Securing Microservices

Antonio Nehme, Vitor Jesus, Khaled Mahbub, and Ali Abdallah,
Birmingham City University, School of Computing and Digital Technology, Birmingham, UK

Abstract—Microservices has drawn significant interest in recent years and is now successfully finding its way into different areas, from Enterprise IT to Internet-of-Things to even Critical Applications. This article discusses how Microservices can be secured at different levels and stages considering a common software development lifecycle.

Keywords—Microservices, Containers, Security.

1 Introduction

Designing applications changed throughout the years, from the early client/server architectures to service-oriented architectures (SOA) and, now, what can be seen as a new SOA paradigm [1], [2], Microservices. Whereas a main driver for SOA was the need to reuse software components, Microservices goes further for two broad reasons. First, it fully allows the so-called Conways Law which means the application closely follows the structure of the enterprise, such as its processes and workflows [3], [4]. Second, it confines the complexity of each application component to a number of small but highly manageable components.

Microservices, per se, is not a new architectural style when thinking of SOA: it is rather SOA implemented following current trends and technologies such as automation of infrastructure operations with DevOps and the adoption of containers [1], [5], [4]. In terms of sectors we see Microservices being introduced in all types of applications, from Business Logic to Internet-of-Things to Critical Applications and Utilities [6].

As a new fast growing application development architecture, yet still maturing, new challenges are introduced, and security comes at the forefront. This paper discusses Microservices from a security perspective. Rather than addressing this topic from a specific angle, we try to lay out a comprehensive approach, that is, we discuss all phases of a typical project lifecycle and related Security context: design, development and testing, business-as-usual (maintenance, verification, monitoring, etc.), infrastructure and interfaces with external parties.

Being a direct evolution of SOA, but also an application development practice, security for Microservices needs to be progressive where it shares commonalities with general software development security. In contrast, new approaches are needed to cope with the fundamentally different approach to application development. In this sense, this paper guides the reader from architectural design (specific to Microservices) to implementation and maintenance (general to software development) and puts less focus as we progress to the general aspects of Software Security. Since we do not assume previous knowledge of Microservices, this paper also discusses the fundamentals of this paradigm, and how it evolved from SOA, and lists key references to its background. This is complemented with a set of references concerning industry initiatives, known security challenges and lessons learned so far, relevant projects and standardization efforts.

The structure of this article is as follows. We start by introducing Microservices and pointing the reader to seminal works and best-practices along with its limitations; we then draft a reference model to guide our analysis. In section 3, we break down Microservices in its key components and provide a Security analysis using a top-down approach: the individual

components of a microservice, the application architecture, the supporting infrastructure, key management, networking aspects, containers, and external interfaces. We also discuss Microservices from the perspective of the software lifecycle. Moreover, we review current guidance for security controls, and illustrate with recent guidance from NIST [7]. In Section 4, we draw the essential lines of a secure deployment and modify our reference model to take into account our findings and recommendations. The last section concludes the article.

2 MICROSERVICES

In SOA, services are mostly implemented as monolithic applications. Consider a point of sale (POS) system: when somebody pays a bill, a transaction is sent along with a balance check; given the availability of funds, the person's credit gets updated, and email notifications are sent as receipt of payment. This is done through API calls. Past SOA applications typically rely on a monolithic architecture which means that the source code is deployed as a single executable artifact and developed using one programming language or framework [8].

While monolithic implementations of SOA enabled rich applications, their limitations soon became apparent:

- Large monolithic applications are complex and hard to maintain [8].
- Maintaining the codebase of a large application introduces time-consuming tasks such as long building and deploying phases, since a small change affects the entire application [8].
- Regular and fast delivery cycles are impractical.
- The entire life of the application is limited to the initial choices of technologies [8].
- There is inefficient allocation of resources given that scaling one popular service requires resources to be allocated to the entire system [8], [9].
- Monolithic applications are more prone to single points of failure [9].

A clear solution is to decompose monolithic applications into small and independently deployable services with each as simple as possible, performing one small business function, and running independently from others [4]. The resulting components, in larger numbers but individually much simpler, communicate with each other, in order to achieve the same level of functionality as in a large monolithic architecture.

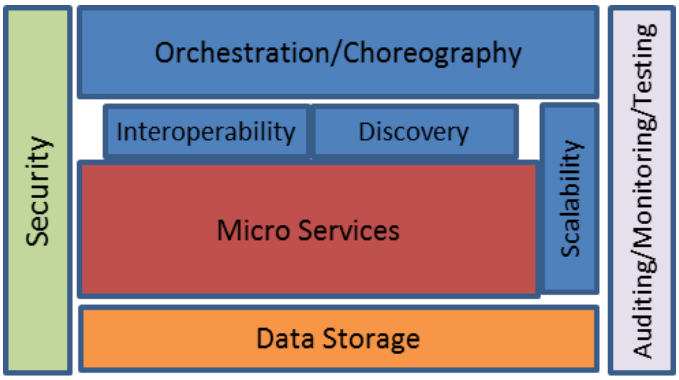

Figure 1. Microservices and Overarching Challenges

Due to their granular nature, microservices entail different components summarized in Figure 1. Microservices need to scale while remaining discoverable and interoperable. State changing events need to be handled by an event store. Orchestration and choreography, which include communication, are key components that are dependent on interoperability, discovery, scalability and event handling. Finally, the entire application needs to be secured, tested and monitored.

2.1 Features of a Microservice Architecture

Microservices architectural style is considered an evolution of the traditional SOA monolithic implementation. It emphasises on dividing systems into small services (the microservices) that perform cohesive business functions [10]. Cohesive in this context means implementing functionalities only related to the concern of the business function that a microservice implements. Microservices have to be loosely coupled, meaning that each service should have the ability to be deployed on its own [8]. It should also have a bounded context so that a service should function without knowing anything about other services [8], [3]. Each should also be autonomous and independently deployable [11], [3].

With this approach, complex systems are developed by joining independent microservices, distributed across different systems, which communicate with each other via lightweight mechanisms over a network [10]; This implies, for example, favoring REST interfaces over the complexity and heavy processing weight of SOAP.

Each of these services typically has its own management and its own database which is often referred to as polyglot persistence. It ensures loose coupling between microservices and allows each service to use the database that best suits its needs [12]. A further advantage is that this allows factorising the workload among different services which scale independently on demand [13], thus fit to be used, for example, in a cloud environment [4]. Finally, it also allows each service to use any convenient environment necessary. Applications for Internet-of-Things can exploit this feature to the fullest when compared to monolithic applications [12].

2.2 Handling the Scale Complexity of Deployment: Containers

Each microservice should be kept as simple as possible which has the further advantage of requiring fewer resources. Given the independent deployability requirement, this currently poses a problem as the lowest-spec server (e.g., on a public cloud provider) is usually too expensive for the resources a microservice needs. Also, setting up virtual machines becomes a complex matter with the diversity of dependencies [14].

Containers are used to mitigate this problem. In a typical setting, many containers, each running a service, run on the same kernel and hardware while being (logically) isolated from each other [15].

A popular implementation of containers is Docker, and it is supported by many tools like Kubernetes [11]. Docker containers are relatively easy to clone with the availability of its registry services, DockerHub.

2.3 End-to-End Coordination of Microservices

In order to achieve application consistency and correctness, microservices are required to cooperate and communicate. Similar concepts in SOA still apply. There are two main mechanisms: orchestration, requiring a central service (the conductor) to send requests and organise the workflow, and choreography, where each service reacts according to events or triggers [16].

Orchestration is normally executed at the gateway level which is a single entry point to the system which makes it ideal for storing logs and auditing tasks. As for choreography, an event store can be used with the ability to store and publish events. Events should be divided in categories to which microservices can subscribe.

In terms of infrastructure, a typical end-to-end model is depicted in Figure 2. For readability, we abstain from using formal software tools. As shown in the figure, the gateway handles requests from a diversity of external clients. Also by having a central position, it can return tailored responses according to the client type [16]. It also handles access management by communicating with an authorization server. The gateway forwards requests to be processed by microservices. To achieve a particular business functionality, microservices communicate with each other by producing and consuming events using an Event Broker, whose role is to publish and store state changing events. As shown in the figure, each microservice can publish and subscribe to events of different categories. An event broker is an autonomous application and, due to its role, is essential in auditing and monitoring.

To illustrate with the POS use-case previously introduced, Transactions, Balance, and Notifications could be examples of microservices. Users, machines, and other services authenticate by giving their credentials. A request to the authentication server verifies the authentication material. If valid, a transaction gets sent with an amount to a Transactions microservice. This triggers an event for the Balance microservice which, given available funding, publishes another event allowing the transaction to complete. The Transaction service listens to this event, allows the transaction to complete, and publishes an event of a successful transaction on a channel to which Balance and Notifications services are subscribed. Balance

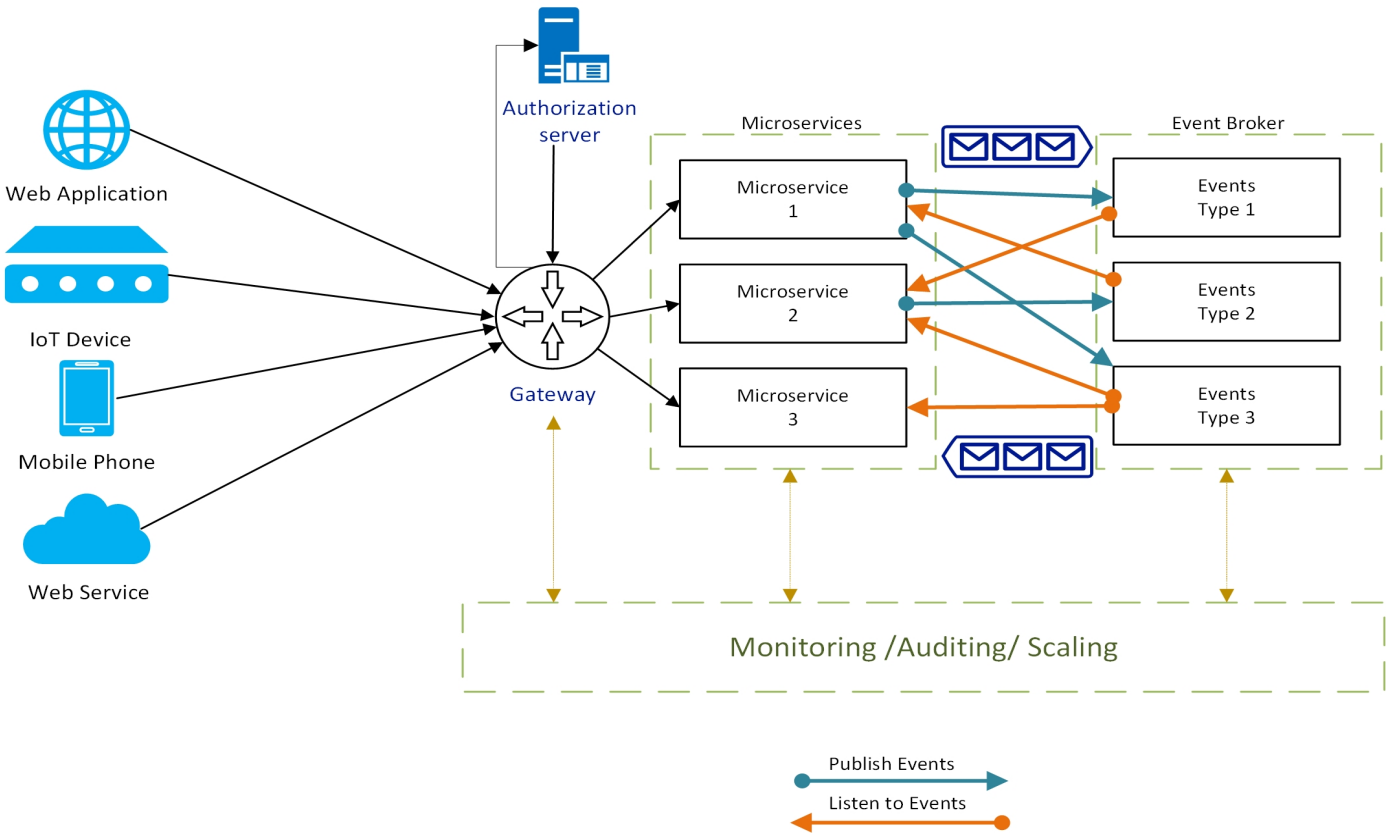

Figure 2. Representative model of a Microservices-based Application

gets updated and a notification gets sent to the user.

3 Beyond the Promise: Security Challenges

Security is multidimensional in the sense that it needs to be present at multiple layers of an application and at all stages of its development. For microservices, our security model has four broad dimensions:

- the microservice *components* themselves, from design to implementation.
- *the application architecture* and the potential need of specific security components or elements, such as instrumentation and detection.
- *the underlying infrastructure*, such as the Operating Systems and the network.
- *external interfaces* used for inter-domain communication, where multiple third-parties may need to cooperate, each with its own security controls.

We now elaborate on each of these dimensions.

3.1 Secure and Trusted Components

Authentication and authorization are essential steps towards securing services. Microservices should only be invoked after requesting authentication and, ideally, authorisation if levels of privileges are available. OAuth (currently in version 2.0) and OpenID Connect are frameworks that lend themselves to typical implementation of microservices that use RESTful APIs [17]. In essence, an access token is issued by an authorization server to a trusted client application. Note that trust is directly relatable to the coordination model. Verifying the access token at the gateway level makes it vulnerable to the Confused Deputy Problem [16]. This comes from microservices trusting the gateway based on its mere identity (sometimes even an IP address), which makes it open to misuse if compromised. Having access control enabled and scopes of the access token checked by microservices prior to responding to a request is a possible mitigation. Note that having a dedicated service acting as an authorization

server provides three main benefits: decoupling and isolation in case of the system is compromised, separation of concerns, and an auditing point [17]. OpenID Connect is built on top of OAuth 2.0, and uses JWT (JSON Web Tokens) as an identity token [18], [17].

3.2 Secure Architecture

A common model for microservices uses API Gateways. Being a dedicated element that does not directly participate in the application itself, it can also act as an Intrusion Detection System (IDS). However, IDS for microservices can be challenging as the signatures for the services need to be, typically, customised to the application, depending on the level of traffic inspection. Availability is also a key component of a Security model; it can be achieved by having elements to detect (by querying, for example) services that are down. If the case, the gateway and the coordination logic should be updated in order to action failover mechanisms.

Moreover, architectural decisions should be carefully thought of to avoid incidents similar to Netflix compromise in 2015, which was due to allowing access to all users cookies from any subdomain. This allowed an adversary to use netflix.com services from one compromised subdomain [19].

A final mention should be made to key management. Given the large number of services, managing cryptographic material is likely to require using Key Vaults and hardware modules.

3.3 Secure Infrastructure

By infrastructure, one means network, servers, devices, specialised elements (such as gateways) and the containers themselves along with Operating Systems.

More than any other paradigm, Microservices depend on fast network messaging given the granularity of each component. Further, inter-service traffic should follow policies derived from the application logic. Finally, note that several applications, each with a large number of services, can coexist together in the same infrastructure and network. Starting at the network level becomes essential, given that microservices brings the potential of increasing the attack surface when compared to a monolithic architecture [8]. Moreover, due to the containerisation trend, special attention is required for the risk of inadvertently using 0-day vulnerabilities in the components that come from public repositories like Dockerhub. A survey by BayanOps in 2015 revealed that three out of four official Docker images created during that year have relatively easy to exploit vulnerabilities which can potentially have high impact [20]. A good approach is to use Docker Security Scanning add-on prior to using images. Another good practice is to plan security roles within containers rather than running root users.

A further challenge is filtering and monitoring traffic for microservices at a level close to the application, as deep-inspection rules need to be made custom to the application. Common web attacks such as SQL injection, are easily detected by commercial Web-application firewalls; however, these are not particularly suitable for microservices.

Containers firewalls attempts do exist, however, with Project Calico being an example. This project can be integrated with Kubernetes, and allows creating policies and firewall rules at the pods level. Pods, holding containers, will scale with the firewall rules.

Overall, securing the network and server infrastructure can use a mix of current technologies to protect up to the container level. Past the container or hypervisor, application security becomes challenging as discussed in the next section.

3.4 Securing the Development Lifecycle and Governance

At this stage, a microservices architecture should draw on well-known secure software development best practices as, in the end of the day, this is software development as usual. Automated testing and verification becomes crucial as typically these applications are developed using Agile methodologies and rely on fast iteration cycles. In general, a comprehensive Secure Software Development Lifecyle (S-SDLC) comprises of

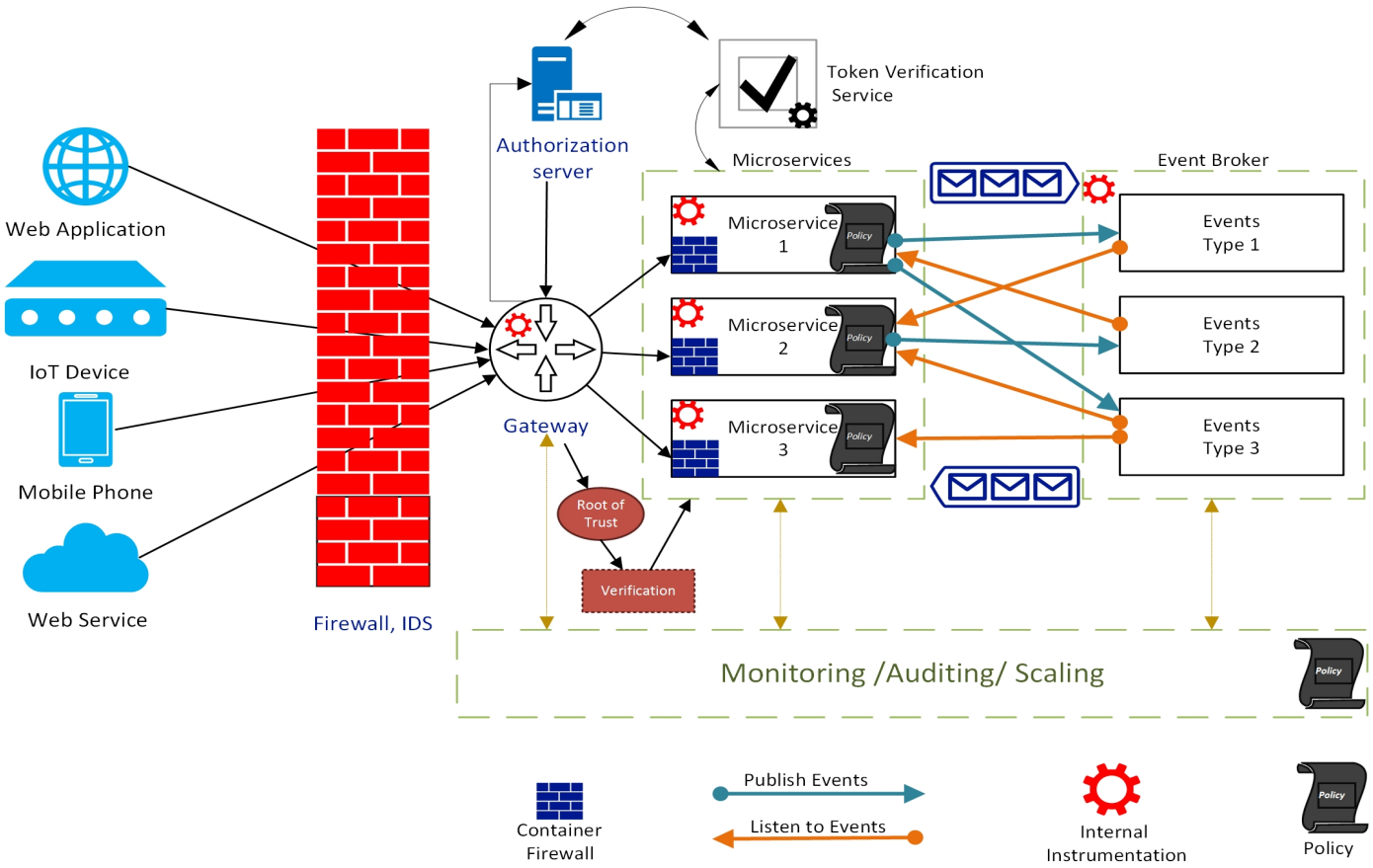

Figure 3. Typical Deployment of Microservices

- early risk assessment before design starts (e.g., handling trust, cryptographic material, etc), relevant at the architecture layout phase but also when selecting and assessing tools and frameworks for their own S-SDLC
- adding accessory functions to the core functionality in order to support security monitoring, auditing, testing and interfaces with external security elements [5]
- development with security in mind, following each language and framework recommendations and, ideally, third-party code reviews
- deployment of the application along with security tests and verification tools that should be continuous and periodic and include vulnerability management
- secure and safe retirement of components and modules

NIST SP 800-190 [7] is in draft stage and offers guidance regarding containers. In a nutshell, the takeaway advice consist of

- Always use container-specific OSes, which are hardened to reduce attack surface.
- Execute highly critical microservices in especially hardened containers and monitor them in depth.
- Handle trust by specialised hardware, a root point, which holds container images, cryptographic material, registries and any critical information.
- Enforce separation of duties (which involves access control) and segregation of traffic and roles between services and applications.

4 A Secure Reference Model

Given all considerations so far, we now attempt a modification of our reference model (Figure 2) in order to embed security – Figure 3.

The changes from the purely functional architecture of Figure 2 reside on three aspects:

- Network elements are inserted in order to apply policies at the network level, from simple traffic rules to deep-packet inspection looking for malicious traffic or drawing intelligence. Policies should also be defined, enforced and verified to segregate inter-service communication and access. The token verification checks the validity of the access token rather than fully trusting the gateway, and policies can define the access rights of the token. This is a mitigation against potential vulnerabilities arising from the gateway being a confused deputy if compromised.
- A subsystem of monitoring, testing and verification is added. These components should interface directly the instrumentation components at the microservice level (represented by gears). These agents are, ideally, an integral part of the skeleton of any service and should be enrichened with service-specific metrics. The containers themselves are to be monitored but one also expects support from the underlying OS.
- A root of trust supports bootstrapping processes by holding containers images, cryptographic material, and configurations. This is used to ensure authenticity of software components and security configurations.

Any request from the outside world must pass through a Firewall and IDS, container firewalls should inspect requests from the gateway or any potential internal traffic. Access tokens should also be verified for authenticity at the microservices level and processed for access control by microservices policy rules. Further, Every critical part of the system should be systematically monitored and container images and configurations must be validated against trusted hardware and software images.

5 Conclusions and Outlook

Microservices is a powerful and promising paradigm for distributed applications that, nevertheless, present security challenges on its own. Whereas current technologies and techniques are directly applicable, others need to be developed and adopted in order to reach the needed level of security maturity. In this article we provided a comprehensive discussion of Security for Microservices by looking at different angles and, wherever possible, reusing current practices. It is clear that Microservices still need to mature at different levels such as, for example, the lacking or unavailability of specialised elements (such as firewalls or IDSes) that are aware of its specificity. Coexistence and multitenancy is also a challenge in terms of security since many services are expected to run on the same hardware and need isolation. Another direction for future work concerns availability of industry guidance of which the recent NIST draft is a good example.

References

[1] O. Zimmermann, "Microservices tenets: agile approach to service development and deployment," *Computer Science-Research and Development*, vol. 32, no. 3, pp. 301–310, 2016.

[2] D. I. Savchenko, G. I. Radchenko, and O. Taipale, "Microservices validation: Mjolnirr platform case study," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*. IEEE, 2015, pp. 235–240.

[3] M. Fowler and J. Lewis, "Microservices," *ThoughtWorks. http://martinfowler. com/articles/microservices. html*, 2014.

[4] J. Th¨ones, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, 2015.

[5] M. Fowler, "Microservices prerequisites," *Martin Fowler*, 2016.

[6] C. Fetzer, "Building critical applications using microservices," *IEEE Security & Privacy*, vol. 14, no. 6, pp. 86–89, 2016.

[7] M. Souppaya, J. Morello, and K. Scarfone, "Application container security guide," *NIST Special Publication*, vol. 800, p. 190, 2017.

[8] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Soft-*

ware Engineering. Springer, 2017, pp. 195–216.

[9] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *Computing Colombian Conference (10CCC), 2015 10th*. IEEE, 2015, pp. 583–590.

[10] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*. IEEE, 2016, pp. 44–51.

[11] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger, "Performance engineering for microservices: Research challenges and directions," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ACM, 2017, pp. 223–226.

[12] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[13] A. Sill, "The design and architecture of microservices," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 76–80, 2016.

[14] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[15] A. Karmel, R. Chadromouli, and M. Iorga, "Nist definition of microservices, application containers and system virtual machines," *Natl Inst. of Standards and Technology (NIST) Special Publication*, pp. 800–180, 2016.

[16] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[17] S. Patanjali, B. Truninger, P. Harsh, and T. M. Bohnert, "Cyclops: a micro service based approach for dynamic rating, charging & billing for cloud," in *Telecommunications (ConTEL), 2015 13th International Conference on*. IEEE, 2015, pp. 1–8.

[18] T. Saito, Y. Tsunoda, D. Miyata, R. Watanabe, and Y. Chen, "An authorization scheme concealing client's access from authentication server," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2016 10th International Conference on*. IEEE, 2016, pp. 593–598.

[19] Y. Sun, S. Nanda, and T. Jaeger, "Security-as-a-service for microservices-based cloud applications," in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*. IEEE, 2015, pp. 50–57.

[20] J. Gummaraju, T. Desikan, and Y. Turner, "Over 30% of official images in docker hub contain high priority security vulnerabilities," Technical report, BanyanOps, Tech. Rep., 2015.

Antonio Nehme is a PhD Student at Birmingham City University
He received a B.S. in Computer Science with a minor in Mathematics from the Lebanese American University. His research interests include microservices security, threat modelling, identity management, trust and risk management.

Vitor Jesus received a degree in Physics in 2000 from University of Coimbra, Portugal, and a MSc (2007) and a PhD (2012) in Computer Science and Networks from University of Aveiro, Portugal, with co-supervision from Carnegie Mellon University. He is currently a Lecturer at the School of Computing of Birmingham City University, Birmingham, UK. He has held several positions in small and large companies in the fields of Data Analytics, Networks and Security. He holds Security and Data Privacy certifications. His research interests are in the areas of Future Internet, IoT/IIoT, CyberSecurity, Data Privacy and Blockchain applications.

Khaled Mahbub (PhD, MEng, BEng) is a Senior Lecturer in Software Engineering at Birmingham City University. His research interests are in the area of Automated Software Engineering, focusing on some key technical issues for the effective realization of service based systems and cloud based systems, including run-time security monitoring and secure system design. In the past Khaled has worked in several EU funded projects including, CUMULUS (Certification Infrastructure for Multi-Layer Cloud Services, EU FP7, STREP Project), ASSERT4SOA (Advanced Security Service cERTificate for SOA, EU FP7, STREP Project), S-CUBE (The Software Services and Systems Network - FP7 EU Project), Gredia (Grid enabled access to rich media content - FP6 EU Project), Serenity (System Engineering for Security & Design, FP6 EU Project), SeCSE (Service Centric System Engineering, FP6 EU Project). He has published more than 30 papers in different international journals and conference proceedings with more than 800 citations.

Ali E. Abdallah BSc, MSc, DPhil(Oxon) Professor of Information Security School of Computing and Digital Technology in Birmingham City University. He leads information security research at BCU focusing on topics ranging from identity management, access control and privacy to securing shared information in virtual organisations and the development of secure and resilient software.