home, equipped with various sensors, like motion and pressure ones. Currently, a generic laptop collects the ground truth, and infers information through a reasoner. We envision a setup where a Fog node substitutes the laptop, and takes on security tasks.

Example 1. Let us assume that our goal is to collect and analyse both real-time and historical sensor data, in order to react to sensors inputs (e.g., a motion sensor that triggers a smart lightbulb), learn residents' behavioural patterns, and detect sensors failures. Real-time reactions require that computation is performed as close as possible to the devices, whereas more cumbersome computation (e.g., analysis of behavioural patterns) can be offloaded on the Cloud.

Example 2. Let us assume that a pressure sensor exhibits an odd behaviour, due to a malicious attacker or a malfunctioning. We want to be capable to detect the problem and manipulate the policy, in order to temporarily rule out this sensor.

III. Fog Computing for IoT

With the advent of IoT, the main question was how computationally limited devices could undertake their tasks. Cloud computing was the natural answer, since it allows users and developers to be (almost) resource-agnostic, through its elastic architecture and pay-as-you-go billing. However, this reduced complexity has a cost, in terms of unpredictable latency and uncertain storage location, with the latter particularly problematic with security-related data.

Fog computing is a paradigm proposed to extend Cloud computing, providing a virtualized middle layer that sits between latency-sensitive applications and data storage centres. Low-latency, mobility, scalability, and heterogeneity are some of the important characteristics that Fog computing proposes to provide to the end-users [9]. These characteristics make the Fog suitable for supporting IoT applications, such as connected vehicles, smart homes, smart cities, and wireless sensors and actuators networks [10] in general.

A simple way of introducing Fog computing in a network is by installing a Fog node. The Fog node can be a generic machine appointed to centralize sensors data, undertake time-bound tasks itself, offloading more cumbersome (and not strictly time-bounded) tasks to the Cloud, and in general serving as a gatekeeper of data traffic. Being a trustworthy node at the edge of the network, the Fog node can play useful security roles. As an example, in S×C we use the Fog node to store the network security policy (Section IV-B), and to evaluate if IoT devices comply with such policy (Section IV-C). Besides, at the Fog computing layer we have enough computing power to enable real-time complex policy enforcement techniques (e.g., policy enforcement based on traffic monitoring).

Last, in order to meet the running case study requirements, we move the E-care@home context reasoner from the laptop to the Fog node, as shown in Fig. 1. As aforementioned, this enables fluid allocation of computing tasks to the Fog and the Cloud, as well as efficient verification of IoT devices compliance to the security policy. Given that S×C policy enforcement strategies might rely upon devices data flow (as we discuss in Section IV-C), managing policies and data at the same level is a reasonable solution.

IV. Security-by-Contract (S×C)

The notion of Security-by-Contract was first proposed to secure mobile applications [5], [6] and then successfully applied to other domains, such as Web services [11] and multi-application smart cards [7]. We claim that S×C, combined with Fog computing, can protect IoT systems as well.

Following the S×C approach, in order to be able to decide whether an IoT device is secure or not in a given context (such as a smart home), we need two pillars. First, we need a **contract** that describes which resources/protocols are necessary for the device to operate, and how the device uses them; second, we need a **policy** that declares what actions are allowed or prohibited within that specific context, and what services/resources the IoT devices needs to correctly work. In this section, we give a definition of contract and policy, and further clarify them through some examples.

Definition 1 (Contract)**.** A contract is a formal complete behavioural specification of the security relevant actions of an IoT device, including the list of services and resources that the IoT device needs to operate.

A contract is *formal* because it gives a rigorous and unambiguous specification of the device behavior (semantics),

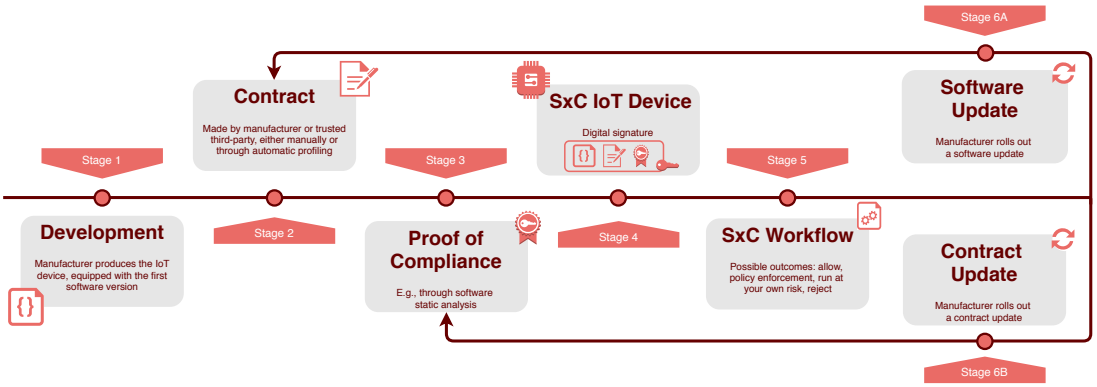

Fig. 2. S×C contract life-cycle. In Stage 4, the manufacturer digitally signs all the components, achieving software validation and non-repudiability.

for instance by means of a formal language (such as ConSpec [12]). Also, a contract is *complete* because it specifies all the security relevant actions of the IoT device. For adoption reasons, we plan to formalize a translation from S×C to MUD contracts. One of the main problems of proposals similar to ours, is that it requires adoption from manufacturers. Therefore, the capability of being MUD-compatible would enable to seamlessly integrate legacy IoT devices with S×C.

Example 3. Continuing with the running case study, contracts could contain the following rules:

- The smart voice assistant requires access to a specific set of external Internet addresses.
- The smart light bulb requires to communicate with the voice assistant, to enable voice commands.
- The smart light bulb allows communication with the manufacturer smartphone app, whether the smartphone is inside or outside the network.
- The oven sensor does not communicate over the Internet.

Definition 2 (Policy)**.** A policy is a formal and complete specification of the acceptable behaviour of IoT devices running under a Fog node's control, including the list of services and resources that the IoT devices need to operate.

Example 4. The policy of the Ängen smart home, installed on the Fog node, could include some of the following rules:

- The motion sensors can transmit outside the LAN (for example, residents might want to use them as home alarms while they are not at home).
- The home door needs a *wifidirect* service to get locked/unlocked.
- The vocal assistant cannot communicate over the Internet during night time.
- The CA system is deactivated if none is at home (i.e., when no motion nor pressure sensors are active).

In the rest of the section we describe how the above concepts of contract and policy are combined with Fog computing. In particular, we describe the S×C contract life-cycle (Section IV-A), the S×C policy life-cycle (Section IV-B), and the resulting overall S×C life-cycle (Section IV-C).

A. S×C Contract Life-Cycle

The S×C contract life-cycle is illustrated in Fig. 2. At Stage 1, the manufacturer develops the IoT device and equips it with a contract (Stage 2) which binds the device to a predefined security behaviour. Source code at hand, the manufacturer can easily list all the requirements necessary for its device to work correctly. Alternatively, the manufacturer could automatically create contracts through an external traffic profiler. Taking MUD as an example (briefly described in Section V), researchers have shown how to automatically extract main functionalities and security requirements of IoT devices [13].

At Stage 3, the manufacturer equips the device with a *Proof-of-Compliance (PoC)*, which binds the IoT software to the contract. This is fundamental in S×C, in order to have a formal proof that the device behaves exactly as described in its contract. This can be easily done, as an example, through static verification of the on-board software. At Stage 4, everything is signed with a private key, in order to achieve non-repudiation. It is important to stress that binding together IoT software, contract, and PoC with a digital signature, *S×C provides a semantics for digital signatures on IoT code*. Indeed, in S×C we trust an IoT device only according to the security behaviour described in its contract, not just because a trusted party signed it (as in traditional digital signatures). In other words, we want trustworthy IoT devices, not trusted ones.

The contract is then matched with the Fog node security policy (Stage 5), according to the S×C workflow (Fig. 4). Depending on the matching result, several situations (with different security implications) may arise. These situations are described in Section IV-C.

Whenever the software is updated (Stage 6A), the manufacturer might need to update also the contract, thus the PoC, resulting in a new S×C IoT device (i.e., cycle restarts from Stage 2). Similarly, if the contract is updated (Stage 6B) the new contract must be checked against the existing software (cycle restarts from Stage 3). Independently from the type of update, a new contract/policy matching must be performed (Stage 5, details in Section IV-C). In all this, we assume that the Fog node can distinguish data from code, as well as software updates from contract updates.

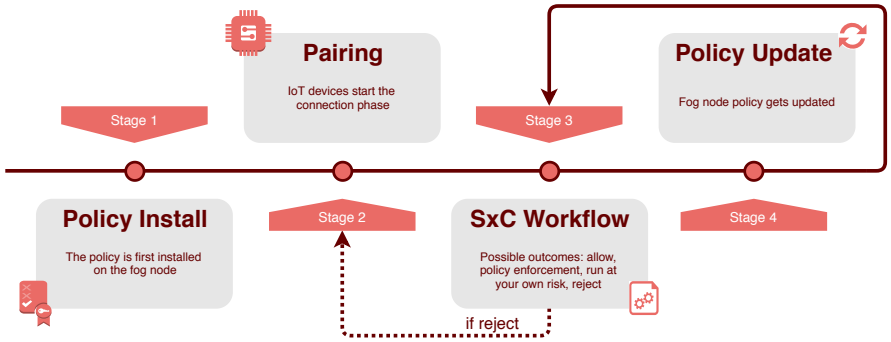

Fig. 3. S×C policy life-cycle. Every time the administrator updates the policy (he could tighten, or loosen, it up), all the devices must be re-evaluated.

Example 5. A manufacturer launches on the market a new generation of smart lightbulbs.

- Stage 1: The manufacturer produces the first device version. The lightbulbs require to be accessible through SSH, to enable admin remote configuration.
- Stage 2: Based on the previous information, known to the manufacturer, it is easy to create a contract that states the necessity of enabling SSH connection.
- Stage 3: The manufacturer also provides a PoC that confirms that the contract matches the on-board software.
- Stage 4: At this stage, we have fully S×C-compliant lightbulbs, which carry the operating software, a contract, and a valid PoC.
- Stage 5: The device is verified against the security policy, and accepted in the network.
- Stage 6A: Let us assume that the manufacturer implements a new feature, which enables the lightbulbs to react to vocal commands. However, the lightbulbs require to communicate over MQTT (a lightweight protocol) with a voice assistant. Therefore, the contract has to change and the PoC needs to be updated, according to the new software. Then, the life-cycle starts again from Stage 2.
- Stage 6B: Alternatively to Stage 6A, if the manufacturer implements a new contract the PoC needs to be updated, and the life-cycle starts again from Stage 3.

B. S×C Policy Life-Cycle

Whether a network is dynamic or not, system policies can change; for example, the administrator might decide to tighten (or loosen) the security measures. Regardless of the reasons, previously acceptable devices might be incompatible with the new policies. This fuels the necessity of re-evaluating the compliance of all devices. As shown in Fig. 3, we follow a simple life-cycle. Whenever the administrator changes the security policy (Stage 4), all S×C contracts need to be re-evaluated for compliance with the new policy (Stage 3). Depending on the contract/policy matching result (see Section IV-C), there can be different situations. The devices whose contracts are compliant with the new policy are allowed to run. For devices whose contract is not compliant with the new policy, there are basically three options: (1) the new policy is enforced, either in the Fog node or directly in the device itself; (2) the device is disconnected from the network and required to restart from the pairing phase (Stage 2); (3) the administrator can decide to run the device anyway, even if not secure. These options are discussed in Section IV-C and illustrated in Fig.4.

Example 6. The administrator of the Ängen smart home has to install and configure a security policy on the Fog node.

- Stage 1: The administrator configures a first policy where he allows both SSH and MQTT protocols.
- Stage 2: The smart lightbulb starts the pairing phase.
- Stage 3: The lightbulb exhibits a contract, which lists SSH and MQTT connections as requirements. If it is policy-compliant, the device pairs successfully; if not, the outcome varies, depending on the result of the S×C workflow. If rejected, it has to retry the pairing (Stage 2).
- Stage 4: Let us assume that, after some time, the administrator updates the policy and prohibits SSH connections. All devices have to pass judgement again, so the whole process restarts from Stage 3. In this case, the lightbulb cannot connect directly anymore. Therefore, it has to undergo the S×C workflow again, and the outcome entirely depends on the configuration chosen by the administrator.

C. S×C Workflow

Once described contract and policy, and their respective life cycles, we put everything together and discuss how S×C can secure IoT devices. Fig. 4 shows the resulting S×C workflow for IoT devices. If a device is S×C compliant (Stage 3 of the S×C contract life-cycle, Section IV-A) and the PoC is valid, we have to check if the contract is compliant with the Fog node policy. This is a key step in the S×C framework known as *contract-policy matching* problem: given a contract exhibited by an IoT device, and a policy specified in the Fog node, is the contract compliant with the policy? Intuitively, matching should succeed if, and only if, the S×C device executes accordingly to its contract, and its behaviour complies with the Fog node policies.

Definition 3 (Contract-Policy Matching)**.** Given a S×C IoT device connected to a Fog node, matching should succeed if, and only if, by running the device every trace satisfying the device contract also satisfies the Fog node policy.

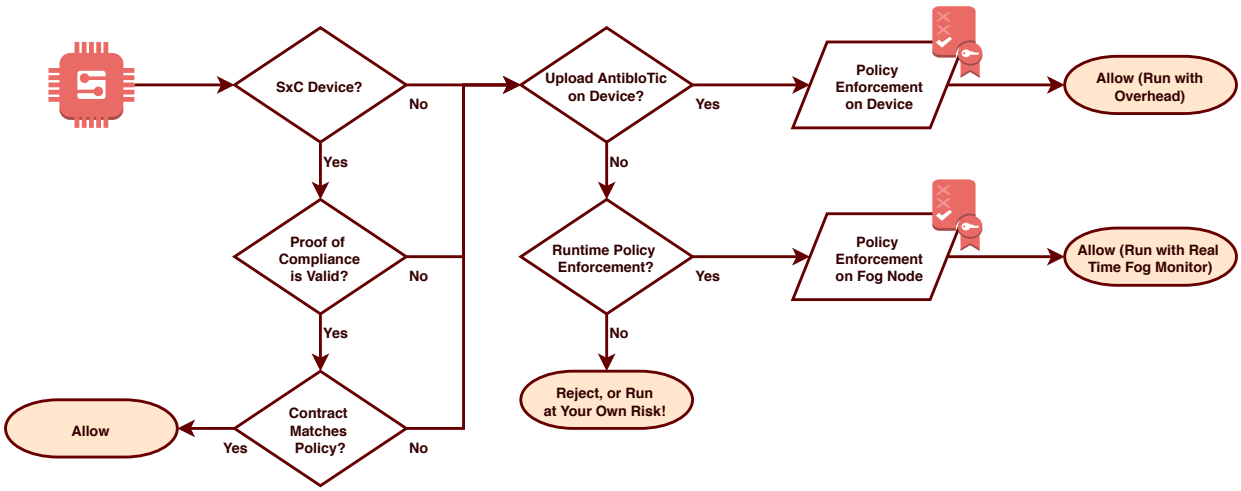

Fig. 4. S×C workflow. The contract of a S×C device must match the security policy of the network it is joining. If not, the device can be rejected, run as is (no security), or run with enforcement of the Fog node security policy. The enforcement can happen either in the Fog node, through traffic monitoring strategies, or directly in the device, for instance through AntibIoTic [14].

With *trace* we mean a sequence of actions that can be potentially performed by the device. A contract matches a policy if all the possible actions that can be performed by the device, as specified by the contract, belong to the set of allowed actions specified by the Fog node policy.

An example of developed S×C contract/policy matching algorithms for mobile apps can be found in [15]. In case the matching succeeds, the device is allowed to operate. Instead, if the device exposes behaviours prohibited by the policy, the matching fails and countermeasures should be taken. The same applies if the PoC is corrupted or invalid, or if the policy within the Fog node changes. Therefore, we need to provide the administrator a way to enforce policies, in case it is necessary. We envision 4 strategies to deal with this problem.

First, the Fog node can enforce the policies on the untrustworthy device through AntibIoTic, an anti-malware designed to live monitor (and eventually sanitize) the hosting device [14]. AntibIoTic is currently being refactored to work in a Fog-based network; in particular, an AntibIoTic-equipped device hosts a client that interacts with the Command-&-Control (C&C) server, hosted on the Fog-node. Therefore, the policy is enforced locally on the device.

Second, in case AntibIoTic cannot be installed on the IoT device, the policy can be enforced on the Fog node, and the correct execution verified through runtime monitoring techniques, such as network profiling and machine learning techniques. Potentially, the Fog node could sandbox the device, isolating it from all the other devices, and analyse its behaviour. But this might not be enough, as some malwares exhibited the capability of understanding whether they are running in a sandbox or not, and change behaviour accordingly[1].

Third, the device can be rejected and impeded from accessing the network. Fourth, and last option, the device can be allowed as is, at the administrator's own risk.

Example 7. Let us assume that the lightbulb exhibits a valid contract and a valid PoC, but its contract does not match the ongoing network policies. The administrator enabled both the AntibIoTic and the runtime monitoring approaches, but the smart lightbulb does not have enough memory to host the AntibIoTic client. Therefore, the Fog node launches a runtime monitoring process enforcing the node policy, and allows the lightbulb to join the network.

V. Related Work

In recent years, IoT security has been largely discussed. Matheu-Garca et al. [16] highlighted that the manufacturers should be included in the loop, in order to create more resilient devices. The authors propose a certification methodology that delivers a measurable evaluation of IoT devices security, as well as an automatic security assessment. Moreover, the authors noted the lack of a dedicated IoT vulnerability database, which would enable better automatic security tests.

Other researchers proposed Fog-based policy enforcement approaches to solve different problems in the IoT world, for example ensuring data privacy [17] and providing secure resource orchestration in Fog computing [18]. Similarly to our work, they share the necessity of enforcing policies at the Fog layer, over devices that might not be compliant by-design.

Cisco Manufacturer Usage Descriptions (MUD) [19] shares a number of common traits with our proposal. First and foremost, they envision a file that acts as a contract and states the device requirements. Second, the MUD file is parsed by a special node within the network (the MUD server) which defines appropriate Access Control Lists (ACLs).

However, a MUD file is pretty restrictive, as it barely describes basic information like allowed protocols and reachable hosts. Also, each MUD-compliant device does not carry the contract itself, but a simple URI that points the MUD manager

[1] [Online, accessed 2018-12-08] https://www.wired.com/2017/05/accidental-kill-switch-slowed-fridays-massive-ransomware-attack/

to the online MUD file: lack of Internet access would entail the total incapability of joining a network. Moreover, it is unclear how the contracts should be parsed, verified, and treated with respect to a security policy. Here, the key S×C concept of contract/policy matching, as well as a way to enforce policies on untrustworthy devices, seems missing.

Other researchers built on MUD. Hamza et al. [20] tried to undertake the problem of enforcing policies by means of combination with a Software Defined Network (SDN). In this context, the authors produced a translation from MUD policies to routing rules, aiming to implement the result into network switches. Again, they also noted that flow-based rules can be a first step to identify volumetric attacks, but they are not perfect. In particular, if such attacks happen on intended ports, MUD alone cannot be enough to identify the ongoing attack.

Another work proposed an automatic process mainly (but not exclusively) targeted to manufacturers, to extract a MUD file from an IoT device traffic trace. Moreover, they sketched the necessity of formally matching MUD files with network security policies [13].

VI. Conclusion

In this paper, we have proposed a novel approach to secure IoT systems based on the concept of Security-by-Contract. We claim that S×C, combined with Fog computing, can be the right answer to protect IoT devices. In particular, we have defined and applied the pillar concepts of the S×C framework to IoT. Moreover, we have discussed the eventuality that a device is not S×C-compliant or incompatible with the Fog node security policy, showing which enforcing strategies can be applied in this case. A running case study based on a Fog-enabled smart home has been used throughout the paper in order to illustrate the main concepts of our proposal, as well as its feasibility in a real-world setting.

In the future works, we plan to further expand on this paper and develop all the building blocks necessary to achieve our vision. In particular, we will define form and structure for S×C contracts, and we will develop an automatic tool for translating MUD contracts to S×C contracts. The reason behind this, is that we have considered the difficulties of promoting our paradigm in the manufacturers' world. As a consequence, we believe that enabling to derive S×C contracts from MUD ones would be enormously helpful, since this would enable to deploy a local S×C framework while using a pre-existing standard. Last, one of the next steps is the definition of the contract/policy matching algorithms, and the implementation of such algorithms on a fog node.

References

[1] "5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018," https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/, accessed: 2018-10-30.

[2] N. Dragoni, A. Giaretta, and M. Mazzara, "The Internet of Hackable Things," in *Proceedings of 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Springer, 2017, pp. 129–140.

[3] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "Analysis of DDoS-capable IoT Malwares," in *Federated Conference on Computer Science and Information Systems*. IEEE, 2017, pp. 807–816.

[4] ——, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation," *Security and Communication Networks*, vol. 2018, pp. 1–30, 2018.

[5] N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan, "Security-by-Contract: Toward a Semantics for Digital Signatures on Mobile Code," in *Proceedings of Public Key Infrastructure (PKI'07)*, J. Lopez, P. Samarati, and J. L. Ferrer, Eds. Springer, 2007, pp. 297–312.

[6] N. Dragoni, F. Massacci, T. Walter, and C. Schaefer, "What the Heck is this Application Doing? A Security-by-Contract Architecture for Pervasive Services," *Computers & Security*, vol. 28, no. 7, pp. 566 – 577, 2009.

[7] N. Dragoni, O. Gadyatskaya, and F. Massacci, "Supporting Applications' Evolution in Multi-Application Smart Cards by Security-by-Contract," in *Proceedings of the 4th Workshop in Information Security Theory and Practices (WISTP 2010)*. Springer LNCS, 2010.

[8] M. Alirezaie, J. Renoux, U. Kckemann, A. Kristoffersson, L. Karlsson, E. Blomqvist, N. Tsiftes, T. Voigt, and A. Loutfi, "An Ontology-based Context-aware System for Smart Homes: E-care@home," *Sensors*, vol. 17, no. 7, 2017.

[9] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis and C. Dobre, Eds. Cham: Springer International Publishing, 2014, pp. 169–186.

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the MCC Workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[11] N. Dragoni and F. Massacci, "Security-by-Contract for Web Services," in *Proceedings of the 2007 ACM Workshop on Secure Web Services*, ser. SWS '07. New York, NY, USA: ACM, 2007, pp. 90–98.

[12] I. Aktug and K. Naliuka, "ConSpec - A Formal Language for Policy Specification," *Science of Computer Programming*, vol. 74, no. 1, pp. 2 – 12, 2008.

[13] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman, "Clear As MUD: Generating, Validating and Applying IoT Behavioral Profiles," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*. ACM, 2018, pp. 8–14.

[14] M. De Donno, N. Dragoni, A. Giaretta, and M. Mazzara, "AntibIoTic: Protecting IoT Devices Against DDoS Attacks," in *Proceedings of 5th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, S. Litvinov, A. Messina, A. Sillitti, and G. Succi, Eds. Cham: Springer International Publishing, 2018, pp. 59–72.

[15] N. Bielova, N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan, "Matching in Security-By-Contract for Mobile Code," *Journal of Logic and Algebraic Programming*, vol. 78, pp. 340–358, 2009.

[16] S. N. Matheu-Garca, J. L. Hernndez-Ramos, A. F. Skarmeta, and G. Baldini, "Risk-based automated assessment and testing for the cybersecurity certification and labelling of iot devices," *Computer Standards & Interfaces*, vol. 62, pp. 64 – 83, 2019.

[17] A. Al-Hasnawi, I. Mohammed, and A. Al-Gburi, "Performance evaluation of the policy enforcement fog module for protecting privacy of iot data," in *2018 IEEE International Conference on Electro/Information Technology (EIT)*, May 2018, pp. 0951–0957.

[18] C. Dsouza, G. Ahn, and M. Taguinod, "Policy-driven security management for fog computing: Preliminary framework and a case study," in *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IRI 2014)*, Aug 2014, pp. 16–23.

[19] E. Lear and B. Weis, "Slinging MUD: Manufacturer Usage Descriptions: How the Network Can Protect Things," in *International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)*. IEEE, 2016.

[20] A. Hamza, H. H. Gharakheili, and V. Sivaraman, "Combining MUD Policies with SDN for IoT Intrusion Detection," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*. ACM, 2018, pp. 1–7.