2019 IEEE International Conference on Edge Computing (EDGE)

Edge Computing Perspectives: Architectures, Technologies, and Open Security Issues

Maurantonio Caprolu*, Roberto Di Pietro*, Flavio Lombardi†, Simone Raponi*

*Hamad Bin Khalifa University (HBKU) - College of Science and Engineering (CSE)
Division of Information and Computing Technology (ICT) — Doha, Qatar
{mcaprolu, sraponi}@mail.hbku.edu.qa, rdipietro@hbku.edu.qa
†Istituto per le Applicazioni del Calcolo, Consiglio Nazionale delle Ricerche (IAC-CNR)
Rome, Italy — flavio.lombardi@cnr.it

***Abstract*—Edge and Fog Computing will be increasingly pervasive in the years to come due to the benefits they bring in many specific use-case scenarios over traditional Cloud Computing. Nevertheless, the security concerns Fog and Edge Computing bring in have not been fully considered and addressed so far, especially when considering the underlying technologies (e.g. virtualization) instrumental to reap the benefits of the adoption of the Edge paradigm. In particular, these virtualization technologies (i.e. Containers, Real Time Operating Systems, and Unikernels), are far from being adequately resilient and secure. Aiming at shedding some light on current technology limitations, and providing hints on future research security issues and technology development, in this paper we introduce the main technologies supporting the Edge paradigm, survey existing issues, introduce relevant scenarios, and discusses benefits and caveats of the different existing solutions in the above introduced scenarios. Finally, we provide a discussion on the current security issues in the introduced context, and strive to outline future research directions in both security and technology development in a number of Edge/Fog scenarios.**

***Keywords*-Edge Computing; Containers; Unikernels; RTOS; Security.**

I. Introduction

Despite its widespread success, Cloud Computing is not a one-size-fits-all solution, one of the main issues being resource centralization. This in turn results in an increased separation between user devices and their clouds, leading to large average network latency and jitter, while delay-sensitive applications, such as gaming, augmented reality and e-health require low latency and jitter—as well as context awareness and mobility support [1].

Edge Computing is a distributed computing paradigm that was introduced to address some of the above issues by moving computing and storage away from centralized points [2]. In fact, Edge Computing pushes applications, data, and services geographically closer to where such services are requested. In particular, Fog Computing uses Edge devices to carry out most computation, storage, and communication locally. Such devices range from small IoT or IoE [3] platforms comprising a single CPU core and less than a Megabyte of RAM to fully fledged servers comprising multi-core CPUs, many-core GPUs, and Gigabytes of RAM.

Although the use-cases of the two current network paradigms (i.e., Cloud and Edge/Fog) are very similar, the context of use could be very different both for the applications and for the different type of devices involved in the architecture. For this reason, the security issues posed by virtualization, extensively studied for the cloud environment, need to be evaluated also in the Edge/Fog scenario.

Edge and Fog devices increasingly rely on Linux technology, be it a regular server distribution or an embedded operating system—for its proven reliability, widespread ecosystem, wide range of technology solutions available (e.g. virtualization), as well as to help decreasing the Time-to-Market for both hardware and software developers. Albeit other technologies exist today, the software trend is towards consolidation, and hence reusing as much of the existing and tested solutions is often a convenient choice, especially when it is possible removing unnecessary components.

Resources located on Edge/Fog servers are increasingly developed using virtualization technologies to offer more reliable, and performing services. Despite the above cited advantages, one should not neglect the potential security issues introduced by the very use of virtualization; if such issue are not correctly addressed, they could hinder the full realization of Edge potential.

Contribution. The main objective of this paper is to analyze the impact that virtualization technologies have on the Edge/Fog network architecture, discussing their advantages, and highlighting the security issues introduced by such a technology. To achieve this, we first categorized the devices used in the Edge and Fog levels, analyzing the virtualization support they provide. Then, after revising the general security issue of both the most used virtualization technologies (i.e., containers and unikernels), and the competing non-virtualized approach (RTOS), we evaluate those technologies in the context of four typical Edge Computing real-world use-case scenarios. Finally, we strive to shed light on some research directions and practical solutions that can affect

978-1-7281-2708-8/19/$31.00 ©2019 IEEE
DOI 10.1109/EDGE.2019.00035

(either positively or adversely) the overall security of the Edge/Fog paradigm in the future.

Roadmap. The remainder of this paper is organized as follows. Section II provides an overview of both the Edge and Fog architectures, while the existing supporting technologies are described in Section III. Section IV depicts the real world scenarios we take into account throughout the rest of the paper. Section V presents the security issues of the supporting technologies introduced in the previous sections. A detailed discussion is provided in Section VI, together with interesting possible future directions, whereas Section VII draws some concluding remarks.

II. Architectural Background

In this section we provide an overview of both the Edge and Fog Computing architectures, discussing their properties, differences, and hardware requirements.

There is no universally accepted definition of these two paradigms, but several scattered descriptions, sometimes overlapping among each other. This ambiguity is probably due to the fact that both architectures share the same goal: bringing the computation closer to the source of the data. Indeed, the low latency performance required by modern end-user applications has requested a change to the classic cloud architecture, where data are sent geographically far to be processed, thus increasing the network delay. In the Edge Computing architecture data are processed (also partially) within the same device that produced them, or in an external device, but located at the edge of the same network. Instead, in the Fog network paradigm, data are processed outside the network, but in a geographical location very close to the origin of the data. In this paper we will deal with both, as such in the following the two terms are used interchangeably. In order to fully understand the impact of virtualization technologies in the Edge/Fog network paradigm, in the following section we provide a comprehensive categorization of the hardware that could be used in the Edge and Fog level, depicted in Fig. 1. For each device category, we discuss its current support for virtualization, based on the hardware architecture and the available implementations.

Edge/Fog hardware: The Edge Computing level includes three types of devices: devices that generate the raw data (i.e., sensors), devices that receive the computed data (i.e., end-user devices such as smartphones), and devices that offer computational power or other services (i.e., servers). The Fog network paradigm instead, includes only servers, since data are both generated and used in the levels below.

1) Edge Devices: This category contains every device that generates data and/or executes an end-user application. As represented in Table I, these devices can be categorized into three main classes: constrained devices, single-board computers, and mobiles, respectively. The constrained devices category includes any IoT device used for different objectives, such as domotics, surveillance, automotive, and

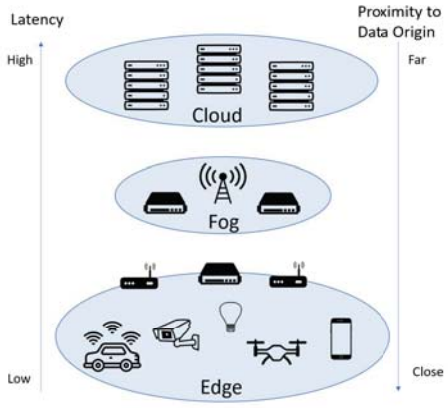

Figure 1. Edge/Fog Devices and their relative placement.

Category	Architecture	Example Devices	Virtualization Support
Constrained Devices	MSP430, Arm	Arduino, TelosB, OpenMote	✗
Single-board computer	x86-64, Armv8	Raspberry Pi, Intel NUC	✓
Mobile	Arm, Armv8	Android/IOS Smartphones	Not yet

Table I
Architectural solutions for Edge Computing devices

many others. This device category does not support any virtualization technology. In fact, due to several constraints such as production cost or physical constraints on characteristics such as size, weight, and available power and energy, these devices have very little available resources (at most 50Kb of RAM and 250Kb of storage)[1]. Single-board computers such as Raspberry Pi or Intel NUC, instead, due to the available resources and their architecture, offer full support to both hardware and software virtualization. Mobile devices, while having an architecture that offers full support for virtualization, do not offer any implementation that supports it yet.

2) Edge/Fog Servers: The devices that could be used as servers in both the Edge and Fog levels can be categorized in three main groups: general purpose servers (i.e., the same servers used on cloud environment), new platforms specifically designed for the Edge/Fog requirements, and other platforms designed for specific use-cases (i.e., automotive), respectively. Table II summarizes these categories, reporting examples of products available on the market and highlighting the offered virtualization support. Existing Edge/Fog servers available on the market, called cloudlets, are miniaturized versions of cloud servers, which are mainly structured based on CPUs with one or more GPUs co-processors. These devices are optimized for batch processing of in-memory data and can hardly provide consistent or predictable performance for processing streaming data

[1]https://tools.ietf.org/html/rfc7228#section-2

Category	Product	CPU Architecture	Virtualization Support
Automotive	Nvidia Drive PX 2 (Tesla)	Armv8-A	Vendor Limited
Edge/Fog Customized Platform	Intel Fog Reference Design Unit	x86-64	✓
	Cisco IOx Edge Compute Devices	x86-64	✓
Cloud Device	Generic Server	x86-64	✓

Table II
ARCHITECTURAL SOLUTIONS FOR EDGE/FOG SERVER

coming dynamically from I/O channels. Therefore, future Edge servers call for a new general-purpose computing system stack tailored for processing streaming data from various I/O channels at low power consumption and high energy efficiency [4]. The Fog reference architecture is described by the Openfog Consortium in [5]. Although a standard Fog device does not exist yet, Intel built the Fog Reference Design (FRD), a device compliant with the reference architecture. The FRD, as well as the Cisco Edge Compute Devices IOx [6], uses the Field-Programmable Gate Array technology (FPGA) to cast proprietary hardware in a chassis, allowing users to both configure and program it after manufacturing [7].

By relying on an x86-64 architecture, both the general purpose servers and the Edge/Fog custom platforms fully support virtualization. Nevertheless, for other platforms the virtualization capabilities depend on the support provided by the vendor. As an example, for the Nvidia drive board reported in Table II, the vendor provides an hypervisor able to run an operating system or a bare-metal application[2].

III. SUPPORTING TECHNOLOGIES

In this section, we provide an high-level description of the existing virtualization supporting technologies that can be employed within the Edge Computing networking architecture. As depicted in Figure 2, we take into account the containerization, the unikernel, and the Real Time Operating System approaches and technologies. Then, we discuss how these technologies can be used in the Edge/Fog network paradigm.

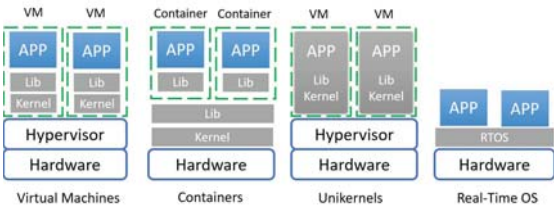

Figure 2. Supporting HW/SW Architectures

Containerization: Containerization is a lightweight alternative to full virtualization that allows the encapsulation of applications within specific recipients, called containers. The concept of container is far from being new, and has taken on different shapes over time depending on the degree of isolation provided. As opposed to virtualization, containers offer near bare metal performance, being a standard unit of software that contains the running code together with the related dependencies [8], [9]. This logical packaging mechanism decouples applications from the infrastructure, thus allowing the consistent and efficient deployment of container-based applications regardless of the environment (e.g., a data center, a laptop, a raspberry Pi, to name a few). Containers, similarly to standard processes, run directly on the host kernel and have the possibility to share libraries with the host, thus avoiding code duplication [9]. Docker is the world leading software container platform, providing a methodical way to automate the deployment of applications within portable containers [10].

Real Time Operating Systems: With Real-Time Operating Systems (RTOSes) we refer to any operating system able to serve real-time applications (i.e., applications that require both a high degree of reliability and precise timing) [11]. Most RTOS projects are proprietary solutions, albeit open source efforts are present. They generally feature a very thin and simple kernel taking exclusive control of the underlying hardware (i.e., they are bare metal in that they do not support virtualization mechanisms). In order to be considered real-time, every critical operations (e.g., OS calls, interrupt handling) performed by the Operating System must have a maximum time limit. This time management brings to a first categorization of the RTOS (see also [12]): a *Hard Real-Time Operating System* (HRTOS) is always able to guarantee a maximum time for each of the critical operations, while a *Soft Real-Time Operating System* (SRTOS) is able to guarantee a maximum time for each of the critical operations most of the time [13]. Furthermore, the RTOSes come with two common designs: (i) *Event-Driven*, when the OS switches task due to higher priority events; and (ii) *Time-Sharing*, when the OS switches task due to the regular clock activity. Edge Computing can potentially benefit from RTOSes for those services that require bounded response times [14].

Unikernels: The Unikernel approach also called Library OS [15], creates specialized, single-purpose, single-address-space images, moving the OS services upon which applications are built into the same address space of the applications themselves. The idea is to revolutionize the structure of Virtual Machines: "fat" machine images will be substituted by small, secure, fast, reusable images. The intuition comes from the fact that most of the online services (e.g., DNS servers) are based on heavyweight images, containing libraries that will never be used by the service itself. This "thinning out" of the libraries leads to safer images (i.e., the attack surface is substantially reduced), which eventually brings to the optimization of the resources of the machines

[2]https://docs.nvidia.com/drive/active/5.0.10.3L/nvfdn_docs/index.html#page/DRIVE Foundation SDK Development Guide/overview_components.html#

that host them. To deploy an application within an image, the developer has to cherry-pick the minimal set of operations that are strictly needed for the application to run. These libraries are compiled into the application executable through a library operating system with the aim of building sealed images, able to run directly on the hardware or an hypervisor (i.e., without an intermediary Operating System) [16].

Virtualization at the Edge of the Network: The virtualization supporting technologies described in previous sections can be used also in the Edge/Fog servers to provide more scalable, reliable, and performing services. A general use-case is depicted in Fig 3. The IoT sensors send the acquired raw data to Edge/Fog servers, instead of sending them to the cloud, and receive commands (solid lines). These servers are directly connected to the IoT network gateway (Edge) or somewhere outside the IoT network (Fog), but geographically close. Virtualization mechanisms are used to ensure the scalability of the network implementing on-demand services by starting a customized VM or a container at the time of the request. The servers elaborate and aggregate the received information, sending results to the cloud if the end-user service is not locally hosted. End-user applications, installed on smartphones or other client devices, send requests to Edge/Fog servers (dotted lines), instead of sending them to the cloud, decreasing the latency and improving the user experience.

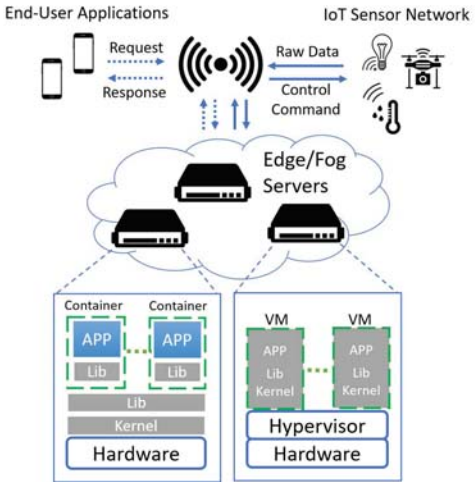

Figure 3. Virtualization on Edge/Fog Computing Architecture

IV. Usage Scenarios

In this section, we introduce four real world use-cases that justify the employment virtualization technologies: Cloud Gaming and Augmented Reality, Smart Home and Cities, E-Health, and Network Function Virtualization, respectively. For each use-case, we analyze how the Edge Computing paradigm could be adopted and we discuss possible benefits compared to the traditional cloud architecture.

Cloud Gaming and Augmented Reality: The worldwide number of active video gamers is growing year after year, with a forecasting of 2.73 billion video gamers expected by 2021 [17], representing a business that exceeds $50 billion worldwide [18]. This led to the creation of innovative applications able to combine computer-generated data with the physical reality [19], commonly named Augmented Reality Applications. Nevertheless, current network, storage, and processing limitations have slowed the spreading of this type of applications. The currently adopted Internet infrastructure is not ready for Augmented Reality due to network bandwidth limitations and the excessive latency caused by the geographical distance from the infrastructure. The adoption of Edge Computing could turn the tables. Indeed, this network infrastructure would allow data to travel for a shorted distance, thus obtaining a more immersive and interactive gaming experience (i.e., with reduced latency and lag times) [18].

Smart Home and Cities: Sensing devices are increasingly present in our homes and cities, as well as actuators. Managing and orchestrating data and control of such pervasive and heterogeneous landscape requires distributed intelligence that can scale up with the number of controlled devices without impacting on functionality and usability. Edge Computing can be vital to this kind of requirements, as it helps reducing response times to real-world stimuli by performing local computations and data buffering/caching.

E-health: The E-health term can comprise services and systems that enable real-time monitoring and exchange of patient data, as well as events triggered by devices attached or very close to the patients themselves. Some of these devices can automatically (re)act or request human intervention to healthcare professionals. Application examples are diagnosis and treatments at a distance, including remote monitoring of patients' functions and direct provisioning of care. Such smart medical devices mandatorily require a network connection to other systems. Failures or delays in the functioning of such devices and/or of the underlying network can cause harm/injury or possibly the death of the patient. As such, E-health has to be provided through reliable and low latency services and devices. Edge Computing can be vital to this requirement as it helps reducing interaction times between the patient and the healthcare infrastructure.

Network Function Virtualization: Network Function Virtualization (NFV) is a technology introduced to overcome the limitations of classic network architectures, characterized by proprietary heterogeneous devices that are usually very expensive both to deploy and manage. Leveraging virtualization technologies, NFV decouples software from hardware, allowing the execution of modular applications on standardized server platforms. Using this technology, any network service can be decomposed into a set of virtualized functions implemented in software running on standard physical servers. Then, those

virtualized functions could be relocated and instantiated at geographically different network locations, avoiding the need to purchase and install new hardware [20]. NFV provides a flexible way to design, develop, distribute, and manage network applications, introducing different advantages in addition to cost savings. Such technology could be used in Edge Computing to provide more efficient and scalable services.

V. Virtualization-originated Security issues

In this section, we study the security implications coming from the adoption of the approaches introduced in Section III. For each of the supporting technologies, i.e., containerization, Real Time Operating Systems, and unikernels, respectively, we evaluate the security analysis presented in the literature and we identify new intriguing security issues that the adoption of each technology could bring. Table III depicts a scenario-driven analysis of the possible attacks, where each row describes a possible scenario and identifies the malicious actions that an attacker can undertake, according to the adopted underlying technology, aimed at disrupting the system or compromising privacy. Note that the absence of the Real Time Operating Systems column in the above-mentioned table is due to the fact that RTOS support neither multiple application images nor their dynamic (re)deployment. In fact, they are more exposed than containerization and unikernels to any kind of vulnerability caused by the lack of security patches/updates.

Containerization - Docker: The worldwide diffusion of containerization in the recent years, and in particular of Docker, has opened the door to numerous studies aimed at identifying the related vulnerabilities and the security issues that could arise. Table III also depicts some possible relevant attacks related to the Docker use-cases studied in the literature. The amount and the severity of the identified attacks explain why security is one of the biggest concerns against the adoption of containers.

One of the first studies that analyzes the security of Docker is [21], where the author surveys the security of both Docker and the interaction with the security features of the Linux Kernel. In [9], the authors provide a detailed survey on the Docker ecosystem vulnerabilities. As a first contribution, they group the related studies in the literature in several macro-categories: container and virtualization comparison, security aspects of containers, defense against specific attacks, vulnerability analysis, and use-cases driven vulnerability analysis, respectively. Then, they investigate the security issues related to the Docker ecosystem according to different use-cases, where each one describes a different usage of Docker. Among their findings, the authors identify five categories of vulnerabilities: insecure production system configuration; vulnerabilities in the image distribution, verification, decompression. storage process; vulnerabilities inside the images; vulnerabilities directly linked to Docker; and vulnerabilities of the Linux kernel, respectively. Authors in [22] take into account 4 generalized use-cases: protect container from applications, inter-container protection, protect host from containers, and protect containers from host, respectively. For each use-case, the authors identify possible related attacks and evaluate possible solutions. The adjectives *semi-honest* and *malicious* in Table III are borrowed from [22]: authors define *semi-honest adversary* a passive adversary that can be part of the cooperation to obtain information but cannot deviate from the specified protocol, and *malicious adversary* as one that has the opportunity to deviate from the protocol to both obtain information and compromise the system.

RTOS: In order to guarantee bounded execution times, RTOSes have to limit complexity in managing hardware and applications. As such, they are mostly event driven to immediately react to events. Nevertheless, in their simplicity, RTOSes can still be vulnerable to various attacks [12] such as: DoSes, e.g., a running process can block shared memory and cannot usually be interrupted; code injection, i.e., changing the command flow to execute malicious code; data exfiltration, given that memory isolation is not usually fully enforced by RTOSes, relying on a simple shared memory layout;

Unikernel: To the best of our knowledge, the only work in the literature that provides a significant study of the security issues of the unikernel approach is [32]. In this report, the author first provides the background of Virtual Machines, unikernels, and hybrids, respectively, then analyzes the security isolation aspects of each technology. The study shows that unikernels provide small footprints and reduced overhead when compared to containers and virtual machines, but lack the ability to separate kernel code from the application code. Indeed, unikernel technology allows kernel code and application code to run at the same execution level (i.e., ring 0 on x86 –the privileged one). This feature on the one hand substantially contributes towards improving performances, because of the absence of unwieldy processor context switches, on the other hand it exposes the system to serious potential attacks. Indeed, by removing all the libraries the application does not need, unikernel is removing all the sophisticated security mechanisms the Operating System can boast. This means that if an attacker finds a 0-day attack in a unikernel application code she would have privileged access to the unikernel-based system (e.g., any buffer overflow can lead to the privileged execution of arbitrary code on the system) [33], [34]. Nevertheless, any hypervisor running at ring -1 can help detect and contain the security issue [24].

VI. Discussion and Future Directions

In this section, we map the real world use-cases introduced in Section IV against the supporting technologies

Scenario	Description	Possible Attacks (Docker)	Possible Attacks (Unikernel)
Protecting from untrusted applications inside images	Each application can be malicious, semi-honest, or honest, respectively. Furthermore, it is assumed there are applications that require root privileges.	Remote code execution; unauthorized access; network-based intrusions; virus, worm, trojan, ransomware; information disclosure, tampering, and privacy issues; privilege escalation, denial of service ([9], [21]–[23]).	Unikernels run the image at ring 0 (or equivalent). As such are fully exposed to malicious code in images. Nevertheless, the isolation provided by the hypervisor using hardware-assisted virtualization helps reducing the tampering and data access, as well as privilege escalation issues [24].
Inter-image protection	Images can be semi-honest or malicious and placed inside one or different hosts.	DoS on other images; ARP spoofing; MAC flooding; port/vulnerability scanning; remote code execution ([9], [22], [23]).	DoS and other attacks on images are feasible. Nevertheless, the isolation provided by ring -1 (or equivalent) might be used to contain the attack.
Protecting host from images	It is assumed that at least one image is semi-honest or malicious within an host.	Attacks on unnecessary services; container escape attacks; DoS; data tampering ([9], [22]).	The host system (hypervisor) runs at a higher privilege level, as such the possible attack are related to hypervisor vulnerabilities as in the regular VM+hypervisor scenarios.
Protecting image from host	Images are honest but the host is either semi-honest or malicious	Profiling in-image application activities; unauthorized access for image data; changing the image behavior ([22]).	Unikernels here have the same issues as containers, unless SGX [25] or similar technology is used.
Microservices-like	Each image hosts a single service in a single process.	Zip-bomb-like attacks; remote code execution; virus, warm, trojan, ransomware; privilege escalation; account hijacking; network communication tampering ([9])	Unikernels are vulnerable as well, but privilege escalation is more difficult and it has to resort to some kind of hypervisor vulnerability.
Application Image Distribution	Using Docker as a way of shipping virtual environments.	Data leakage; DoS; DoS on other containers; attacks on the container integrity; privilege escalation; container escape attacks([9])	Unikernels are equally vulnerable to third-party image hosting. Only privilege escalation is harder (unless some CPU backdoor is leveraged by the hosted code [26]).
Image deployment on the Cloud	Docker integration provided by the main Cloud Providers.	Data leakage; DoS; DoS on other containers; Zip-bomb-like attacks, remote code execution; virus, warm, trojan, ransomware; privilege escalation; account hijacking; network communication tampering ([9]).	Same as above, similar to containers but potentially more isolated when deployed.
Image repository	Each image has been provided by a repository through a distribution process	Zip-bomb-like attack; remote code execution; virus, warm, trojan, ransomware; privilege escalation; data leakage ([9], [22], [27]–[30]).	Same as above, similar to containers but potentially more isolated when deployed.

Table III
MOST RELEVANT SCENARIO-DRIVEN ATTACK IDENTIFICATION

Scenarios	Tech Performance Fitness	Security Impact
Cloud Gaming Virtual Reality	RTOSes and containers fit better as they provide slightly less overhead (read:latency) than unikernels	Low security impact, as attacks would mostly just cause a DoS in the game being played.
Smart home Smart Cities	Containers and unikernels are equivalent, with a small advantage of unikernels for security isolation and an advantage of containers for the ease of setup. RTOSes are relevant only for constrained devices	Here the consequences of a successful attack can be more serious e.g., up to the house catching fire or some DoS in the surveillance cameras.
E-Health	Unikernels have a small advantage over containers due to the increased security isolation/resilience that is crucial in this contest. RTOSes could be adopted only in some corner case on constrained devices.	Vital is the keyword, as DoSes and malicious software/device can possibly cause harm to the patient.
NFV	Containers and unikernels are roughly equivalent, with a small advantage of unikernels for security isolation and an advantage of containers for the ease of configuration; RTOSes are still relevant but suffer from difficult updates	The impact on security is relevant as malicious/altered software might cause relevant DoSes to other applications relying on the network.

Table IV
EDGE-IOT SCENARIOS, REQUIREMENTS, AND SECURITY IMPACT [31]

introduced in Section III. The goal is to understand how the application of these technologies can affect the specific scenario with respect to two fundamental dimensions: (i) performance; and, (ii) security. The result of this study is summarized in table IV.

One issue longitudinal to all the above use-case scenarios is related to security software updates. In fact, there is a pervasive strong need to ease/automate deployment of security fixes over time, as new vulnerabilities have to be patched quite often and, ideally, within the shortest delay. Containers, but even more unikernels, can leverage the flexibility given by the additional layers between application images and the hardware to provide seamless patching. RTOSes can be updated in principle, but this process usually requires manual intervention and physical access to the hardware. As such, software update scalability is quite limited.

In the Cloud gaming/virtual reality use-case scenario, increasingly present in the years to come (see the efforts by Google and others [35]), guaranteeing performance is at premium with respect to ensuring code/data integrity. Approaches such as RTOSes and unikernels might be suitable to optimize this dimension. On the one hand RTOSes—e.g., proprietary solutions—, albeit viable for large firms, are usually more expensive and, as discussed above, feature

more complex software updates. On the other hand, a microservices container-like approach would help to ease the need for updates, and it could be feasible when digital right management is not a critical constraint—e.g., checking the code integrity of the video game for billing purposes. Moreover, unikernels would also ensure a reduced attack surface that will help guaranteeing code integrity, ease of updates, and adequate performance (i.e., reduced latency).

As regards the Smart Home and Cities use-case scenario, robustness and reliability are way too relevant with respect to the ease of updates and performance. As such, leveraging unikernel technology can be increasingly convenient, unless the involved devices are so simple and cheap that hardware virtualization support cannot be guaranteed. In this latter case, current bare metal RTOSes could be the most viable (i.e., the sole) option. However, this solution would be increasingly subject to attacks, due to the difficulty in keeping such devices up-to-date. This could potentially lead to experience large Denial of Service (DoS) attacks, or other massive issues that might require device re-deployments.

The E-Health use-case mandates reliability, security, and privacy. As such, the performance gains of RTOSes and unikernels can be set aside and unikernels appear as the most suitable approach, leveraging on the reduced attack surface and ease of updates. The implication of the above introduced choice is that more and more efforts devoted to simplifying image creation, updating, and management of unikernels [36] would be needed, to maintain, extend, and secure the success of this technology.

As for the Network Function Virtualization scenario, some detailed examples are provided in the following.

The first example regards the Domain Name System (DNS), i.e., the decentralized hierarchical system providing translation services between users and Internet-connected resources. In its traditional implementation, the DNS server provides the user with the IP address of the requested resource, according to its database. NFV enables developing a new type of advanced services that includes both the DNS and the end-user services. This technology can be used to run microservices that only exist after they have been resolved in DNS. A good example is Jitsu [37], a DNS server that automatically boots virtualized instances of the resource requested by the user. When Jitsu receives a DNS query, a virtual machine is booted automatically before the query response is sent back to the client. This technology could be deployed at the Edge of the network, providing efficiently and scalable on-demand systems running a different image for every URL. Container and unikernel technologies are both viable as per performance (i.e., boot time). Nevertheless, unikernel images, when properly created (i.e., by stripping out all unnecessary functionality), can offer a very limited attack surface, further enhanced by the isolation among services provided by the hypervisor. As such, one of the main issues with unikernels' security is being able to remove all unwanted/unneeded code from images. To this end, automated smart tools for image creation are sorely needed (see also [36]).

A second example regards the dynamic nature of the virtual networks that implies new security needs that traditional firewalls are unable to meet for different reasons. First of all, traditional firewalls provide effective security on specific and static network topology. Unfortunately, they do not have enough flexibility and adaptivity to provide the same security level in a highly dynamic environment, where Virtual Machines (VMs) are dispersed and continuously migrated among different network segments. To mitigate this problem, firewalls could be implemented as a software instance, breaking the dependency on fixed network topology and providing the necessary flexibility to protect virtual networks [38]. In this particular use-case scenario, containers and unikernels are mostly equivalent. Nevertheless, the containerization technology might be preferable as it can be configured in a simpler and faster way than unikernels. It remains to be seen, as per the previous use-case, whether automated, simplifying delta-based image update approaches will improve the unikernel user-friendliness [36].

VII. Conclusions

In this paper we have first discussed how the Edge/Fog paradigm relies on virtualization technologies in order to achieve high performance and scalability, that are at the basis of its success. Later, we have surveyed the possible attacks against the most relevant virtualization technologies. Then, we have contextualized these threats into four different Edge/Fog Computing scenarios. For each scenario, we have discussed advantages and drawbacks related to the adoption of every considered virtualization technology, also highlighting the possible impact of the identified attacks. Finally, we have indicated some promising future research directions, as well as possible technological development.

Acknowledgement

This publication was partially supported by awards NPRP-S-11-0109-180242, UREP23-065-1-014, and NPRP X-063-1-014 from the QNRF-Qatar National Research Fund, a member of The Qatar Foundation. The information and views set out in this publication are those of the authors and do not necessarily reflect the official opinion of the QNRF.

References

[1] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[2] J. Gedeon, J. Heuschkel, L. Wang, and M. Mühlhäuser, "Fog computing: Current research and future challenges," *KuVS-Fachgespräch Fog Comput*, vol. 1, pp. 1–4, 2018.

[3] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog Orchestration for Internet of Things Services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, 2017.

[4] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?" in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[5] "Openfog reference architecture for fog computing," OpenFog Consortium, Tech. Rep., 2017.

[6] "Cisco IOx Documentation," https://developer.cisco.com/docs/iox/\#\!introduction-to-iox/what-is-iox.

[7] "Intel Fog reference design overview," https://www.intel.com/content/dam/www/public/us/en/documents/design-guides/fog-reference-design-overview-guide.pdf.

[8] "What is a container?" https://www.docker.com/resources/what-container, 2018.

[9] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem–vulnerability analysis," *Computer Communications*, vol. 122, pp. 30–43, 2018.

[10] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[11] P. Hambarde, R. Varma, and S. Jha, "The Survey of Real Time Operating System: RTOS," in *International Conference on Electronic Systems, Signal Processing and Computing Technologies*, 2014, pp. 34–39.

[12] L. Pike, P. Hickey, T. Elliott, E. Mertens, and A. Tomb, "TrackOS: A Security-Aware Real-Time Operating System," in *Runtime Verification*, 2016, pp. 302–317.

[13] "What is a real-time operating system (rtos)? – white paper," http://www.ni.com/en-lb/innovations/white-papers/07/what-is-a-real-time-operating-system--rtos--.html, 2012.

[14] A. Manzalini and N. Crespi, "An edge operating system enabling anything-as-a-service," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 62–67, 2016.

[15] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *Acm Sigplan Notices*, vol. 48, no. 4, pp. 461–472, 2013.

[16] R. C. Pavlicek, *Unikernels: Beyond Containers to the Next Generation of Cloud*, 2017.

[17] "Number of active video gamers worldwide from 2014 to 2021 (in millions)."

[18] M. Wojcik, "Mobile gaming at the edge," 2019.

[19] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, 2017.

[20] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[21] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[22] S. Sultan, I. Ahmad, and T. Dimitriou, "Container Security: Issues, Challenges, and the Road Ahead," *IEEE Access*, vol. 7, pp. 52 976–52 996, 2019.

[23] T. Combe, A. Martin, and R. Di Pietro, "To Docker or Not to Docker: A Security Perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.

[24] F. Lombardi and R. Di Pietro, *Security for Cloud Computing*, ser. Artech House computer security series. Artech House, 2015, ISBN 9781608079902.

[25] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 689–703.

[26] C. Domas, "Breaking the x86 ISA," 2017.

[27] B. Tak, C. Isci, S. Duri, N. Bila, S. Nadgowda, and J. Doran, "Understanding security implications of using containers in the cloud," in *2017* {*USENIX*} *Annual Technical Conference (*{*USENIX*}*ATC 17)*, 2017, pp. 313–319.

[28] J. Gummaraju, T. Desikan, and Y. Turner, "Over 30% of official images in docker hub contain high priority security vulnerabilities," *https://banyanops.com*, pp. 1–6, 2015.

[29] O. Henriksson and M. Falk, "Static Vulnerability Analysis of Docker Images," 2017.

[30] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 269–280.

[31] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT Edge Computing with Lightweight Virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[32] M. J. De Lucia, "A Survey on Security Isolation of Virtualization, Containers, and Unikernels," US Army Research Laboratory Aberdeen Proving Ground United States, Tech. Rep., 2017.

[33] R. Bias, "Unikernels Will Create More Security Problems Than They Solve," 2016.

[34] J. Dileo and S. Michaels, "Unikernel Apocalypse: Big Trouble in Ring 0," 2018.

[35] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. Oliver Wu, "Improving Cloud Gaming Experience through Mobile Edge Computing," *IEEE Wireless Communications*, pp. 1–6, 2019.

[36] "Unik: A platform for automating unikernel & MicroVM compilation and deployment," https://github.com/solo-io/unik, 2018.

[37] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, D. Sheets, D. Scott, R. Mortier, A. Chaudhry, B. Singh, J. Ludlam *et al.*, "Jitsu: Just-in-time summoning of unikernels," in *12th* {*USENIX*} *Symposium on Networked Systems Design and Implementation (*{*NSDI*}*)*, 2015, pp. 559–573.

[38] J. Deng, H. Hu, H. Li, Z. Pan, K. Wang, G. Ahn, J. Bi, and Y. Park, "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," in *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015, pp. 107–114.