Security Analysis of Dynamic Infrastructure Clouds

Sören Bleikertz*[1], Thomas Groß†[2], and Sebastian Mödersheim‡[3]

[1]IBM Research - Zurich
[2]University of Newcastle upon Tyne
[3]DTU Informatics

1 Introduction

Multi-tenant infrastructure clouds can bear great complexity and can be exposed to security issues. Even if one only analyzes the static configuration of hosts, VMs, network and storage as well as their inter-connectivity, one faces a complex system. Configuration and topology changes make the system a moving target and can introduce violations of a security policy, for instance, manifest in incorrect deployment or isolation breaches. Although there have been analyses of isolation failures in complex static configurations of clouds, such as Bleikertz et al. [3], the analysis of dynamic configuration changes is largely missing [2].

Misconfigurations and Insider Attacks Even if committed unintentionally, misconfigurations are among the most prominent causes for security failures in infrastructure clouds. The ENISA report on cloud security risks [5] names isolation failure as major technical risk, with misconfiguration, lack of resource isolation and ill-defined role-based access policies as notable root vulnerabilities. If committed intentionally by a malicious insider, misconfigurations expose the infrastructure to greater risks. The CSA threat report [4] as well as the ENISA report agree to insider attacks as a TOP 10 cloud security risk as well as malicious insiders as a "very high impact" technical risk [5]. The analysis of all configuration changes is crucial here, as the insider could create a transient insecure state to attack the system and change it to a secure configuration before the next security analysis. Therefore, there is a need to analyze management operations for their security properties and to achieve overall accountability for administrator actions. The question is: How can we model configuration changes induced by management operations of cloud administrators and check these changes for violations of a security policy?

Analysis of Dynamic Infrastructure Clouds The concrete aim of our research is mitigate the security impact of misconfigurations: (a) We enable honest administrators to create a change plan for an infrastructure in advance and have this change plan checked for violations of the security policy in a what-if analysis. (b) We establish an authorization proxy to have all configuration changes independently checked for violations of the security policy, creating a run-time audit for misconfigurations and their security impact. (c) We direct this research towards the run-time mitigation of misconfigurations and enforcement of security policies. Whereas (b) only establishes an audit log of misconfigurations causing security problems, (c) is aiming at enforcing the security policy.

To achieve this analysis, we need multiple ingredients. First, we need a faithful representation of the topology and configuration of the virtualized infrastructure, which we call a realization model. Second, we need a description of information flow traversal of infrastructure components to evaluate information flow as the key security aspect. We draw upon the work of Bleikertz et al. [3] for both points. Third, we

*sbl@zurich.ibm.com
†thomas.gross@newcastle.ac.uk
‡samo@dtu.dk

need a language to specify security goals for virtualized infrastructures, where we resort to the language *VALID* [1], already used in research prototypes for that very purpose.

A crucial piece missing for a dynamic system analysis, however, is a formal model of topology and configuration changes in infrastructure clouds. The model needs to capture how relevant operations change the topology and its security properties. Such a model needs to capture basic operations, such as VMware's UpdatePortGroup as an operation that changes the VLAN configuration, or larger asynchronously executed tasks, such as creating or migrating a virtual machine. Expressing such cloud operations in a formal system is challenging: If the model is too abstract, it may fail to capture many significant attacks, while if it is too detailed, formal reasoning about security in the model becomes infeasible—both for manual and for automated verification. The main contribution of this paper is to develop an *operations model*, which enables us to establish an analysis system for misconfigurations.

Our approach builds upon graph rewriting, where topology, security goals and configuration changes are expressed as graphs and transformations of graphs. This methodology allows us to check efficiently whether configuration changes applied to a given topology will violate the security goals or not. We realize a prototype tool for this analysis as well as application cases in change planning and run-time audit of misconfigurations and establish this analysis in a practical environment with VMware.

2 System and Threat Model

2.1 A Graph Model of Virtualized Infrastructures

A graph-based model of static virtualized infrastructures has been proposed in [3]. The vertices of such a graph represents the virtualized infrastructure elements, e.g., physical servers or virtual machines, and the edges model the relationship among the elements, thereby capturing the topology of the system. Furthermore, nodes are typed, e.g., vMachine, and attributed to capture further properties and configuration aspects of each element. We consider the model as a directed, node and edge typed, and attributed graph.

In order to build the model we require sufficient information on the topology and configuration of a virtualized infrastructure. The authors of [3] describe two steps for their model construction: *Discovery* (§4.1) and *Translation* (§4.2). The discovery extracts the configuration from the hypervisors or management system of heterogeneous virtualized infrastructures, and translates the extracted configuration data into the model.

2.2 Threat Model

As core threat, we model non-malicious (deliberate and non-deliberate, accidental and incompetence) human-made faults. Even if administrators are honest, they can still make mistakes that lead to a security breach. Therefore, their behavior is not malicious and byzantine, but comes with some fairness constraints: A provider administrator will attempt to issue commands in a well-defined way through the service interface. A provider's behavior will take feedback from a security analysis or an audit into account. As part of the system foundations, we require that the analysis probes discovering the infrastructure configuration get an authentic view of the configuration.

In general, the analysis method proposed is capable to handle malicious, byzantine adversaries as well. To protect against those, the infrastructure needs to be modified to enforce sole access through the management interface (and prevent circumvention approaches, such as a direct SSH log-in to the physical hosts). Also, the security validation needs to be mandatory for all management operations, which is part of future work.

3 Modeling and Analysis of Dynamic Infrastructure Clouds

3.1 A Model of Dynamic Virtualized Infrastructures

The core of our model are graph transformations and we introduce a novel modeling of management operations and their impact on the configuration and topology of a virtualized infrastructure given as a graph. Furthermore, we integrate existing approaches [1, 3] for the formalization and analysis

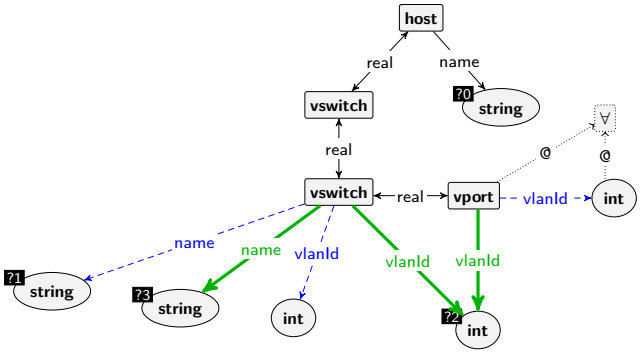

Figure 1: UpdatePortGroup

of security goals for virtualized infrastructure in our model, thereby establishing a unified approach. Additionally, these existing approaches are extended to enable the security analysis of dynamic virtualized infrastructures.

The basic idea of graph transformation is as follows. We have a graph transformation rule p, also called a *production*, in the form of $p : L \xrightarrow{r} R$, where graphs L and R are denoted the left hand side (LHS) and right hand side (RHS), respectively. A partial graph morphism r, the *production morphism*, establishes a partial correspondence between elements in the LHS and the RHS of a production, which determines the nodes and edges that have to be preserved, deleted, or created. A *match* m finds an occurrence of L in a given graph G, then $G \overset{p,m}{\Longrightarrow} H$ is an application of a production p, where H is a derived graph. H is obtained by replacing the occurrence of L in G with R.

Our unified model forms a *graph grammar* that consists of a *start graph* and a collection of *productions*, which transform the start graph. In our case, the start graph is a graph model of the virtualized infrastructure, and the productions represent our model of topological changes, information flow analysis, and policy specification.

3.1.1 Modeling of Infrastructure Changes

The *Operations Transition Model* captures the changes to the topology and configuration of a virtualized infrastructure through management operations. Our goal is a practical security system for virtualized infrastructures, therefore we focus our modeling efforts, as an example, on VMware and its management operations. Each management operation is modeled as a graph production that transforms the virtualized infrastructures, which is modeled as a graph, into a modified one.

For any existing real-world virtualized infrastructures like VMware, the API documentation does not provide a precise formal description and model, but rather a semi-formal description of the operations, parameters, as well as the preconditions and effects that the operations have. A contribution of this paper is to obtain a formal model that allows for precise statements to be made and proved or refuted. It is of course not possible to formally prove the correctness of such a model itself, however there is a good methodology to obtain a "good" model by combining two directions.

The first direction is to follow the documentation and translate the documented effects into our (abstract) graph model. The second is to experiment with the real implementation, to verify that the operations indeed do have the effect on the infrastructure that our model predicts. To study these experiments we need to translate the real infrastructure topology into our abstract graph before and after the operation has been performed, and check that the resulting graph transformation coincides with our model of the operation.

An example of a operations model for the VMware operation UpdatePortGroup is given in Figure 1. Using this operation, an administrator can change the configuration on an existing portgroup. The portgroup is identified by its name, as well as the host where it resides on, and the operation allows to change the portgroup's name and VLAN ID. Changing attributes is modeled as changing the edges to different data nodes based on the input parameters. In order to maintain compatibility with the existing graph model, not only does the portgroup node contain the VLAN ID, but also the associated vport nodes, i.e., virtual switch ports. Therefore, changing the VLAN ID of the portgroup also requires to change the

VLAN ID of all virtual ports associated to that portgroup. For this we use a universal quantifier ∀ that updates the vlanId attributed of all matching vport nodes [9].

3.1.2 Dynamic Information Flow Analysis & Security Policies

An approach that has been presented in [3] performs an information flow analysis on a graph-based model of a virtualized infrastructure, in order to detect isolation failures. The approach consists of a set of *traversal rules* that captures how elements in the infrastructure provide isolation, e.g., VLANs provide network isolation. A graph traversal guided by the set of traversal rules computes the transitive closure and determines the information flow in the system.

This information flow analysis is so far limited to a static snapshot of the network and thus unable to deal with the dynamic nature and frequent changes of such infrastructures. We need to integrate this information flow analysis into our dynamic graph model and thus obtain a dynamic information flow analysis. We do so by expressing the traversal rules of the information flow analysis as graph production rules. We use the graph rewriting control language of GROOVE to compose the information flow analysis from the rules.

The final part of our unified model deals with the formalization of security policies, which describe properties of the topology and configuration of an infrastructure cloud. We follow here the approach of the policy language *VALID* [1]. Such policies can also be expressed as graph production rules, where a rule matches parts of the graph with potential additional conditions on this match. Typically, one would express a *violation* of a security policy as a graph production rule, and try to match the rule on the evolving graph. Once a match is found, a violation of the security policy is found.

3.2 Automated Analysis using *GROOVE*

Many graph transformation tools have been developed in the past, among them: *GROOVE* [7, 8], AGG [11], GRGen [6], and PROGRES [10]. For this work, we decided to use *GROOVE* as our graph transformation environment. *GROOVE* is a general-purpose graph transformation tool that enables an expressive specification of production rules, e.g., by providing nested quantifications and path constructions using regular expressions on edge labels. Furthermore, an imperative control language allows to schedule the application of rules, which also allows more complex control flow constructions, as well as enabling parametrized rules, where parameters are passed from a control program to a production rule. We refer to a detailed comparison between different graph transformation tools to [7].

3.3 Application Scenarios of our Analysis System

3.3.1 Change Planning

The administrator specifies the sequence of change requests (either directly as a change program in *GROOVE*'s control language or in a proprietary provisioning language, translated to it). Once the change program is submitted to *GROOVE*, the tool will apply the changes to the realization model, derived from the actual infrastructure. By that, the tool can establish a what-if analysis and determine what security impact the intended changes will have on the infrastructure.

If the new realization model obtained from the execution of the change program violates the *VALID* security goals, the tool notifies the administrator to reject the proposed change requests and provides the *GROOVE* output of the matched attack sub-graph as diagnosis. Otherwise, the tool returns that the intended changes are compliant with the security goals, after which the administrator can provision the changes to the infrastructure.

3.3.2 Runtime Audit of Misconfigurations

Run-time audit of misconfigurations expands on the principles of the change planning. Whereas change planning requires the administrator to devise the changes in advance and have them checked by our analysis statically, the run-time audit intercepts change requests dynamically at an authorization proxy and checks them concurrently as they occur. The idea of the run-time auditing is to establish accountability for administrator actions: administrator's configuration changes are validated against the security policy

and the results of these checks entered into the audit logs along with the administrator's username and the committed commands.

We introduce an authorization proxy as wrapper of the administration API, which acts as policy enforcement point (PEP) and auditor on configuration changes, and employ our analysis as part of the policy decision mechanism.

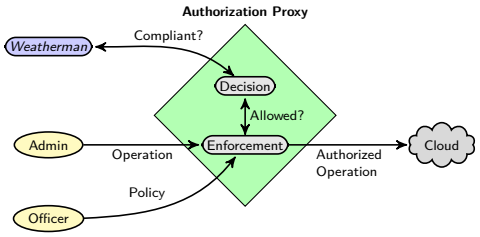

Figure 2: Architecture for Run-time Audit.

Acknowledgments

This research has been partially supported by the TClouds project[1] funded by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement number ICT-257243.

References

[1] Bleikertz, S., and Gross, T. A Virtualization Assurance Language for Isolation and Deployment. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY'11)* (Jun 2011), IEEE.

[2] Bleikertz, S., Gross, T., and Mödersheim, S. Automated Verification of Virtualized Infrastructures. In *ACM Cloud Computing Security Workshop (CCSW'11)* (Oct 2011), ACM.

[3] Bleikertz, S., Gross, T., Schunter, M., and Eriksson, K. Automated Information Flow Analysis of Virtualized Infrastructures. In *16th European Symposium on Research in Computer Security (ESORICS'11)* (Sep 2011), Springer.

[4] CSA. Top threats to cloud computing v1.0. Tech. rep., Cloud Security Alliance (CSA), mar 2010.

[5] ENISA. Cloud computing: Benefits, risks and recommendations for information security. Tech. rep., European Network and Information Security Agency (ENISA), nov 2009.

[6] Geiss, R., Batz, G. V., Grund, D., Hack, S., and Szalkowski, A. GrGen: A Fast SPO-Based Graph Rewriting Tool. In *Third International Conference on Graph Transformation (ICGT 2006)* (2006), A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, Eds., vol. 4178 of *Lecture Notes in Computer Science*, Springer, pp. 383–397.

[7] Ghamarian, A. H., de, M. M., Rensink, A., Zambon, E., and Zimakova, M. Modelling and analysis using GROOVE. *International Journal on Software Tools for Technology Transfer (STTT)* (March 2011).

[8] Rensink, A. GROOVE: GRaphs for Object-Oriented VErification. `http://groove.cs.utwente.nl/`.

[9] Rensink, A., and Kuperus, J.-H. Repotting the geraniums: on nested graph transformation rules. In *Graph transformation and visual modelling techniques, York, U.K.* (2009), A. Boronat and R. Heckel, Eds., vol. 18 of *Electronic Communications of the EASST*, EASST.

[10] Schürr, A., Winter, A. J., and Zündorf, A. The PROGRES approach: Language and environment. In *Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools*, H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[11] Taentzer, G. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In *Applications of Graph Transformations with Industrial Relevance (AGTIVE 2003)* (2003), J. L. Pfaltz, M. Nagl, and B. Böhlen, Eds., vol. 3062 of *Lecture Notes in Computer Science*, Springer, pp. 446–453.

[1] `http://www.tclouds-project.eu`