Security Automation for Cloud-Based Internet of Things Platforms

Mheni Merzouki
National Institute of Standards and Technology
Gaithersburg, Maryland, USA
mheni.merzouki@nist.gov

Charif Mahmoudi
Siemens Corporation
Princeton, New Jersey, USA
charif.mahmoudi@simens.com

Robert Bohn
National Institute of Standards and Technology
Gaithersburg, Maryland, USA
robert.bohn@nist.gov

Cihan Tunc
The University of Arizona,
Tucson, Arizona, USA
cihantunc@email.arizona.edu

***Abstract*— Internet of Things (IoT) is reshaping the way Cloud Service Providers (CSP) collect data from sensors. With billions of devices deployed around the world, CSP are providing platforms dedicated to IoT that provides advanced features for those devices. This paper focuses on the virtualization of IoT devices and the way security automation can be achieved for this emerging category of virtualization. We present the different approaches used to virtualize IoT, the fundamental difference between IoT virtualization against server virtualization, and our contribution in terms of providing a cloud-based automatic mechanism to secure IoT cloud platforms.**

Keywords—Internet of Things, IoT, Cloud Computing, Cyber-Security, Virtualization, Virtual Devices, Virtual Machines

I. INTRODUCTION

In 1999 Kevin Ashton coined the term Internet of Things (IoT) as a network connecting objects in the physical world to the Internet. Since then the deployment of IoT devices has continued to grow at overwhelming scales and has now spread to various industries [1]. IoT devices cover applications from basic sensors and actuators to sophisticated embedded systems implementing the Operational Technology (OT) for factories. Virtualization approaches were proposed initially in early 2000s with frameworks such as Trango and PikeOS implementing Type-1 hypervisors for embedded systems [2]. Back then, solutions such as Trango and PikeOS were designed to provide for embedded systems similar features as ESXi (ESX at that time) and Kernel-based Virtual Machine (KVM) provide for servers. However, because embedded systems have limited capabilities and resources, their embedded hypervisors are highly primitive and heterogenous or ad-hoc.

In this paper, we define the term virtual device (VD) as the virtualization (emulation) of an IoT device moving the physical devices from the physical IoT operations to the virtual environment/world so that we can have the IoT devices simulated/emulated, tested, and managed in a virtual environment. These devices consist of a large set including sensors (i.e., from temperature and light sensors to smart environment sensors) and actuators (i.e., from light switches to smart home locks), and so on. Even though this idea seems to be similar to the Virtual Machine (VM), there are multiple challenges for the IoT device virtualization since the IoT devices, in contrast to VMs, are very heterogeneous in nature and, thus, their virtualization is not straightforward. Also, these devices have limited physical resources; hence their security enforcement is a laborious process.

Cloud Computing (CC) has become the obvious choice for most businesses to host services as it uses virtualization techniques to enable convenient, on demand access to configurable computing resources (networks, storage, servers, applications, and services). Through virtualization, the virtual resources used in CC are rapidly provisioned and released with minimum management effort or service provider interaction [3]. This wave of adoption also benefits IoT as several CSPs started dedicated platforms for IoT. Amazon announced their IoT core platform in 2015, Microsoft Azure IoT in 2016, Google Cloud IoT Core in 2017, and Siemens MindSphere 2.0 in 2018. Since then, cloud providers have had a high interest in enabling new features for the IoT fleet.

With the phenomenal growth of CC and its fast adoption rate, many concerns are raised including the security of the information systems, the networks, and all other aspects that make up the cloud. CC introduces new classes of security concerns that were unaccounted for in the old information systems making some of the old security solutions less efficient [4]. Currently, when a user creates a VD in the cloud, no security controls are applied to the VD until it was completely provisioned, and often in a manual way.

CC solves a crucial resource management challenge by virtualizing resources into VMs and mapping the VMs to the physical servers, which simplifies the process of resource allocation to the users [5]. This process is also applied to VDs. However, due to the large pool of users, and the variety and dynamic character of their needs, scheduling was adopted as the mechanism to avoid resource wasting. Scheduling is the process of provisioning VMs on the cloud–based on the user requirements, scheduling solves the accommodation of the environment and reserves the needed resource until released by the user.

Current scheduling solutions for CC have impressive management mechanisms for CPU and memory considering the performance, cost, and power consumption. However, the scheduling solutions lack security management. Hence, to solve this problem we use OpenStack as a cloud platform [6]. OpenStack is a set of software tools for building and managing cloud computing platforms. Although there are many aspects of cyber security in information systems including CC, we only

consider the aspect of network security in this paper. The networking service of OpenStack is handled using the Modular Layer 2 (ML2) plug-in to utilize simultaneously the variety of layer 2 networking technologies; its main implementation is using OpenVSwitch (OVS). OpenStack delegates the network management to OpenDaylight (ODL) a Software-Defined Networking (SDN) controller that controls OVS using a library of drivers and plugins called networking-odl to integrate OpenStack Neutron API with ODL [7].

The main contribution of this paper can be summarized as achieving security automation by solving the security control enforcement during the resource scheduling process of the VDs, and examining the SDN capabilities to secure the communications IoT devices in cloud platforms. Hence, the paper is organized as follows: Section II reviews the different virtualization techniques, challenges, and approaches relevant for IoT cloud platforms. Section III presents the proposed architecture for securing VMs and VDs that is designed on top of OpenStack. Section IV describes our experimentation and shows how the CC scheduling component can be leveraged to implement the security controls automatically. Section V presents our measurements of the impact of this alteration to the scheduler on the system. Finally, in Section VI, we discuss the future efforts.

II. IoT VIRTUALIZATION AND CLOUD SECURITY

This section describes the state-of-the-art approaches that address the IoT virtualization challenges both from the perspective of the hypervisors that emulate the physical resources of the IoT device and the cloud platforms that manage the emulated instances and their exchanges with the physical devices. Moreover, this section introduces how the security controls are currently implemented on the cloud platforms.

A. *Embedded Hypervisors*

Although our approach is focused on the IoT Cloud Platforms, history of embedded hypervisors is relevant as the same techniques and constraints are used on the cloud.

Since embedded hypervisors are deployed on resource-restricted environments they are required to be extremely efficient while being very small size, also the small size of the embedded hypervisors enables the formal verification and can help guarantee a bug-free run-time environment, embedded hypervisors are also required to enable software to run in real-time and best effort modes. [8]

These properties might be required for the IoT cloud platform so that the VDs to maintain compatibility with the physical devices and support similar use cases.

B. *Cloud IoT platforms*

The VD implementations differ depending on the applications (e.g. industrial, data collections, …), features associated with a VD may or may not conform to the properties described in the previous subsection. The following features are examples of three implementation-specific VDs that are used in the industry and literature:

- **Device shadow**: Enables the user to keep a proxy on the cloud that keeps the state of the device that allows querying the device even when the device is offline [9].
- **Virtual device**: Creates a fully virtual device that might be used for test purposes to enable integration from online applications.
- **Digital twin**: A virtual representation of a device covering not only the state (unlike a shadow) but enabling a full emulation of the device including its technical specificities [10].

C. *Openstack nova Scheduler*

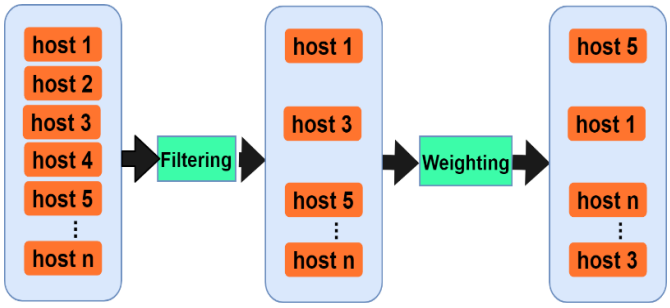

Figure 1: OpenStack scheduler filtering and weighting

The computing service (Nova) is one of the essential services of OpenStack as it interacts with the hypervisors on each node to provision and manage VMs and VDs and the resources allocated to them. Nova-Scheduler is a sub-service of Nova that is responsible for placing new VMs and VDs in the most suitable host available. when a request for a new VM or VD is generated, Nova-API receives the request and forwards it to Nova-Scheduler (see Figure 1) which runs a filtering algorithm that excludes the hosts that are unable to fulfil the requirements (e.g., CPU, RAM, disk) specified in the request. The second step is to sort the curated list of hosts using a weighting algorithm where weights are calculated for each host to select the best host from a list of valid hosts [11].

During the process of selection of the most suitable host, multiple resource filters can be considered by the Nova-Scheduler, such as CPU (computeFilter), RAM (ramFilter), disk (diskFilter), and even instance-based filters (e.g., differentHostFilter puts the requested VM on a different host). However, there are no filters considering the security of the VDs, VMs, and hosts.

D. *Security controls*

Security is a primary concern for all. The National Institute of Standards and Technology (NIST) published the SP 800-53 (Rev 4a) that defines security controls for Information Management Systems based on their Risk Assessment and contains a catalogue of security controls (SCs) that need to be implemented by the CSPs. For ease of use, the SCs are organized into three classes (management, operational, and technical) and eighteen families based on their security function [12]. Despite this categorization, organizations can find it challenging to select

the appropriate set of SCs that meet the security requirements of federal information systems. To overcome this challenge, security control baselines were introduced by the FIPS 199 standard [13]. A security control baseline is the minimum set of SCs recommended for an information system based on an attack's security impact on each of the Confidentiality, Integrity, Availability (CIA) of the system.

Three impact categories are introduced in FIPS 199 standard [13]:

- **Low impact**: A loss of C, I, or A will cause a degradation of service, but the organization will be able to perform its primary functions.
- **Moderate impact**: A loss of C, I, or A will cause severe degradation of service and disturb the effectiveness of the organization in performing its primary functions.
- **High impact**: A loss of C, I, or A will cause severe degradation of service to the extent that the organization is not able to perform the primary functions.

E. Cloud security controls enforcement

Security implementation, usually performed at the VM level, is not delegated to the cloud stack since it is considered application specific. However, with VDs, the implementation of the security controls cannot be pushed at the VD level because of the restricted and heterogeneous nature of the VDs. Unlike VMs, VDs might be primitive codes running minimum code to be compatible with the hardware device that the VD represents. Due to this fact, the existing VM level security control mechanisms are not suitable to the VDs.

One relevant option considered in this paper is the use of software defined networking (SDN) along with OpenStack networking to support the enforcement of those security controls at the cloud level. For this integration, ML2 driver and L3 plugin are used to enable communication of OpenStack networking component (Neutron) L2 and L3 resources API to OpenDayLight Backend. As Neutron uses OVS for underlying networking, it creates communication bridges called br-int, br-ext, and br-tun. These bridges enable communication between VMs and the external network at L2/L3. Two operation modes are supported by OVS: normal mode and SDN mode. In normal mode, OVS acts as any regular layer 2 switch (i.e. it detects MAC addresses from incoming frames and forwards the frames to the appropriate interface). In SDN mode a flow table is defined which contains flow rules that govern how the frames are forwarded or dropped.

III. ARCHITECTURE

The proposed architecture addresses two challenges related to the implementation and enforcement of the security controls for IoT cloud platforms. As illustrated in Figure 2, we propose the use of multiple variations of Qemu [14] to enable virtualization of IoT devices. We assume that the guest machines run using a different architecture from the bare metal host. Hence, we use Qemu to emulate the different architectures that

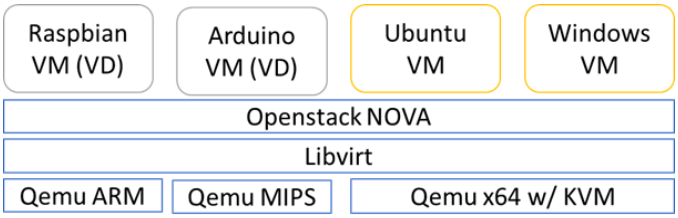

Figure 2 emulating IoT devices with the Qemu hypervisor

are used for IoT devices. The integration with OpenStack is transparent as Libvirt is used to provide these emulation services the same way as the x64, x86, and AMD architectures [15]. This integration allows initiation of VDs as regular VMs to support the IoT virtualization. This approach enhances the way the IoT devices are virtualized by providing a homogeneous abstraction for the IoT platform to extend the physical capabilities of the devices. However, only the efficiency and correctness properties can be guaranteed with this approach. The real-time property cannot be assured as the communication between the physical devices and the IoT cloud platform uses the Internet where there is no control over the jitter and latency of the communications.

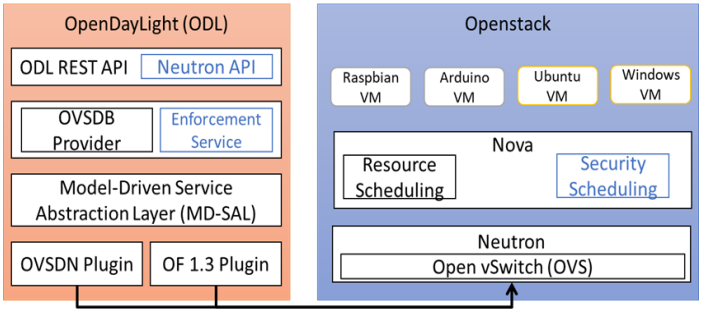

Figure 3 Enforcing network security with OpenDayLight

Regarding the security enforcement, our architecture introduces a novel approach to enforce the network related SCs. We introduce two main components on top of the OpenStack architecture as illustrated in Figure 3. The first component (OpenDayLight security enforcement service) challenges the way the SCs are enforced by leveraging the SDN capabilities of Openstack. This enforcement service lives on ODL and contains the information regarding the rules to apply to enforce the network related SC. To demonstrate the behavior of the enforcement service, the SSH access SC is chosen. Enforcement of this SC requires closing the SSH port to disable remote access to the VM. In the case of a regular VM, this rule can be implemented within the VM [16] using iptables or ufw [17] for Ubuntu VMs. However, this way of disabling the SSH access SC is not suitable for VDs due to the absence of a built-in firewalls which limits the implementation strategies. Therefore, the enforcement services proposed by our architecture pushes the implementation to the cloud infrastructure level. This means that the rules associated with closing the SSH port are sent to OVS to block the port by Neutron instead of blocking it from the VD. Another SC that is considered is transmission confidentiality that is achieved using encryption algorithms on the data to be transmitted. Encryption robustness differs from

one encryption algorithm to another with the more robust algorithms requiring more resources, as VDs usually lack powerful CPUs and resources in general they end up using weak encryption algorithms [18]. We suggest delegating the encryption and decryption of traffic to our ODL controller which can run powerful encryption, this shortens the distance the data needs to travel with weak encryption the controller will act as an encryption proxy switching from the VD's weak encryption to a stronger encryption for outgoing traffic and vice-versa for incoming traffic.

Some SCs require checks on the physical machine, data center, or region. Those SCs restrict the physical location where the VM or VD could be instantiated. To address this challenge, our architecture introduces a new way the VM and VD placement is done on Nova. We introduce the notion of security scheduling to Nova in addition to the existing resource scheduling. Typically, Nova scheduling focuses only on resource-based scheduling where the target host is determined by filtering the host set against the required resources needed to run the VM. Our security scheduling approach adds an additional layer to the Nova behavior to enable consideration of the SC during the scheduling process.

The combination between the Enforcement Service and the Security Scheduler allows Openstack to automate a significant set of the SC at the cloud infrastructure layer.

A. *Definition of the security profile*

For both the proof of concept and the impact experiments, a SC profile is needed to illustrate how they can be implemented and what is the impact of such an implementation on the system. From the full list of SCs we extract the subset of security controls that are relevant to IoT systems and devices. The selected SCs are added to a default baseline to tailor to specific IoT systems requirements. The filtering algorithm will take the tailored baseline as input and match it to the hosts that implement the security controls in that baseline. Figure 4 illustrates the process of tailoring the security baselines.

To apply our new security filters, we need a way to specify security requirements when requesting a new VD. OpenStack flavors are a way to describe requirements in terms of CPU, RAM, storage along with extra field called `extra_specs` that enables us to create custom requirements. In our case, we use the `extra_specs` field to describe security baseline requirements.

To generate experimentation data for each request we randomly choose the security baseline from the set of possible values {*LOW, MODERATE, HIGH*}. We then randomly add network and IoT SCs to make the tailored baseline, which will then be passed to the scheduler through the `extra_specs` field of the flavor. The selected baseline is mapped to the set of SCs that fall under it. The security-based filter will then check the implementation of each of these SCs on the hosts and eliminate those that do not have them implemented.

B. *Security Filtering*

The security filtering is applied when a new VD request is issued, similar to the same technique used by Nova-scheduler.

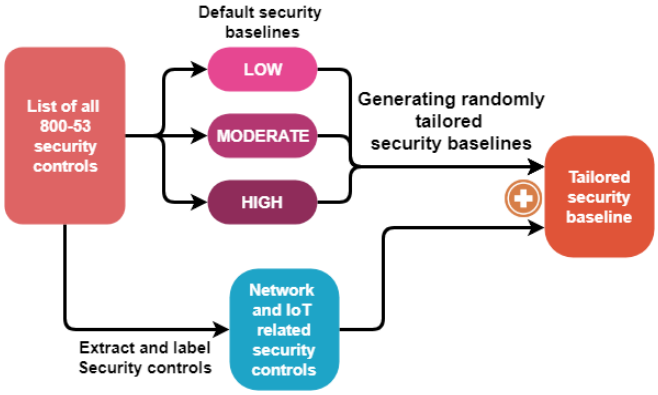

Figure 4: security baseline tailoring process

The following is the sequence of events triggered by a new VD request with an emphasis on the Nova-scheduler involvement: The requestor's identity is sent to Keystone[19] for authentication.

1. In case of a valid identity, a RESTful request is sent to Nova-API that is running on the controller node with multiple parameters including the image which is the name of the image containing the operating system the VD will run, and the flavor with the requirements.
2. An RPC request with the specified requirements is sent to Nova-scheduler using RabbitMQ.
3. Nova-Scheduler filters and weights the available hosts and identifies the most suitable hosts.
4. An RPC request is sent to Nova-compute on the selected host.
5. Nova-Compute starts the provisioning by requesting resources from glance (image file), neutron (IP address), cinder (disk volume).

Once all resources are reserved, the instance is provisioned and can be used. *Algorithm 1* is the filtering algorithm that Nova-scheduler will run to determine the hosts that are compliant with the specified security requirements in the `extra_specs` of the flavor.

Algorithm 1: SecurityFiltering (H, sr)

Input:
H set of available hosts
sr: Required security baseline
Output:
H' subset of hosts compliant with security requirements
Let $si_0, si_1, \ldots, si_{n-1}$ be the implemented security baseline on the available hosts $h_0, h_1, \ldots, h_{n-1} \in H$
while i<n **do**
  if sr $\subseteq$ si_i **then**
    H' = H' U $\{h_i\}$
  end if
  i = i+1
end while
return H'

IV. EXPERIMENTAL SETUP

This section describes our experiment that extends OpenStack to support cloud infrastructure level security capabilities. It aims to describe the environment and the software configuration that enables choosing the hosts where the VMs and VDs will be provisioned according to both the resources needed and the SC.

The experiment has two distinct objectives: First is illustrating a proof of concept regarding the implementation of the SC at the cloud infrastructure layer. Second is measuring the impact of such an implementation on the OpenStack's scheduling behavior.

A. *Experiment environment*

For this experiment, a private cloud testbed was deployed on three nodes with 40 CPU cores, 512Gb of RAM and 1.2 Tb of SSD storage using OpenStack. All the nodes have 10Gb network interfaces and are connected through a 10Gb fiber connection to a Brocade SLX switch. Figure 5 illustrates the physical topology used for this experiment. it shows also the services deployed on the Openstack controller and the compute nodes, it also shows a fourth node where OpenDayLight controller is installed and connected to Neutron via OpenVswitch.

This setup is used to illustrate the proof of concept of our approach. As Qemu is deployed on the compute nodes, we used those nodes as infrastructure to run VDs on top of Intel based processor. Even if the communications to the outside word does not preserve the real-time properties, our design choice was for a low-latency physical networking infrastructure to be as close as possible to real-time communication within the cloud infrastructure.

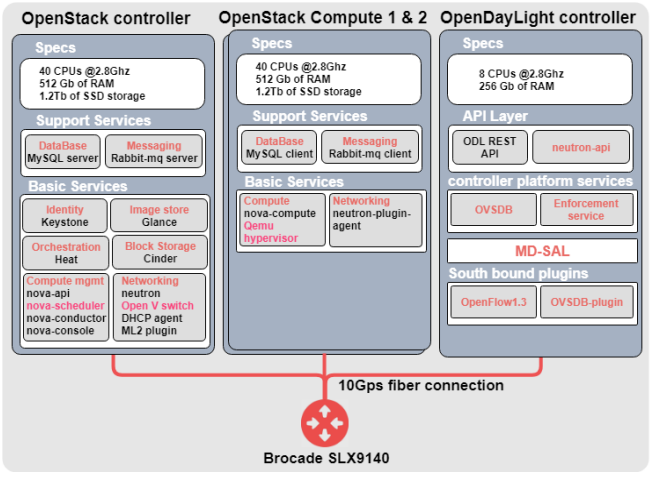

Figure 5: Experiment testbed

B. *IoT Virtualization*

We used Qemu with Openstack to virtualize IoT devices as VDs. Table 1 illustrates a subset of platforms that Qemu supports. We are interested in these platforms as they cover a significant part of the IoT devices targeted by our approach. We are using Advanced RISC Machines (ARM) to virtualize the Raspberry Pi Raspbian OS, Microprocessor without Interlocked Pipelined Stages (MIPS) to virtualize some of the Arduino platforms [20] and i386/x86-64 to virtualize industrial controllers like Siemens IoT2040 [21].

As IoT communications and virtualization have been subject of several efforts for defining protocols and interaction pattern specific to this domain of application such as the Mobile Offloading and Communication Protocol (MOCP) [22] and the Networks of 'Things' [23], we leveraged some of the concepts introduced by those efforts to build our testbed. We ensured that the primitives and element of the proposed VD respects the primitives and the elements to enable a composition as a network of things. However, we did not follow the MOCP specification in terms of interactions as we are only interested in the IoT virtualization concepts introduced by MOCP.

First, we use Qemu to run an emulation of the target operating system within a VD where we install all basic packages and update the system. Once completed, the running emulation is saved in an image file using the Glance imaging service. This will be used in the future to run VD instances with the installed packages on our Openstack environment.

TABLE I. CPU PLATFORMS SUPPORTED BY QEMU

Platform	**Virtualization mechanisms**		
	Hardware	***Emulation***	***Cloud Integration***
ARM	KVM	Yes	Yes
i386/x86-64	KVM, HAX, HVF, WHPX	Yes	Yes
MIPS	KVM	Yes	No

V. MEASUREMENTS AND RESULTS

We present our results obtained by measuring the impact of our approach on the scheduler as well as the automation coverage in term of cloud infrastructure automation of the SC.

A. *Impact on the scheduling*

We have measured the impact of the new security filter on the time consumed to provision a new VM. We executed multiple tests with incremental number of requests where we first apply the built-in OpenStack resource filter and subsequently apply our security baseline filter on the same data batch.

The results presented in Figure 6 show that the security filtering algorithm is very comparable to built-in algorithms such that provisioning a new VD with security requirements do not introduce a significant overhead to the scheduling component. Indeed, our results demonstrate that the processing time for the security filter is both linear and comparable to the resource filter currently implemented on OpenStack. Our results verify that the security filter algorithm that we propose in this paper could be integrated on an existing system without significant impact on the system.

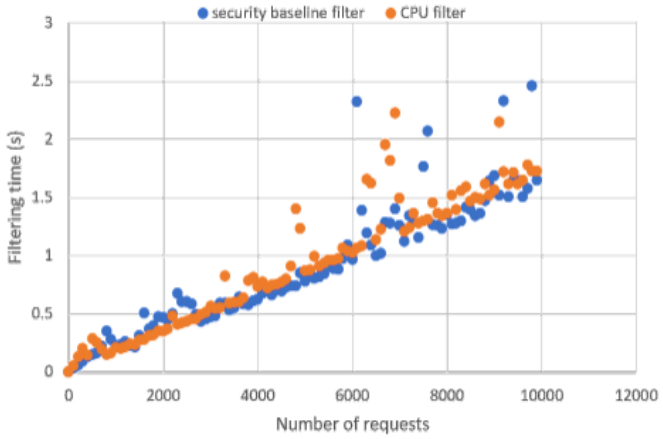

Figure 6: Security-based filtering execution time

B. *Coverage of the automation*

We have also measured the coverage of the automation per security baseline. The NIST SP 800-53 v4 lists more than 300 SCs across all baselines, some of these controls concern the organization's administrative processes and workforce responsibilities that requires human intervention resulting in very hard, if not impossible, to automate. Baselines are composed from a set of SCs where only network-related ones are automatable. To illustrate the current coverage of our approach, we identified the SC from the three baselines that meet the network automatization requirement. The criteria is used to determine if the SC is eligible for automatization is it eventual capacity to be translated in network controls. As example, SC that are related to network ports controls or authentication checks are typical SC that can be handled by our approach.

Figure 7 illustrates the number of the security controls that are automatable against the ones that needs to be implemented otherwise. Those results show that less than 20% of the possible SC's are eligible for automation.

VI. Conclusion and Future Work

In this paper, we present our contribution regarding the automation of the SC for cloud IoT platform. We illustrate the need for handling of this automation at the cloud level as the VDs are not ready to the run automation scripts locally because of the heterogeneity of the processors architecture. Moreover, we presented the proposed software architecture that enables the automation of the SC at the cloud level. Our experiments and results show that the security control filtering algorithm is not introducing a significant overhead to the cloud scheduler.

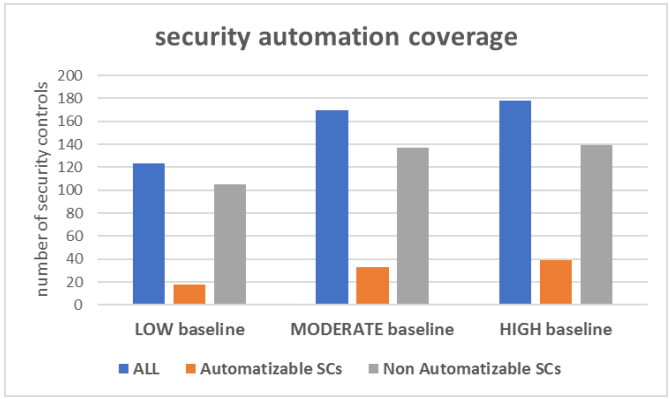

Figure 7: Security controls automation coverage graph

In terms of coverage, the 20% average coverage of our current approach needs to be improved. We are planning to extend our cloud level support beyond the network-based SCs. SDN allows us to control some aspects of the automation; however, we are exploring other software-defined solutions to cover non-network aspects of the SCs.

Disclaimer

Any mention of commercial products or organizations is for informational purposes only; it is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose.

References

[1] N. Penneman, D. Kudinskas, A. Rawsthorne, B. De Sutter and K. De Bosschere, "Evaluation of dynamic binary translation techniques for full system virtualisation on ARMv7-A," Journal of Systems Architecture, vol. 65, pp. 30-45, 2016.

[2] B. Baranidharan, "Internet of Things (IoT) Technologies, Architecture, Protocols, Security, and Applications: A Survey," in Handbook of Research on Cloud and Fog Computing Infrastructures for Data Science, IGI Global, 2018, pp. 149--174. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] P. Mell, T. Grance, The NIST definition of cloud computing, NIST Special Publication, pp. 800-145, 2011.

[4] Dimitrios Zissis, Dimitrios Lekkas "Addressing cloud computing security issues" Future Generation Computer Systems Volume 28, Issue 3, March 2012, Pages 583-592.

[5] B.Sotomayor et all, "Enabling Cost-Effective Resource Leases with Virtual Machines", 2007.

[6] Openstack: what is openstack? [online] Available: "https://www.openstack.org/software/"

[7] OpenStack and OpenDaylight [online] Available: "https://wiki.opendaylight.org/view/OpenStack_and_OpenDaylight"

[8] A.Patel et. al, "Embedded Hypervisor Xvisor: A comparative analysis" 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015.

[9] L.Reinfurt et. al, "Internet of things patterns" 21st European Conference on Pattern Languages of Programs Article No. 5 July 2016.

[10] S.Haag, Reiner Anderl "Digital twin – Proof of concept", 2018 Society of Manufacturing Engineers (SME) Volume 15, Part B, January 2018, Pages 64-66

[11] Openstack documentation - Filter Scheduler [Online] Available: "https://docs.openstack.org/nova/rocky/user/filter-scheduler.html"

[12] "Security and Privacy Controls for Federal Information Systems and Organizations," NIST Special Publication 800-53 Revision 4

[13] S.M. Radack, "Federal Information Processing Standard (FIPS) 199, Standards for Security categorization of federal information and information systems" 2015

[14] Qemu, an opensource emulator [online] Available: https://www.qemu.org/

[15] Libvirt, virtualization API [online] Available: https://libvirt.org/index.html

[16] C.Tunc et. al, “Cloud security automation framework” 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems, Sept. 2017.

[17] UFW, uncomplicated firewall [online] Available: https://www.linode.com/docs/security/firewalls/configure-firewall-with-ufw/

[18] K.Boeckl et. al “Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks” NISTIR 8228, Sept 2018.

[19] Keystone, the openstack Identity service [online] Available: https://docs.openstack.org/keystone/latest/

[20] MIPS, CPU architecture [online] Available: https://www.mips.com/products/architectures/

[21] J.Nakajima, et. al “X86-64 XenLinux: Architecture, Implementation, and Optimizations”, Proceedings of the Linux Symposium, July 2006

[22] C.Mahmoudi, Fabrice Mourlin “MOCP: An Offloading Protocol for Mobile Cloud and IoT virtualization”, International Conference on Selected Topics in Mobile and Wireless Networking, June 2018

[23] J.Voas, “Networks of ‘Things’” NIST Special Publication 800-183, July 2016