A New Paradigm to Address Threats for Virtualized Services

S. Covaci
Institute for Telecommunication Systems
Technical University of Berlin
Berlin, Germany
Email: stefan.covaci@tu-berlin.de

R. Rapuzzi, M. Repetto
S^{3}ITI Lab
CNIT
Genoa, Italy
Email: matteo.repetto@cnit.it

F. Risso
Dept. of Control and Computer Engineering
Politecnico di Torino
Turin, Italy
Email: fulvio.risso@polito.it

***Abstract*—With the uptaking of virtualization technologies and the growing usage of public cloud infrastructures, an ever larger number of applications run outside of the traditional enterprise's perimeter, and require new security paradigms that fit the typical agility and elasticity of cloud models in service creation and management. Though some recent proposals have integrated security appliances in the logical application topology, we argue that this approach is sub-optimal. Indeed, we believe that embedding security agents in virtualization containers and delegating the control logic to the software orchestrator provides a much more effective, flexible, and scalable solution to the problem.**

In this paper, we motivate our mindset and outline a novel framework for assessing cyber-threats of virtualized applications and services. We also review existing technologies that build the foundation of our proposal, which we are going to develop in the context of a joint research project.

I. Introduction

Cloud management software typically provides isolation among multiple tenants by running applications in virtual machines or software containers, and by connecting them through overlay networks (e.g., VLAN, VXLAN, GRE tunnels). In addition, basic packet filtering (i.e., firewalling) is usually available to control incoming/outgoing traffic. Motivated by the lack of common and uniform Security-as-a-Service APIs in cloud management software and the substantial untrustworthiness of the underlying infrastructure, especially with the expected advent of distributed edge technologies [1], security is now starting to be seen as integral part of the design of cloud applications.

Recent efforts in software design for cloud services have investigated new paradigms beyond service-oriented architectures and web services, which pursue more automation in developing, deploying, and managing applications. The purpose is to define high-level programming abstractions that describe each application as a logical "graph" or "chain" of elementary components (also referred to as micro-services or virtual functions), while smart engines provision cloud resources and automatically deploy and manage service graphs [2]. TOSCA [3], ETSI MANO [4], and IEEE SFC [5] are currently the main orchestration models used in the cloud and network function virtualization domains.

The modular architecture of virtualized services implicitly suggests that virtual instances of security appliances might be themselves software components to be included in the service graph, either at design time or at deployment time (in this case, by including proper security policies to be interpreted by the orchestrator). However, we argue that this approach suffers some limitations in terms of scalability, effectiveness, efficiency, integration, and security. In this paper, we propose a new paradigm that de-couples inspection tasks from the detection logic; the former to be integrated into the different forms of virtualization containers, and the latter to be part of the orchestration process and directly interacting with application management to provide situational awareness and support quick reaction and mitigation actions.

The rest of the paper is organized as follows. Section II briefly reviews the current approach and its limitations. We describes our concept in Section III, together with a preliminary architecture for building situational awareness of virtualized applications. We also discuss enabling technologies in Section IV, and point out open challenges and research issues for the implementation of the devised architecture in Section V. Finally, we give our conclusions and plans for future work in Section VI.

II. State of the Art and motivations for a new paradigm

Cloud management software provides basic isolation between different tenants and distributed firewalling;clearly, this is a very rough approach and requires service owners to apply security appliances within each sandbox, by integrating them in the design of their service graph, in order to inspect software, network traffic, user and application behavior.

Since, on first approximation, virtualization environments could be viewed as special instances of physical networks, more and more vendors are now offering software versions of their security appliances (Intrusion Prevention/Detection Systems, Firewalls, Antivirus, Network Access Control, etc.), mostly for data centers and virtualized IT environments, which simplify deployment and re-configuration. This approach demands for tight integration between service design and op-

The final publication is available at IEEE Xplore via https://doi.org/10.1109/COMPSAC.2018.10320.

eration, which is not always simple to achieve in current development processes; to address elasticity and autonomicity, several research projects are developing policies frameworks and orchestration models to integrate security appliances into service graph design [6].

Unfortunately, virtual environments have peculiar characteristics that point out some important limitations of current approaches. First, hypervisors and overlay networks provide isolation, but are not immune to attacks. DoS attacks against the physical network affect all virtual networks of all tenants, while a compromised hypervisors is a potential source of eavesdropping and alteration for every hosted virtual machine or software container [7]. Second, software-based security appliances do not benefit from hardware acceleration. Complex multi-vector attacks are evolving inspection from memoryless simple string matching to stateful rules (such as regular expressions) [8], hence more processing power is required, which is likely to overwhelm software-based implementation of load balancers, firewalls, and intrusion prevention systems, especially in case of large volumetric attacks.Third, security appliances run in the same type of virtualization environment (e.g., virtual machines, containers) of the actual services; hence they are also vulnerable to attacks and possibly increase the attack surface of the whole application. Fourth, programmers and service developers are not usually security experts, since security is usually managed by operation staff. Integrating security appliances in graph design may lead to weak or ineffective protection [1], giving false trust confidence to service users.

III. Basic concept and approach

We envisage a new approach to assess situational awareness of virtualized services and effectively support quick remediation actions, beyond mere integration of security appliances in service graphs. The main concept is the disaggregation of cyber-security appliances into business logic (i.e., detection algorithms) and data plane (i.e., monitoring and inspection tasks), mediated by orchestration logic and proper security models.

Fig. 1 depicts the main differences between current approaches and the new concept. Instead of overloading the execution environment with complex and sophisticated threat detection capabilities, efficient processing capabilities are provided in the execution environment that create events and knowledge; algorithms for detection of threats and vulnerabilities are moved upwards and process such data in a coordinated way for the whole execution environment.

To implement the above concept, we envision the multi-tier architecture shown in Figure 2. Multiple *programmable hooks* are present in the virtualization container (in the OS kernel, in system libraries, and in the micro-service code), to monitor what happens inside. Programmable hooks include logging and event reporting capability developed by programmers into their software, as well as monitoring frameworks built in the kernel and system libraries that inspect network traffic and system calls. The *security model* logically decouples programmable hooks from orchestration. It uses specific semantics to describe security-related capability, e.g., logging, event reporting, filtering, deep packet inspection, system call interception. Through the security model, *orchestration* knows what kinds of operations can be carried out on each component, collects data and measurements, and feeds the layers above. The management framework includes the definition of policies and detection algorithms. *Policies* describe in an abstract form life-cycle management; for example, what kind of actions should be undertaken upon detection of potential attacks. Orchestration provides a sort of adaptation that abstracts the heterogeneity of deployed components, protocols, and interfaces to the overall management framework. Finally, *algorithms* analyze and correlated information provided by orchestration at graph level to detect threats, anomalies, vulnerabilities, attacks.

The security model and the policies are used to take deployment, placement, and (re-)configuration decisions, hence shaping the system behavior according to the evolving context. For instance, orchestration takes into consideration both user-defined policies and output from detection algorithms to configure monitoring and executive processes. This means that packet filters, types and frequency of event reporting, level of logging is selectively and locally adjusted to retrieve the exact amount of knowledge, without overwhelming the whole system with unnecessary information. The purpose is to get more details for critical or vulnerable components when anomalies are detected that may indicate an attack, or when a warning is issued by cyber-security teams about new threats and vulnerabilities just discovered. This approach allows lightweight operation with low overhead when the risk is low, even with parallel discovery and mitigation, while switching to deeper inspection and larger event correlation in case of anomalies and suspicious activities, hence being able to properly scale with the system complexity, even for the largest services (e.g., carrier's large scale virtual networks, and worldwide mass applications as social nets).

IV. A framework for addressing cyber-threats of virtualized applications

The multi-layer architecture described in Section III represents the foundation for addressing cyber-threats for virtualized applications. Fig. 3 sketches a framework for deployment of virtual applications and management of security aspects. We identify three main logical blocks:

- *service engineering*, concerning the development and modeling of software components (micro-services, virtual functions) and service graphs;
- *service management*, dealing with secure deployment and life-cycle management of service graphs;
- *situational awareness*, responsible for detecting threats and certifying data for security audits and court investigations.

Service engineering is based on the usage of metadata both at the component and service graph level. Meta-data include information about characteristics, dependencies on other components (e.g., web application that needs a database

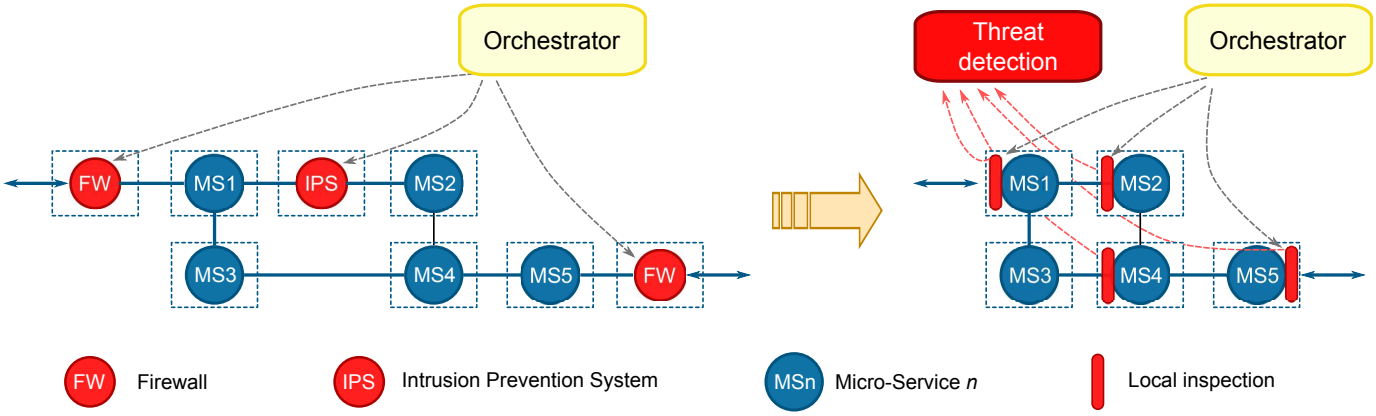

Fig. 1: Our approach entails a transition from in-service embedding of security appliances to a multi-layer architecture tightly integrated with orchestration logic.

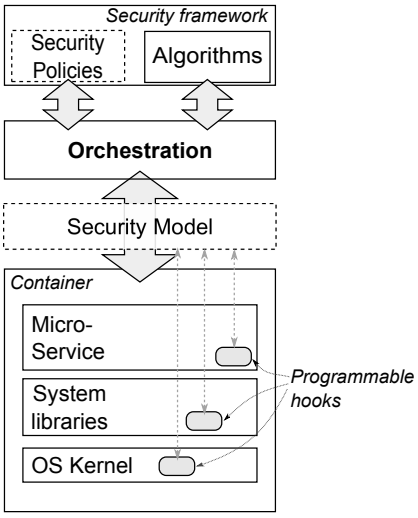

Fig. 2: Conceptual multi-layer architecture.

for storage), requirements on computing/networking/storage resources (e.g., number of CPUs, RAM, bandwidth), etc. They also contain hooks for lifecycle management operations that can be invoked on the component: start/stop/restart the service, clone the service, de-provision the service, configure the service after deployment, monitor performance and execution, and so on. Orchestration makes use of this information for deployment, by adapting the service and its components to the changing context. According to this interpretation, meta-data are indicated as the *context model*. In a similar way, life-cycle management is driven by *policies*, which are described by a condition/action pattern.

The context model includes specific information about security-related capabilities, including what kinds of logs, events, and behavioral/traffic monitoring can be generated or consumed by the micro-service, the hooks for configuring and controlling the generation of such data, and trust and certification mechanisms to seal information with legal validity for forensics investigation.

Policies are sets of operating rules, usually in the form 'on event, if condition, then action', that describe life-cycle management operations, so that orchestration can shape the system behavior to the evolving context without altering the system implementation. Specifically, security policies (defined in Section III) reflect the resource owner's intention of adequately protecting valuable resources. A list of security policies include but is not limited to: deployment constraints (trustworthiness and security of the underlying infrastructure and the other chainable components in the graph), re-configuration of individual components to change their reporting behavior, and re-action to threats and attacks.

Service management entails various life-cycle operations on the service graph, in addition to policies for automated tasks. Our framework also envisions authentication and encrypted channels for interacting with the service components. Thus, management is mediated by ABAC (Attribute-Based Access Control) and ABEC (Attribute-Based Encription Control). It also contains an IdM (Identity Management) component and a PKI infrastructure (rooted at a trusted and public Certification Authority). Secure deployment also entails selection of trusted services, hence a TSL (Trusted Service List) component is present.

Finally, situational awareness represents a totally new functional block in frameworks for developing and deploying cloud applications. It includes all the components to collect and process security-related data, and to provide knowledge and evidence about cyber-security threats, vulnerabilities, and attacks. The *context broker* collects context information, likely by a Pub/Sub or similar paradigm, and feeds other engines that process and store such information. *Threat intelligence* is a collection of detection algorithms that analyzes events, data, and logs, arguably by combining innovative detection methodologies (rules-based, machine learning) with big data techniques; the purpose is to locate vulnerabilities in the graph and its components, to identify possible threats, and to timely detect on-going attacks. The target is protection from both software vulnerabilities and network threats, hence

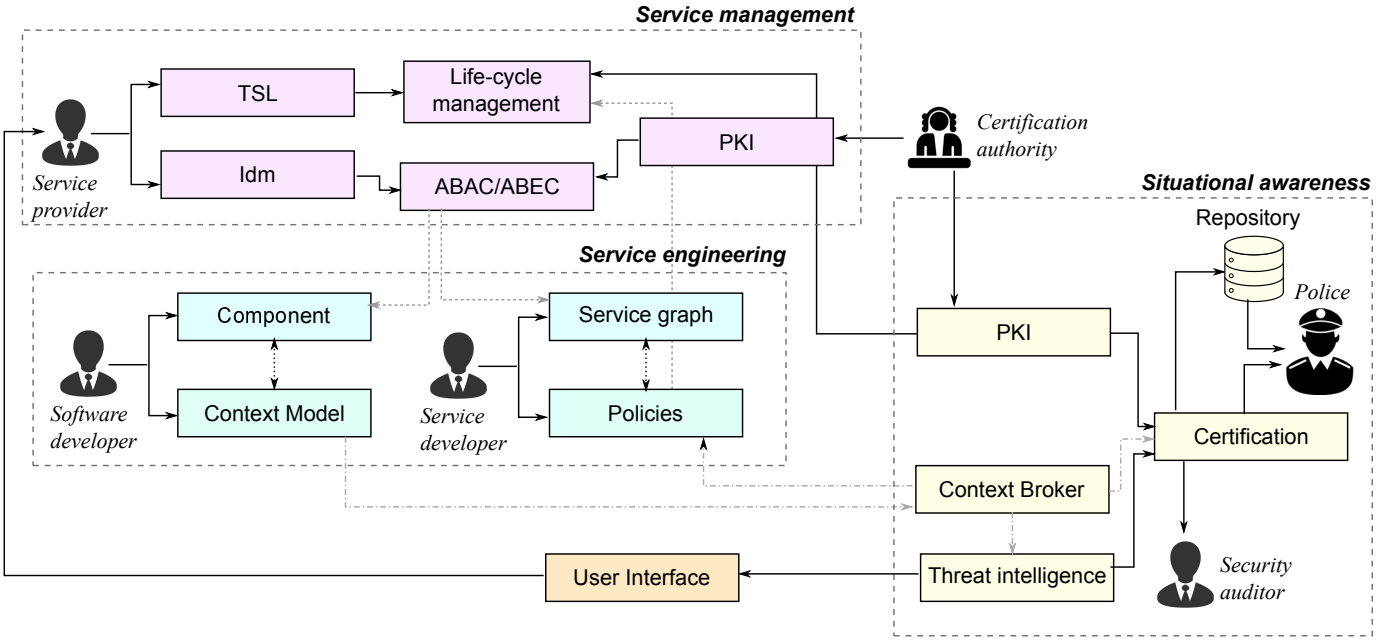

Fig. 3: Conceptual architecture of the framework for addressing cyber-threats of virtualized services.

involving a mix of source and run-time code analysis, formal verification, network analytics, and packet filtering techniques. We remark that, since orchestration manages all service graphs of the same service provider, threat intelligence combines and correlates context information from different graphs, which further improves threat detection and brings the possibility to fix vulnerabilities in advance before other components get compromised. Knowledge built by the detection logic is shared with other processes:

- the *user interface*, to provide proper visual representation of the current situation to the service provider, and to assist him in taking decision for remediation actions and countermeasures;
- the *certification process*, which is responsible for origin, timestamping, digital signing, integrity of relevant information that is used for security audits and legal interception; the solution should be able to capture enough information to trace security attacks in a reliable manner and to interpret the data post-factum;
- the *secure repository*, which conserves data with legal validity (security audit trails) for forensics investigation that is initiated after the threat or attack has been identified.

V. Open challenges and research issues

Beyond the main functions envisioned by the conceptual architecture in Fig. 3, a deeper definition is necessary to clearly define what to capture, where to store, how to access, and how to search. In this Section we review open challenges and research issues for the main functional blocks of the proposed framework.

A. Orchestration

If security functions were part of service graph design, they would be basically treated as all other software components, with no appreciable results in tackling the specific challenges of virtual environments (see Section II). For example, the missing part of all the NVF-based architectures is that the "clones" which are deployed for each tenant are not properly customized for the usage resulting into very uniform security deployments, not appropriate for the service usage. Indeed, true and effective integration of security in service orchestration requires a new approach that addresses at least the following needs:

- checking the trustworthiness of software and service graphs at deployment time;
- adapting the service graph to the evolving security context – e.g., replace compromised or vulnerable components with equivalent (but better) ones, inject new functions or disable existing ones;
- triggering software analysis at run-time periodically or after suspicious events;
- translate policies and high-level instructions into proper code and configuration at the container/micro-service level – e.g., setting/changing firewalling rules, monitoring and deep packet inspection, forwarding and routing policies;
- ensure, though formal models and methods, the correct implementation of security policies.

Beyond adaptation between algorithms and the underlying security mechanisms, orchestration plays a crucial role in checking correctness, congruence, and consistence of security policies and actions. A limitation of all current techniques is that they are not integrated into the orchestration process, but

they act either before it (on the user-specified service graph) or as a post-processing step after orchestration. This is not the best solution. An early check, in fact, may miss security problems introduced afterwards, while with a later check, if errors are detected by the verifier, service deployment fails because the orchestrator does not have clues about how to fix the errors or the orchestrator has to iterate through the many possible solutions, which is clearly inefficient.

It is therefore necessary the development of formal approaches that, while providing final assurance levels similar to the ones of the state-of-the-art formal verification techniques, are incorporated into the secure orchestration process, which in this way produces network configurations that, once deployed into the underlying infrastructure, are formally guaranteed to satisfy the required security policies. Ensuring the correctness of service orchestrators has already been recognized a crucial problem in security-critical cloud computing environments, so that the idea of formally verifying orchestration procedures has been recently proposed (e.g., [9]–[11]), for verifying cloud-related policies (e.g., verify that payment procedures are properly designed). Future extensions should also address networking aspects, in particular for NFV services that steer packets across different virtual functions.

B. Monitoring, inspection, enforcement

Many network security appliances (typically for DoS protection) already collect traffic information and measurements from network devices, originally by flow collectors like NetFlow, sFlow, and IPFIX, and, more recently, by also integrating software-defined protocols (OpenFlow and NetConf). In addition, the concept of distributed firewall has recently been proposed for virtualization environments, to integrate packet inspection and filtering in hypervisors [12]. A distributed firewall removes the need for traffic steering (all network packets go through the hypervisor, which is part of the firewall) and IP-based rule structures (through the notion of logical "containers" or "security groups"). These kinds of approaches are largely used in commercial and open-source products for analyzing traffic flows and enforcing filtering rules, but does not have the flexibility to provide deep and flexible inspection of network packets and system calls tailored to the specific needs for detecting a broader class of threats and attacks.

As the data plane is not intended only as the transport layer for network traffic, the above monitoring/inspection/enforcing elements should operate on a larger extent of information such as any data that is actually being generated in the virtualized environment, such as kernel-level system calls, disk I/O, and more, hence offering a broader and more precise coverage of what happens in the system under control. In this respect, the IOVisor technology [13] offers a wider range of options, including both in-kernel network eXpress Data Paths (XDP), enhanced Berkeley Packet Filters (BPF), and analysis of system calls. However, current IOVisor technology has been validated mostly with monitoring applications, hence with limited (or no) capabilities to perform more effective actions (e.g., data modification/manipulation) on the incoming data. Furthermore, only simple data plane programs are allowed, i.e., without support for complex programs created according to the split data/control plane paradigm as originally proposed with SDN/OpenFlow. It is therefore necessary to extend this technology (i) to support more powerful programs, which can operate according to the split data/control plane paradigm; (ii) to support more powerful actions on the data in transit, which enable to implement some proactive security actions (e.g., drop network traffic, modify packet information, craft ad-hoc packets for specific purposes) that go beyond simple monitoring.

C. Threat detection

Fine grained monitoring and inspection capabilities as described in Section V-B would not only be suitable for volume anomaly detection that processes raw flow information [14], but would also support effectively signature- and rule-based detection similarly to existing IPS/IDS tools [7], [15]. Different techniques may be used for these purposes: regression analysis, predictive and prescriptive analytics, data mining and machine learning. Obviously, a larger base of data and events would increase the processing burden, but this should not be a problem, since the control plane is outside the service graph and could run in dedicated infrastructures with big data techniques.

From a conceptual point of view, virtual services constitute a "hackable" network of services; thus, a continuous internal audit of their security is required. Both static and dynamic analysis of code are required to protect virtual services during their lifecycle. Static code analyzers use flow-insensitive and interprocedural constraint-based analysis to extract a vulnerability detection model from the source code targeting the general class of buffer-related vulnerabilities. This can be applied to the detection of vulnerability types such as buffer overflow, format string attack, and code injection [16]–[18]. Even though there are many static analyser tools there is a clear lack of code assessment mechanisms that targets specifically cloud applications and virtual network functions. Dynamic tools fall mainly into two categories: dynamic and concolic analysis systems. Dynamic analysis systems [19], such as "fuzzers", monitor the native execution of an application to identify flaws. Concolic execution engines [20], [21] utilize program interpretation and constraint solving techniques to generate inputs to explore the state space of the binary, in an attempt to reach and trigger vulnerabilities. Here the challenge is to develop hybrid vulnerability analysis tools that leverage packet fuzzing, packet sniffing and selective concolic execution (some of the prominent vulnerability assessment techniques) in a complementary manner, to find deeper (possible) bugs during the execution of the services and their components. By using techniques such as packet sniffing and fuzzing, a set of the most severe attacks can be investigated in order to study their after-effects and help identify sufficient mitigation actions.

D. Legal and forensics investigation

Even the most reliable system may occasionally be compromised, and in this case it is important to investigate the cause to identify additional protection measures. In this respect, a critical issue is the legal validity of the extracted data to prosecute attackers. Common challenges in this area include: *i)* storing trusted evidence, *ii)* respecting the privacy of users when acquiring and managing evidence, *iii)* preserving the chain of custody of the evidence. We remark that in the proposed framework the problem is not the same as the definition of *Cloud forensics* [22], [23], since investigation in our case is carried out by the service owner and not by the cloud provider.

VI. Conclusions

In this paper, we have outlined the background concept and the overall approach that will be followed in the XXXXX project for managing the cyber-security of virtualized applications. The framework will be based on a multi-layer architecture that decouples monitoring and inspection tasks (data plane) from the detection and reaction logic (control plane).

We believe that our proposal brings far more dynamicity, scalability, robustness, and self-adaptability than current practice of inserting virtual instances of security appliances in the application topology. Our belief relies on more elasticity brought by the intermediate orchestration layer, more effectiveness in monitoring and processing in the data plane, reduced overhead on graph execution and attack surface by avoiding additional security appliances, and more adaptability brought by programmable infrastructures.

Acknowledgment

The work has been supported in part by the European Commission, under grant no. 786922 (ASTRID Project).

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, no. 2, pp. 680–698, January 2018, in press.

[2] J. Wettinger, U. Breitenbücher, and F. Leymann, "Standards-based DevOps automation and integration using TOSCA," in *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, London, UK, Dec. 8–11, 2014, pp. 59–68.

[3] "Topology and orchestration specification for cloud applications," OASIS Standard, November 2013, version 1.0. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

[4] "Network functions virtualisation," ETSI ISG NFV whitepaper #3, October 2014. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper3.pdf

[5] J. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," RFC 7665, October 2015. [Online]. Available: https://tools.ietf.org/rfc/rfc7665.txt

[6] T. Quang Thanh, S. Covaci, T. Magedanz, P. Gouvas, and A. Zafeiropoulos, "Embedding security and privacy into the development and operation of cloud applications and services," in *17th International Telecommunications Network Strategy and Planning Symposium*, Montreal, QC – Canada, Sep. 26th–28th, 2016, pp. 31–36.

[7] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, January 2013.

[8] European Cyber-Security Organisation, "European cybersecurity Strategic Research and Innovation Agenda (SRIA) for a contractual Public-Private Partnership (cPPP)," •, June 2016. [Online]. Available: http://www.ecs-org.eu/documents/ecs-cppp-sria.pdf

[9] W. Chareonsuk and W. Vatanawood, "Formal verification of cloud orchestration design with TOSCA and BPEL," in *International Conference on Electrical Engineering/Electronics Computer, Telecommunications and Information Technology (ECTI-CON 16)*, Chiang Mai, Thailand, Jun., 28th–Jul.1st, 2016.

[10] H. Yoshida, K. Ogata, and K. Futatsugi, "Formalization and verification of declarative cloud orchestration," in *Formal Methods and Software Engineering*, ser. Lecture Notes in Computer Science, B. M., C. S., and Z. F., Eds. Springer, Cham, 2015, vol. 9407, pp. 33–49.

[11] A. F. and M. F., "Exploiting cloud and workflow patterns for the analysis of composite cloud services," *Future Generation Computer Systems*, vol. 67, pp. 255–265, February 2017.

[12] B. Hedlund, "What is a distributed firewall?" Post on vmware blogs – Network virtualization, July 2013. [Online]. Available: https://blogs.vmware.com/networkvirtualization/2013/07/what-is-a-distributed-firewall.html

[13] "IOVisor project – advancing in-kernel io virtualization by enabling programmable data planes with extensibility, flexibility, and high-performance," Web site, Aug 2017. [Online]. Available: https://www.iovisor.org

[14] H. Kasai, W. Kellerer, and M. Kleinsteuber, "Network volume anomaly detection and identification in large-scale networks based on online time-structured traffic tensor tracking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 636–650, September 2016.

[15] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, January 2013.

[16] G. Chatzieleftheriou, A. Chatzopoulos, and P. Katsaros, *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications. ISoLA 2014*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2014, vol. 8803, ch. Test-Driving Static Analysis Tools in Search of C Code Vulnerabilities II, pp. 486–488.

[17] L. Xin and C. Wandong, "A program vulnerabilities detection frame by static code analysis and model checking," in *IEEE 3rd International Conference on Communication Software and Networks (ICCSN)*, Xi'an, China, May, 27th–29th, 2011, pp. 130–134.

[18] A. Armando, G. Bocci, G. Chiarelli, G. Costa, R. De Maglie, G. Mammoliti, and M. Alessio, *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2015, vol. 9035, ch. SAM: The Static Analysis Module of the MAVERIC Mobile App Security Verification Platform, pp. 225–230.

[19] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos, "Dowsing for overflows: A guided fuzzer to find buffer boundary violations," in *22nd USENIX Security Symposium*, Washington, D.C. – USA, Aug., 14th-16th, 2013, pp. 49–64.

[20] V. Chipounov, V. Kuznetsov, and G. Candea, "S2e: A platform for in-vivo multi-path analysis of software systems," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS XVI)*, Newport Beach, California – USA, Mar., 5th–11th, 2011, pp. 265–278.

[21] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing mayhem on binary code," in *IEEE Symposium on Security and Privacy*, San Francisco, California – USA, May, 20th-25th, 2012, pp. 380–394.

[22] K. Ruan, J. Carthy, T. Kechadi, and M. Crosbie, "Cloud forensics: An overview," in *Proceedings of the 7th IFIP International Conference on Digital Forensics*, Orlando, FL – USA, Jan. 31st – Feb. 2nd, 2011, pp. 35–46.

[23] D. Barrett and G. Kipper, *Virtualization and Forensics – A Digital Forensic Investigator's Guide to Virtual Environments*, 1st ed. Elsevier, May 2010.