Towards Secure Cloud Orchestration for Multi-Cloud Deployments

Paladi, N., Michalas, A. and Dang, H.

This is an electronic version of a paper presented at the *5th Workshop on CrossCloud Infrastructures & Platforms,* Porto, Portugal, 23 to 26 April 2018.

Towards Secure Cloud Orchestration for Multi-Cloud Deployments

Nicolae Paladi
RISE SICS
nicolae.paladi@ri.se

Antonis Michalas
Tampere University of Technology
antonios.michalas@tut.fi

Hai-Van Dang
University of Westminster
H.Dang@westminster.ac.uk

Abstract

Cloud orchestration frameworks are commonly used to deploy and operate cloud infrastructure. Their role spans both vertically (deployment on infrastructure, platform, application and microservice levels) and horizontally (deployments from many distinct cloud resource providers). However, despite the central role of orchestration, the popular orchestration frameworks lack mechanisms to provide security guarantees for cloud operators. In this work, we analyze the security landscape of cloud orchestration frameworks for multi-cloud infrastructure. We identify a set of attack scenarios, define security enforcement enablers and propose an architecture for a security-enabled cloud orchestration framework for multi-cloud application deployments.

Categories and Subject Descriptors Security and privacy [*Systems security*]: Distributed systems security

Keywords Orchestration, cloud infrastructure, microservices, virtualization, security.

1. Introduction

Cloud infrastructure deployments became larger and more elaborate in the two decades since the cloud-computing paradigm emerged. The growth in the number of intercommunicating components, variety of supported application programming interfaces (APIs), and additional mechanisms to enable flexible scalability and computation efficiency led to a remarkable increase in complexity. As a result, *cloud orchestration* became essential at all stages of the cloud infrastructure lifecycle.

The role of cloud administrators gradually shifted from manually deploying, configuring and monitoring cloud infrastructure components to configuring orchestrator frameworks (or simply, orchestrators) and deployment templates. This includes writing configuration policies that describe what the orchestrator *should do* to realize the cloud infrastructure, or expressing high-level imperative intents describing how the infrastructure *should look like* and leaving the implementation details to the orchestrator. Beyond deployment functionality, modern orchestrator systems also include monitoring, load balancing and continuous workload deployment functions, supporting the cloud infrastructure lifecycle. Automated orchestrators allow to maintain a stable, continuous, and highly available set of cloud services with minimal or no human interference, based solely on configuration policies or *intents*.

While cloud operators leverage orchestration capabilities to set up and operate cloud infrastructure, adversaries can leverage misconfigured or maliciously modified orchestrators, as well as forged configuration policies and intents, to conduct attacks on cloud infrastructure. For example, adversaries can exploit orchestrator vulnerabilities to disrupt the generation of credentials provisioned to virtual machine (VM) instances; insert backdoors in VM images to automate the deployment of maliciously modified VM instances; or take control over the placement policy of virtual resources, either to benefit a service provider (in case of federated deployments) or deploy virtual components on hardware resources with compromised physical security.

We aim to outline a security architecture for cloud orchestrators, that addresses the current attack vectors in cloud infrastructure deployment. The proposed security architecture aims primarily to address the attack vectors in existing orchestrators; however, it is extendable to make use of the recent evolution in the field of micro-services, trusted computing and lightweight virtualization. In particular, with regard to lightweight virtualization, the emerging unikernels [14] may present a more secure alternative to container-based (hypervisor-free) approaches, as application developers have explicit control over core security areas. Finally, logical centralization of orchestrator systems is another important aspect at the crossroads between functional and security architectures: decentralized orchestration needs careful consider-

ation with regard to discovery, synchronization, coordination and security aspects of cloud application agents.

To prevent malicious actions in increasingly automated and autonomous cloud deployments, cloud orchestrators must include built-in mechanisms to validate inputs and component behavior against predefined security policies, verify the integrity of deployed software and take corrective measures in case of deviations. However, while popular orchestrator systems include some security features, none of them implements cloud orchestration with a comprehensive set of security mechanisms.

We address this by proposing a security architecture for federated infrastructure orchestration. We base the security architecture on Occopus, a multi-cloud orchestrator to deploy and manage complex infrastructures [11]. We further extend the cloud orchestrator security architecture to the application level, based on MiCADO, a microservice-based cloud application-level dynamic orchestrator [10].

1.1 Contribution

The contribution of this paper is twofold. First, we address a gap in existing literature by analyzing the security landscape of cloud orchestration. Based on an extensive security review of current orchestration approaches, we outline the potential risks that must be considered when designing cloud orchestrators. We use the analysis of existing security issues and identified risks to define a threat model specifically addressing the requirements of cloud orchestrators. Based on the identified risks and on the introduced threat model, we describe a set of security and privacy-preserving mechanisms for cloud orchestrators and an initial security architecture.

1.2 Organization

The remainder of this paper is organized as follows. We review related work in Section 2, discuss two main cloud orchestration paradigms in Section 3 and introduce the threat model in Section 4. Next, we present and motivate a set of orchestration security enablers in Section 5. In Section 6 we propose an orchestration security architecture using the security enablers. We conclude in Section 7.

2. Related Work

Security of cloud orchestration and cloud bursting was investigated earlier, as multi-cloud deployments became increasingly popular. Nair et al. [16] identified a set of security concerns for orchestration in multi-cloud deployments. These include data privacy, confidentiality and integrity; security integration with existing infrastructure; and platform vulnerabilities. Although the concerns are relevant for the available cloud orchestrators, the analysis does not describe a suitable threat model and does not outline any solutions. We complement this work by extending the security risks with a set of attack scenarios and respective security enablers.

Pawar et al. [21] present a trustworthiness assessment framework for a multi-cloud orchestration and and broker system. The assessment uses an empirical approach and assesses compliance with service level agreement parameters, service provider satisfaction ratings, and service provider behaviour. However, the framework does not include security guarantees on either service provider or resource component level. We address this gap through several cloud orchestration security enablers performing virtualization platform attestation and virtual image container verification.

Weerasiri et al. [24] lists security as one of the cross-cutting concerns in cloud orchestration implementations, along with service level agreements and negotiations, portability, interoperability, standardization, resource demand profiling, resource pricing, profit maximizing and other runtime issues. The study, found that such cross-cutting concerns are addressed both by research initiatives, and to a larger extent by proprietary orchestration products, such as: Amazon Web Service OpsWorks, CFEngine, Docker, Heroku, Juju, Puppet, and VMWare vSphere.

3. Orchestration Models

We first review the role and purpose of cloud orchestration, and analyze the two major models of cloud orchestration. In complex systems such as clouds and software-defined network (SDN) deployments, system administrators use orchestrators to dynamically deploy services. Orchestrators are logically centralized entities that manage and coordinate the lifecycles of components constituting the service. For services within one administrative domain, one orchestrator may be responsible for the end-to-end service setup.

Cloud orchestration is used when monolithic legacy applications ported to cloud platforms are replaced by microservice-based dynamic applications. Legacy applications use *fixed layering* and inter-layer invocation through well-defined layer-specific interfaces. Likewise, they are tied to infrastructure resources, such that the same entity owns service features, functionality and physical resources.

In contrast, *dynamic applications* are built from microservices providing a small number of primitive features. They use recursion rather than layering, and functionality is accessed using generic service interfaces reused throughout all layers of the stack. Furthermore, features are decoupled from resources: owners of the features and functionality may be different from the owners of the physical infrastructure. The orchestrator can decide at each step in the decomposition process whether to manage a component itself, or hand off responsibility for that component to a different orchestrator in a different domain, that may in turn use recursive decomposition to deploy that component on its own resources.

The distinction above leads to two types of orchestrators: imperative orchestrators and declarative orchestrators [5, 13]. Imperative orchestrators focus on the deployment approach and expect detailed input about the procedure of deploying services and prescribing the exact set of actions the orchestrator must take. Automation techniques

Table 1. Impact of the orchestration paradigm on orchestrator architecture.

Declarative Architectures	Imperative Architectures
Organizing construct: **recursive decomposition**	Organizing construct: **static layering**
Arbitrary number of levels of recursion	**Fixed** number of layers
Flexible resource layer: resources accessed as services, and services exposed as resources (no architectural distinction)	**Inflexible resources layer**: distinction between resource layer and services layer is baked into the architecture
Identical orchestration functionality at each recursion level	**Layer-specific** orchestration functionality
Interface paradigm based on **negotiation** (request/response) and **delegation**	Interface paradigm based on **management** (higher layers control lower layers)
Interface implementations in a **Domain-Specific Language**	Interface implementations based on **APIs**
Identical Domain-Specific Language on all recursion levels	**Layer-specific APIs**
Federation built-in: North-south and east-west interfaces are the same	**Federation must be added-on**: North-south and east-west interfaces are different

such as "infrastructure as code" are an example of this approach. Declarative orchestrators focus on the intended infrastructure and expect a description of the end-goal of the envisioned service deployment, delegating the interpretation and implementation up to the orchestrator [13]. Table 1, summarizes the comparison between declarative and imperative orchestrators. While imperative orchestrators are still relevant for porting legacy applications to cloud infrastructure, the recent introduction of serverless computing [1, 9] emphasizes the shift towards declarative orchestrators. We address both models, specifically in Section 4, where we outline the threat model considerations for orchestrators.

4. Threat Model

The shift from legacy applications to dynamic applications (both deployed on cloud infrastructure) introduces new security challenges. These include key provisioning, dynamic integrity verification of hosts and system components, as well as integrity (and potentially confidentiality) protection of system configuration data. Furthermore, cloud tenants that own and operate dynamic cloud applications require verifiable guarantees that the infrastructure is deployed according to the specified templates or intents.

We next describe the security threats towards cloud orchestrators. We aim to present a generic overview of the threat model for cloud orchestration; this threat model will be later used to define the orchestrator security architecture.

4.1 Threat Model Assumptions

We base the threat model for cloud orchestration on related models defined in [17, 18, 20]. We first outline the assumptions underpinning the orchestration threat model.

Hardware Integrity Media revelations highlighted the issue of hardware tampering en route to deployment sites [8]. We assume that cloud providers take necessary technical and non-technical measures to prevent such hardware tampering.

Physical Security We assume physical security of the data centers where the Infrastructure-as-a-Service (IaaS) resources are deployed. The assumption applies both for provider-managed resources as well as third-party datacenter capacity, since physical security can be observed, enforced and verified through known best practices.

Low-Level Software Stack We assume that at installation time, IaaS providers reliably record integrity measurements of the low-level software stack: the core root of trust for measurement; BIOS and host extensions; host platform configuration; option ROM code, configuration and data; initial platform loader code and configuration; state transitions and wake events, and a minimal hypervisor. We assume the record is kept on protected storage with read-only access and the adversary cannot tamper with it.

Cryptographic Security We assume encryption schemes are semantically secure and the adversary cannot obtain the plain text of encrypted messages. We also assume the signature scheme is unforgeable, i.e. the adversary cannot forge the signature of the tenant and that the message authentication code (MAC) algorithm correctly verifies message integrity and authenticity. We assume that the adversary, with a high probability, cannot predict the output of a pseudorandom function.

Availability We explicitly exclude denial-of-service (DoS) attacks [15] that aim to disrupt service availability of the infrastructure deployed by the orchestrator platform. DoS can be caused by a wide variety of approaches and is especially difficult to prevent in distributed environments relying on components in different administrative and trust domains.

4.2 Adversary Capabilities

The remote adversary can intercept, drop, inject or otherwise interfere with all network communication.

Network Infrastructure The adversary has physical and administrative control of the network. The adversary is in full control of the network configuration, can overhear, create, replay and destroy all messages communicated between the tenant and their resources (VMs, virtual routers, storage

abstraction components) and may attempt to gain access to other domains or learn confidential information.

4.3 High Level Attacks

We next describe a set of high-level attacks that the adversary can launch by exploiting vulnerabilities in cloud orchestration platforms. The set of high-level attacks is based on earlier research [18, 20] and review articles [3, 7, 22, 24].

VM Substitution Attack The adversary induces the orchestrator to launch a VM instance that contains hidden vulnerabilities, using a maliciously modified VM image instead of the tenant-selected image. In case of success, the adversary can extract sensitive information from the VM instance or monitor the activity of the tenant in the VM instance.

Host Substitution Attack The adversary induces the orchestrator to ignore placement policies regarding host selection, to instantiate the VM on a compromised or vulnerable virtualization host. In case success, the adversary can extract sensitive information from the VM instance or monitor the tenant's activity in the instance.

Storage Host Substitution Attack The adversary induces the orchestrator to ignore data storage placement policies, to attach data storage with exploitable vulnerabilities. In case of success, the adversary can extract sensitive information from the stored data.

Resource Parasite Attack The adversary induces the orchestrator to modify the infrastructure configuration requested by the tenant. In case of success, the adversary can execute hidden parasite processes (such as e.g. crypto currency mining) on the tenant infrastructure [23].

Placement Bias Attack The adversary induces the orchestrator to ignore placement policies in federated cloud deployments to favour a deployment target. In case of success, the adversary can increase the utilization and implicitly the profit of a chosen infrastructure service provider.

5. Orchestration Security Enablers

We next introduce a set of security enablers for secure multi-cloud orchestrators.

Crypto Engine The *Crypto Engine* implements a set of cryptographic algorithms and can be deployed as a separate microservice that supports the following functionality:

- Computing cryptographic hash functions;
- Computing MACs, i.e. Message Authentication Code;
- Key generator;
- Nonce generator.

Credential Manager The Credential Manager (CM) stores the credentials of entities that can access the cloud orchestration service. The CM can receive requests from any entity and is responsible for realizing the corresponding credential in a secure and privacy-preserving way. Furthermore, all credentials managed by the CM are stored such that the CM can only verify their validity, without revealing any other information about the content of the credential.

Firewall Service This service updates the firewall rules of the virtual containers instantiated by the orchestrator according to the security policies defined from user intents or topology descriptions.

PKI Manager The PKI Manager is responsible for issuing and revoking credentials based on a typical challenge-response protocol between the requestor and the issuer.

Attestation Service While hypervisors (or operating systems) enable isolation of virtual execution environments, this is insufficient to establish a trust relationship with the target computing resource, since tenants cannot know whether they are communicating with the intended software or a maliciously modified instance. We use *remote attestation* to address this gap in cloud orchestration security. Attestation of a target is perfomed by an *appraiser*, an entity - generally a computer or a network - making a decision about one or more other cloud resources, known as *targets*. A *target* is a party (for example a computer system) about which an appraiser needs to make such a decision [6]. Communication between an appraiser and a target is conducted through an *attestation protocol*, as defined in [6].

Image Integrity Verifier Corrupted image files are a substantial threat for cloud environments. Image integrity functionality allows the orchestrator to verify the integrity of *virtual container* image files. We adopt a broad definition of virtual containers, that includes virtual machines, lightweight virtualization containers and unikernels but excludes *stateless functions* [1, 9], that rely on process isolation only.

Image Delta Verifier This functionality of the attestation service identifies configuration differences between image files. Concrete implementations include verifying differences between VM images [2] or lightweight virtualization containers [12].

6. Orchestration Security Architecture

We next review the proposed security architecture and describe the interaction of the main components (see Figure 1). We propose a security architecture for orchestrator frameworks under development, such as [10, 11]. However, it can also be adapted to legacy cloud orchestrators.Furthermore, it is important to meniton that in this paper we only focus on security components. For a concrete description of the core components in an orchestrator framework, we refer the interested readers to [10].

6.1 Secure Component Deployment

We aim to enable remote system administrators to securely deploy cloud applications and obtain security guarantees

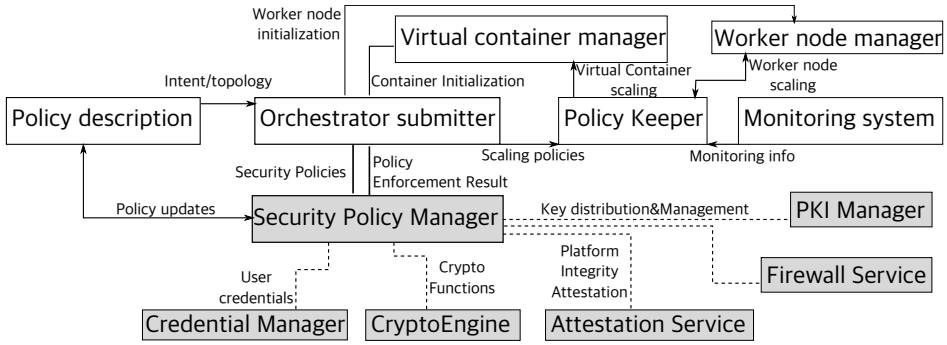

Figure 1. Security Architecture. Security Components in light grey.

about the underlying computing infrastructure. We assume that the underlying master nodes of the platform have been already launched and operating. During infrastructure deployment, the *PKI manager* publishes the public keys of architecture components. The public keys are generated by the *Crypto Engine*, according to administrator-defined security policies defining key length, algorithm selection and entropy requirements. System administrators describe their orchestration *intents* in policy files, using a domain-specific language, such as the Topology and Orchestration Specification for Cloud Applications (TOSCA) [4].

To deploy an application, users first generate a template describing the topology and the security policies of the application. The template file is next sent over a secure communication channel to the *orchestrator submitter*, and protected with the public key of the orchestrator submitter. Upon reception, the orchestrator submitter extracts the contained security policies along with the application information and VM configuration of worker nodes, included by the user. The orchestrator submitter parses the user template and submits security policy data and access credentials to the *Security Policy Manager* - a component that defines and implements security policies. Upon reception of the message, the Security Policy Manager validates the received security policies (i.e. that the received policies are subset of a predefined security policies set). If policy validation is successful, the Security Policy Manager uses the access credentials and other relevant information (e.g. the image identifier of the virtual container where the application will be hosted) to enforce the user-defined security policies. These include firewall settings (implemented by the *Firewall Service*), as well as policies regarding the use of other security enablers (crypto engine, credential manager, attestation service). The security architecture parallels the orchestrator architecture, which extracts functional information from the TOSCA templates to deploy the cloud infrastructure, set up the network communication topology and the intended applications.

6.2 Component Interaction

The Security Policy Manager invokes security enablers and interacts with orchestration components, as follows.

Prior to application deployment, the Security Policy Manager validates the virtual container. Thus the Security Policy Manager verifies the integrity of the virtual container image before the virtual container is launched, by invoking a function of the *Attestation Service* described in Section 5. Once the virtual container is validated, the Security Policy Manager authenticates the user to the cloud service provider and instantiates the virtual container from the image. This enables the VM orchestrator to uses user-supplied cloud credentials to authenticate to the available cloud service providers and subsequently generate or delete VMs on remote resources. Meanwhile, the virtual container manager generates, deletes or migrates virtual containers based on resource consumption of the running application. Prior to launching the application, the system administrator can request to attest the integrity of the software executing on the host, using the *attestation authority* described in Section 5. This provides the user with security guarantees about the trusted state of the entire host. The underlying protocol for the remote attestation is based on previous work [19, 20].

6.3 Discussion

In the proposed security architecture, we focus on the separation between the security and the core components in an orchestrator architecture. This provides more flexibility in extending security features in an existing orchestrator system. Apart from that, in orchestrator frameworks, multi-cloud deployment relies on the worker node manager component which supports various platforms. Furthermore, most of the provided security features are at machine and/or container level. Thus, the implementation during the development phase can be really simplified. The only exception is the attestation service, where the Security Policy Manager needs to deal with a multi-cloud deployment.

7. Conclusion

In this paper, we reviewed the security landscape of cloud orchestration and address the current knowledge gap in related literature. This paper, describes the building blocks of a cloud orchestrator for federated cloud deployments with support for security enablers. In this study, we analysed the

security of current orchestrators and described a set of relevant security risks. We defined a threat model, outlined the security assumptions and described the actual attack surface. Based on the security analysis, we proposed a set of security and privacy-preserving enablers for cloud orchestraors for federated cloud deployments. Finally, we proposed a security architecture that has the potential to enhance the security of cloud orchestration. In upcoming work, we intend to extend the MiCADO open-source orchestration framework by adding security enablers identified in this work.

Acknowledgments

The research was conducted within the COLA project and received funding from the European Union's Horizon 2020 research and innovation programme under grant No 731574.

References

[1] G. Adzic and R. Chatley. Serverless Computing: Economic and Architectural Impact. In *Proc. 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 884–889. ACM, 2017. ISBN 978-1-4503-5105-8.

[2] W. Antosz, S. Gringanze, P. Pardyak, D. Russell, R. Spencer, P. Sterley, A. Tiwary, and M. Vainer. Automation for virtualized IT environments, 2009. US Patent App. 12/396,353.

[3] C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu. From security to assurance in the cloud: A survey. *ACM Comput. Surv.*, 48:2:1–2:50, July 2015.

[4] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann. TOSCA: Portable Automated Deployment and Management of Cloud Applications. In *Advanced Web Services*, pages 527–549, New York, NY, 2014. Springer New York.

[5] F. Cabitza and C. Simone. Computational coordination mechanisms: A tale of a struggle for flexibility. *Computer Supported Cooperative Work (CSCW)*, 22(4):475–529.

[6] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10(2):63–81, April 2011.

[7] R. Dowsley, A. Michalas, M. Nagel, and N. Paladi. A survey on design and implementation of protected searchable data in the cloud. *Computer Science Review*, In press, August 2017.

[8] G. Greenwald. How the NSA tampers with US-made Internet routers. *The Guardian*, 10(5), May 2014.

[9] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. Occupy the Cloud: Distributed Computing for the 99%. In *Proc. of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 445–451. ACM, 2017. ISBN 978-1-4503-5028-0.

[10] T. Kiss, P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, and G. Terstyanszky. Micado - microservice-based cloud application-level dynamic orchestrator. *Future Generation Computer Systems*, 2017.

[11] J. Kovács and P. Kacsuk. Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. *Journal of Grid Computing*, November 2017.

[12] N. Kubala. Introducing container-diff, a tool for quickly comparing container images, November 2017. URL `https://opensource.googleblog.com/2017/11/container-diff-for-comparing-container-images.html`. Online; Accessed November 20, 2017.

[13] Lauwers, Chris. Declarative vs. Imperative Orchestrator Architectures. Online; Accessed December 2017, June 2017. URL `http://blog.ubicity.com/2017/06/declarative-vs-imperative-orchestrator.html`.

[14] 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. In *Proc. of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 461–472. ACM, March 2013.

[15] A. Michalas, N. Komninos, N. R. Prasad, and V. A. Oleshchuk. New client puzzle approach for dos resistance in ad hoc networks. In *Information Theory and Information Security (ICITIS), 2010 IEEE International Conference*, pages 568–573. IEEE, 2010.

[16] S. K. Nair, S. Porwal, T. Dimitrakos, A. J. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan, and A. U. Khan. Towards Secure Cloud Bursting, Brokerage and Aggregation. In *Proc. Eighth IEEE European Conference on Web Services*, 2010.

[17] N. Paladi. Towards Secure SDN Policy Management. In *Proc. 8th International Conference on Utility and Cloud Computing*, UCC '15, pages 607–611, December 2015. .

[18] N. Paladi and C. Gehrmann. TruSDN: Bootstrapping Trust in Cloud Network Infrastructure. In *Proc. 12th International Conference on Security and Privacy in Communication Networks*, SecureComm'16, pages 104–124. Springer, 2016.

[19] N. Paladi, A. Michalas, and C. Gehrmann. Domain Based Storage Protection with Secure Access Control for the Cloud. In *Proc. 2014 International Workshop on Security in Cloud Computing*, SCC '14, pages 35–42. ACM, June 2014.

[20] N. Paladi, C. Gehrmann, and A. Michalas. Providing User Security Guarantees in Public Infrastructure Clouds. *IEEE Transactions on Cloud Computing*, 5:405–419, July 2017. .

[21] P. S. Pawar, M. Rajarajan, T. Dimitrakos, and A. Zisman. Trust Assessment Using Cloud Broker. In *Trust Management VIII*, pages 237–244. Springer Berlin Heidelberg, 2014.

[22] D. Sgandurra and E. Lupu. Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems. *ACM Comput. Surv.*, 48:46:1–46:38, February 2016.

[23] R. Tahir, M. Huzaifa, A. Das, M. Ahmad, C. Gunter, F. Zaffar, M. Caesar, and N. Borisov. Mining on Someone Else's Dime: Mitigating Covert Mining Operations in Clouds and Enterprises. In *Proc. 20th International Symposium on Research in Attacks, Intrusions, and Defenses*, RAID 2017, pages 287–310, Cham, September 2017. Springer, Cham.

[24] D. Weerasiri, M. C. Barukh, B. Benatallah, Q. Z. Sheng, and R. Ranjan. A Taxonomy and Survey of Cloud Resource Orchestration Techniques. *ACM Comput. Surv.*, May 2017.