Proactive Security Analysis of Changes in Virtualized Infrastructures

Sören Bleikertz
Carsten Vogel*
IBM Research - Zurich
{sbl,ten}@zurich.ibm.com

Thomas Groß
Newcastle University
thomas.gross@newcastle.ac.uk

Sebastian Mödersheim
DTU Compute
samo@dtu.dk

ABSTRACT

The pervasiveness of cloud computing can be attributed to its scale and elasticity. However, the operational complexity of the underlying cloud infrastructure is high, due to its dynamics, multi-tenancy, and size. Misconfigurations and insider attacks carry significant operational and security risks, such as breaches in tenant isolation put both the infrastructure provider and the consumers at risk.

We tackle this challenge by establishing a practical security system, called *Weatherman*, that proactively analyzes changes induced by management operations with respect to security policies. We achieve this by contributing the first formal model of cloud management operations that captures their impact on the infrastructure in the form of graph transformations. Our approach combines such a model of operations with an information flow analysis suited for isolation as well as a policy verifier for a variety of security and operational policies. Our system provides a run-time enforcement of infrastructure security policies, as well as a what-if analysis for change planning.

1. INTRODUCTION

Multi-tenant virtualized infrastructures offer self-service access to a shared physical infrastructure with compute, network, and storage resources. While administrators of the provider govern the infrastructure as a whole and the tenant administrators operate in partitioned logical resource pools, both groups change the configuration and topology of the infrastructure. For example, they create new machines, modify or delete existing ones, causing large numbers of virtual machines to appear and disappear, which leads to the phenomenon of server sprawl. Therefore, self-service administration, dynamic provisioning and elastic scaling lead to a great number of configuration and topology changes, which results in a complex and highly dynamic system.

Misconfigurations and insider attacks are the adverse results of such complex and dynamic systems. Indeed, even if committed unintentionally, misconfigurations are among the most prominent causes for security failures in IT infrastructure [20]. Notably, according to studies by ENISA [7] and CSA [6], operational complexity, which leads to misconfiguration and security failures, as well as isolation failures are among the top threats in virtualized infrastructures. Isolation failures put both the provider as well as the consumers at great risk due to potential loss of reputation and the breach of confidential data. Further, malicious insiders and their attacks are considered a top, very high impact security risk. Consider an example of isolation breach from misconfiguration, which we encountered in the security analysis of a financial institution's in-house VMware-based production cloud: An administrator performed a wrong VLAN ID configuration change leading to an unnoticed network isolation breach between the high-security and the test security zone.

Core Idea: In combating such security failures, the assessment of configuration changes and rigorous enforcement of security policies is a crucial requirement. It is important to establish whether an intended configuration change will compromise the security of the system before the change is deployed. We build a practical analysis system, called *Weatherman*, that uses a model-based approach for assessing configuration changes and their impact on the security compliance of a virtualized infrastructure. We call our system *proactive* as changes are analyzed before they are deployed.

Facing an intended configuration change, *Weatherman* needs to establish how the infrastructure would be affected. Our *operations transition model* (§3.1) covers security-relevant operations and models their impact on the infrastructure configuration and topology in a graph rewriting language. For our example, it contains a model of the VMware operation UpdatePortGroup encoding how VLAN ID changes affect the network. Having established a what-if infrastructure model for the intended change, the next important question is: How does the information flow and isolation change in the system? *Weatherman* performs an information flow analysis in the what-if infrastructure model as an intermediary step to determine isolation properties (§3.2). Finally, the infrastructure model is checked against a variety of security and operational policies, which are implemented as graph matches and evaluated by the graph transformation engine (§3.3). Overall, our system establishes whether a future configuration change will constitute a security compromise and rejects the change if a violation is detected (§4).

Our contributions are the following: **1)** We propose the first formal model of cloud management operations, the *op-*

*Work done at IBM, now affiliated with Hylastix, cv@hylastix.com.

ACSAC '15, December 07 - 11, 2015, Los Angeles, CA, USA

ISBN 978-1-4503-3682-6/15/12...$15.00

DOI: http://dx.doi.org/10.1145/2818000.2818034

erations transition model, that captures how such operations change the infrastructure's topology and configuration. We express the operations as transformations of a graph model of the infrastructure, which is based upon the formalism of graph transformation [23]. **2)** We propose a unified model that integrates with the operations model the specification of security policies as well as an information flow analysis suited for isolation policies. We formalize a variety of policies, such as in the areas of isolation, dependability, and operational correctness using graph matching. **3)** Based on our model, we design and implement a practical security system, called *Weatherman*, which assesses and proactively mitigates misconfigurations and security failures in VMware infrastructures. We analyze and discuss the security of our system for a practical deployment environment.

2. SYSTEM AND SECURITY MODEL

In Fig. 1 we illustrate our model of a virtualized infrastructure, which consists of (virtualized) computing, networking and storage resources that are configured through a well-defined management interface. We consider multiple administrators with different privileges, where the *provider* administrators govern the entire virtualized infrastructure, and *tenant* administrators manage an assigned logical resource pool. The model is poised towards a proactive analysis based on operations that are intercepted at the management host and the analysis system operates on a model of the virtualized infrastructure.

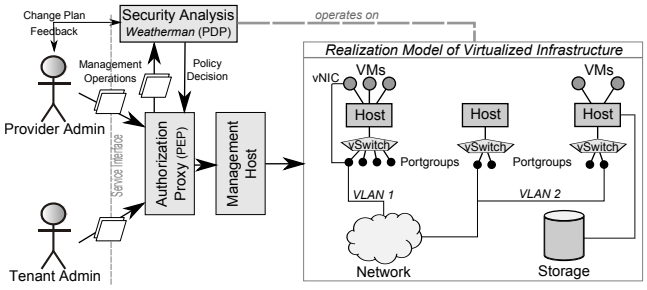


Figure 1: The System Model consists of a topology model of the virtualized infrastructure, an authorization proxy as Policy Enforcement Point, and a run-time security analysis of operations.

We represent the virtualized infrastructure in a graph model, called *Realization* model [4], which is an undirected, vertex typed and attributed graph. The vertices of the graph represent the components of the virtualized infrastructure, which may be entire sub-systems, such as physical servers or virtual machines, or low-level components, such as virtual network interfaces. Vertices are typed, e.g., type vm denotes a virtual machine, and annotated with name/value attributes. The attributes encode detailed properties of the components and capture their configuration. The edges of the graph represent the connections and relationships among components of the virtualized infrastructure, encoding its topology. Fig. 1 illustrates the Realization model, which spans compute, network, and storage resources. In particular, we illustrate the networking part in more detail. Physical hosts and their hypervisors provide networking to VMs by virtual switches that connect the VMs to the network. A virtual switch contains virtual ports, to which the VMs are connected via a virtual network interface card (vNIC). Virtual ports are aggregated into *port groups*, which apply a common configuration to a group of virtual ports. Virtual LANs (VLANs) allow a logical separation of network traffic between VMs by assigning distinct VLAN IDs to the port groups. Our network model is focused on the OSI Layer2.

The Realization model is populated through an automated extraction of the configuration of the virtualized infrastructure from the central management host and the translation of the configuration into graph nodes and vertices. For each element in the configuration, such as a virtual machine, it constructs a corresponding model vertex and populates the required attributes. To ensure a complete translation of all relevant elements in the configuration, an element is either translated or explicitly ignored. A translation warning is thrown for unhandled elements. Since we are dealing with a dynamic infrastructure, we also need to keep the graph model of the infrastructure in sync. For this, we continuously monitor the virtualized infrastructure for changes and translate the observed changes into updates of the graph model using an approach presented in [5].

Threat Model: We establish a threat model based on the dependability taxonomy [1]. Agents, users and administrators can be malicious or non-malicious. Thereby, we cover all classes of human-made faults. Faults can be introduced deliberately as result of a harmful decision or without awareness; faults can be introduced accidentally or by incompetence. These fault classes include misconfigurations as well as malicious attacks and resulting security failures and, thereby, constitute a strong adversary model. Agents that operate on behalf of a human are canonically covered by this threat model, because the threat model is independent from the issuer of an operation. Combined with the system model covering compute, network and storage, this threat model allows for a comprehensive security analysis of virtualized infrastructures. As constraints, the adversary is bound to the well-defined management host API and cannot subvert the communication between the management hosts and the analysis system. In §5.1 we assess secure deployment approaches to realize such a constraint in practice. The software security of the management host and the hypervisors is out of scope.

3. A MODEL OF DYNAMIC VIRTUALIZED INFRASTRUCTURES

We capture multiple aspects relevant for the analysis and integrate them into a unified model based on graphs and graph transformations. We represent the topology and configuration of the virtualized infrastructure, establish how the infrastructure can be changed by management operations, and verify the infrastructure with regard to security policies. As we are focusing on isolation properties, we further need to determine information flows in the system.

3.1 Modeling of Infrastructure Changes

We model the impact of management operations in terms of infrastructure changes using graph transformations. We will briefly introduce the formalism and describe our methodology how we can create a model for a practical system, followed by concrete examples of models for specific VMware operations.

3.1.1 Modeling Operations as Graph Transformations

We model each operation as a graph transformation rule, which takes the graph representation of the virtualized in-

Table 1: Overview of Security-Critical VMware Operations [27].

Operation	Description	Policy Impact
AddPortGroup	Creates a new port group on a given host and virtual switch, with a name and VLAN ID.	Network Isolation
UpdatePortGroup	Updates the name and/or VLAN ID of an existing port group on a given host.	Network Isolation
RemovePortGroup	Removes an existing port group on a host given by name.	Dependability
UpdateNetworkConfig	Updates the network configuration of a host; another means of creating or updating port groups.	Network Isolation
CreateVM	Creates a VM on a host with virtual storage and network resources (modeled as sub-operations).	Compute Placement
AddVirtualDisk	Creates a virtual disk for a VM with file backend.	Storage Isolation
AddVirtualNic	Creates a virtual NIC connected to a port group.	Network Isolation
ReconfigVM	Updates a VM's configuration, including storage and network resources.	
UpdateVirtualDisk	Updates the file backend of a virtual disk.	Storage Isolation
UpdateVirtualNic	Connect a virtual NIC to a new port group.	Network Isolation

frastructure as input and transforms it into a modified one. According to [23], we define a graph transformation rule as the following.

Definition 1 (Graph Transformation Rule)
A graph transformation rule p, also called a production *rule, has the form* $p : L \xrightarrow{r} R$*, where graphs L and R are denoted the left hand side (LHS) and right hand side (RHS), respectively. The* production morphism *r 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.*

An important extension to graph transformations are application conditions that express constraints on the applicability of a production rule, which includes constraints on the attribute values of vertices. Further, parameterized rules capture expected attribute values as parameters that need to be satisfied by the application condition. This is important for our model as management operations are parameterized.

The *Operations Transition Model* consists of graph transformation rules and captures how operations change the topology and configuration of a virtualized infrastructure.

Definition 2 (Operations Transition Model)
The Operations Transition Model consists of named and attribute-parameterized graph production rules which is written as the set $P = \{p_1, \ldots, p_n\}$ *. Each rule corresponds to a parameterized management operation op and models the effects of op on the infrastructure as graph modifications on the* Realization *graph model. The name of each production rule corresponds to the name of the management operation.*

The ordering of the rules is not relevant for the modeling as the rules model the operations independently. However, the ordering becomes important for the analysis (cf. §4) which performs an ordered application of a subset of rules with parameter values on a given infrastructure model graph.

3.1.2 Modeling Methodology

For any existing real-world virtualized infrastructure like VMware, the API documentation does not offer a precise formal definition and model, but rather a semi-formal description of the operations. A contribution of this paper is to create a formal model that allows for precise statements to be made and proved or refuted. It is of course not possible to formally prove that our formal model captures the informal description, however there is a methodology to obtain a "good" model by combining the following directions:

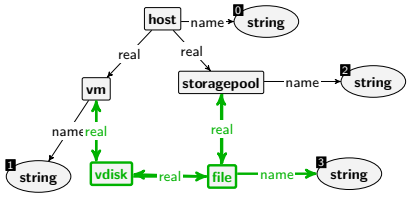
1) API Documentation: We follow the API documentation that describes for each operation the functionality, the required parameters as well as the preconditions and effects that the operation has on the infrastructure. For the relevant operations, we determine the parameters that are security-critical and which will have an impact on the model when the operation is performed. Overall, the API documentation provides us with a list of relevant operations, their parameters, and a high-level idea of their impact on the infrastructure.

2) Infrastructure Change Assessment: In order to understand how the infrastructure is changed in detail by an operation, we inspect the configuration of the infrastructure *before* and *after* the operation has been issued. For each operation that we have selected based on the API documentation, we vary the parameter values to determine their different effects, if applicable. For example, varying the VLAN identifier parameter of a virtual network re-configuration preserves the same effect, whereas varying the device configuration of a new virtual machine creation may lead to different topology changes, e.g., attaching the VM to a different virtual network. We do not only study the differences in the configuration after each operation, we also investigate the differences in the resulting graph models. The changes from the graph model of the configuration before the operation was performed and the graph model after the operation guides us how a graph production rule of the operation may look like. The graph model changes include new and deleted vertices and edges, as well as attribute changes.

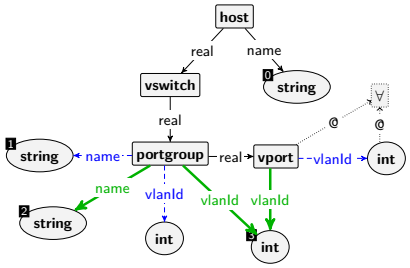
3) Validation with Administrative Tasks: Finally, we also performed common administrative actions from the graphical management client, which itself issues the documented API operations. We intercepted and analyzed these issued operations and discovered that the management client makes use of other operations from the API to perform the same task. For example, to change the VLAN identifiers of a virtual network component the usual operation is UpdatePortGroup, however the client software issues the much more general operation UpdateNetworkConfig. We extended our model to include these other variations of performing security-critical tasks.

3.1.3 Modeling of a Practical System

The VMware API (v5.0) consists of 545 methods [27], but many of these operations do not affect the topology or configuration of the virtualized infrastructure, because they deal with VMware-specific management and operations aspects such as licensing and patch management, handling of administrative sessions, or diagnostics and alarms. We identified 95 operations that modify the topology or configuration of the infrastructure. We model a security-critical subset of VMware management operations as listed in Table 1, which also indicates potential policy violations (cf. §3.3). We consider



(a) AddVirtualDisk



(b) UpdatePortGroup

Figure 2: Storage and network operations modeled as graph transformation rules in *GROOVE*.

changes to the virtual compute, network, and storage infrastructure, such as the creation of virtual machines, creation or updates of virtual switches and interfaces, and attachment of storage to virtual machines. Complex operations, such as creating VMs, are broken down into sub-operations.

From the subset of operations, we present the production rules of two operations: The UpdatePortGroup operation changes the isolation property of a virtual network, as well as the sub-operation AddVirtualDisk of the CreateVM operation that connects a new virtual disk to a created VM. The two examples cover a spectrum of operation classes: First, operations that create infrastructure elements as well as updating existing ones; Second, operations that work on different resource types, namely, storage and network.

The production rules are illustrated in Fig. 2. We selected *GROOVE* [9], a tool for specifying and applying transformation rules, as our graph transformation environment and the rules are shown in its visual notation. Each rule is represented by a graph that describes both the LHS and RHS (cf. Def. 1) with the following semantics: **Readers** (thin line) are nodes and edges that need to be matched in the graph for the rule to be applicable and which are preserved in the transformation, i.e., they belong to both the LHS and RHS. **Creators** (bold line) for newly added nodes and edges, which only belong to the RHS. **Erasers** (thin dashed) are nodes and edges that need to be matched, and which will be deleted by the transformation, i.e., only belong to the LHS. **Embargoes** (thick dashed) are nodes and edges that need to be *absent* in the graph, in order that the rule matches.

Disk Creation Operation: AddVirtualDisk.

```
AddVirtualDisk(string hostname, string vmName, ↩
    string storagepool, string filename)
```

As part of the creation of a virtual machine, a virtual disk is created and attached to the VM, which is identified by a given hostname and the VM name. Virtual disks are file-based (given by a filename), and the file is residing on a storage pool, given by a name. The production rule of Fig. 2a finds the corresponding subgraph where the names of host, VM, and storage pool match the rule's parameters. New nodes for the virtual disk (vdisk) as well as the file backend (file) are created and connected to the matched subgraph by specifying them as creator elements (visually thick line). In *GROOVE*, attributes of a node are represented by data nodes, visually indicated as ecliptic shapes, that are connected by a labeled edge, where the label denotes the attribute name. The numeric superscript on data nodes show that an attribute value is matched against a rule parameter, e.g., the host's name is matched against parameter 0.

Virtual Network Update Operation: UpdatePortGroup.

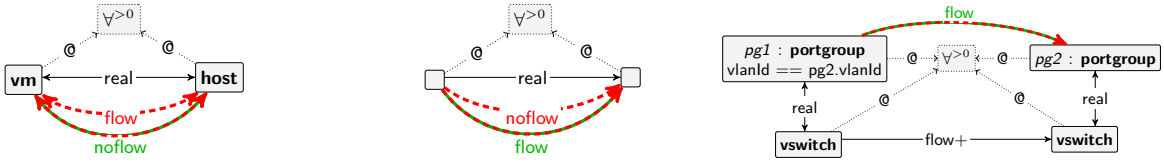
```
UpdatePortGroup(string hostname, string pgName, ↩
    string newPGName, int newPGVlanId)
```

Using this operation, an administrator can change the configuration of an existing port group. The port group is identified by its name, as well as the host where it resides on, and the operation allows to change the port group's name and VLAN ID. Changing attributes is modeled as changing the edges to different data nodes based on the input parameters. The VLAN ID is not only contained in the port group nodes, but also in the associated vport nodes, i.e., virtual switch ports. Therefore, changing the VLAN ID of the port group also requires to change all virtual ports associated to that port group. For this we use the universal quantifier ∀ that applies a sub-rule, given by nodes connected to the quantifier with @ labeled edges, to all its matches [21]. In this case, it updates the vlanId attributes of all matching vport nodes.

3.2 Dynamic Information Flow Analysis

Our information flow analysis computes potential information flows within the infrastructure, and thus enables the system to determine isolation failures between tenants. A set of graph production rules capture trust assumptions on the isolation of particular infrastructure elements, and construct the information flow graph by introducing edges that denote if flow is either permitted or denied.

We are drawing from existing work that computes information flow in virtualized infrastructures by using a graph coloring and traversal approach based on a set of traversal rules [4]. The traversal rules define for a pair of connected Realization model vertex types if the traversal and coloring should proceed or not. The rules further consider the traversal direction, vertex attributes, and the current graph color. We adapted the existing traversal rules, which capture best-practices on virtualization and network security, and formalized them as graph production rules. The challenge of such a formalization is that a direct encoding of the graph coloring approach in *GROOVE* would result in an expensive blow-up of the state space. Therefore we opted for the construction of an information flow graph instead of performing a graph coloring. Another challenge is the formalization of color-dependent traversal rules, which have been used to model forms of network tunneling such as virtual networks with VLANs. We model the tunneling with "fast edges" that connect the logical endpoints of the tunnel directly with an information flow edge. The fast edge rules capture the termination properties of the original traversal rules, e.g., the endpoints must have the same VLAN identifier, as well as the connectivity condition of the endpoints, i.e., they are mutually reachable. Overall, our formalization of the information flow analysis as graph transformations differentiates from the existing work [4]. In particular, we construct an information flow graph as an overlay that enables us to dynamically adjust it upon changes in the system model. Further, as we are addressing the previous challenges, our approach is better suited to be expressed as graph transformations.



(a) Simple Stop Rule: Trusted hypervisor prevents flow from VMs to Host and vice versa.

(b) Default Rule: Flow between any pair of connected vertices not yet covered by a noflow edge.

(c) Fast-Edge Rule: Flow between Portgroups with the same VLAN ID and connected VSwitches.

Figure 3: Examples of different kinds Information Flow Rules as modeled in *GROOVE* as Production Rules.

3.2.1 Information Flow Model

We now define our information flow model and present the set of rules that construct such the information flow graph.

Definition 3 (Information Flow Model)
*We model information flow in a virtualized infrastructure, given by a Realization model graph $G_R = (V_R, E_R)$, as a directed and edge-typed graph $G_I = (V_I, E_I)$, where $V_I = V_R$. An edge type function $t_e : E_I \to \{\text{flow}, \text{noflow}\}$ denotes for each information flow edge $e = (u, v)$ if information flow from u to v is possible (*flow*) or not (*noflow*).*

We denote the information flow graph as an overlay on the realization model graph, because they share the same vertex set. We consider a unified graph $G = (V_R, E_R \cup E_I)$ where the Realization model graph edges are typed real.

The information flow analysis takes a Realization model graph, a set of traversal rules in the form of graph production rules (as shown in Fig. 3), and applies them on the Realization model, thereby constructing the information flow graph overlay. When the Realization model changes, e.g., due to the operations transition model, we also adjust the information flow graph.

3.2.2 Information Flow Rules and Application

We differentiate between three kind of information flow rules: A *simple* rule describes information flow between a pair of adjacent vertices given by their types with potential conditions on the vertices' attributes. A *default* rule is a simple rule that matches any pair of adjacent vertices without any conditions. Finally, a *fast-edge* or *complex* rule describes information flow between non-adjacent vertices.

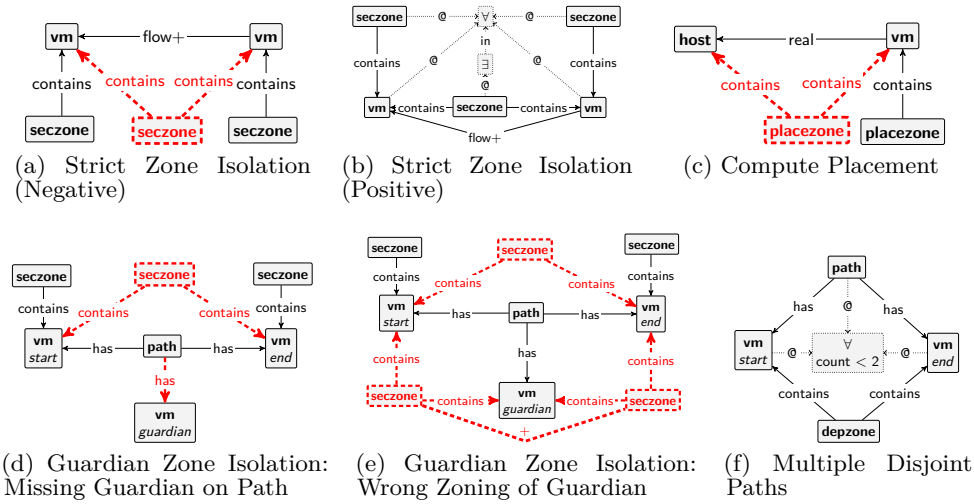
Simple Information Flow Rules: The first kind of rules are used both when the information flow is computed for the first time on the initial graph, or when new edges are added. They are simple in the sense that they work on directly adjacent nodes connected by a Realization model edge (real), and either introduce a directed information flow edge for flow or noflow. Fig. 3a shows a simple information flow rule that stops information flow between a host and a virtual machine (vm) by creating bidirectional noflow edges between them, if not already present. This captures the (arguable) trust assumption that no side-channel information leakage exists between virtual machines on the same host [22]. The noflow edge is created with a *conditional new*, i.e., it is only created if not already present by combining a creator and embargo edge. Applying the simple rules will eventually terminate when all pairs are connected by either a flow or noflow edge. We design the rules to be *confluent*, i.e., whenever more than one explicit rule is applicable, it does not matter for the result which one we take first. We can thus use the universal quantifier $\forall^{>0}$, which requires at least one match for the rule to be applicable, to express that we apply the production rule to all possible matches greedily (i.e., we do not have a state exploration).

Default Rule: The above simple rules typically represent trust assumptions on *isolation* properties of elements in the infrastructure and therefore introduce noflow edges. The flow edges are conditionally introduced by a *default* rule, as shown in Fig. 3b, if neither a flow nor noflow edge are present between a pair of nodes. The rule is applied when no more simple rules are applicable. Thus, the default means that we assume information may flow when the simple rules do not tell us otherwise. This may be too pessimistic, but with this over-approximation we are generally on the safe side. We achieve the operational aspect by designing simple *GROOVE* rule application strategies, in this case to first apply simple rules as long as possible and then apply the default rule as long as possible, i.e., until all node pairs have been evaluated.

Fast-Edge Information Flow Rules: A direct encoding of the original graph coloring of [4] is not suitable in *GROOVE* as the change in the graph state leads to an expensive blow-up of the state space. A feasible alternative is the introduction of fast-edges representing the pairs that need to have the same coloring (i.e., allowing a flow). As an example, Fig. 3c shows a production rule that creates a fast-edge between two VLAN endpoints that are not necessarily directly connected by a real edge, but which are connected through a path of flow edges. Here, two VMware port groups, which are modeled as portgroup with a VLAN identifier, are hosted on different virtual switches, and the rule fires for pairs of portgroups with the same VLAN identifiers, if the underlying switches are connected. A similar rule exists when two port groups are connected to the same vswitch.

Adjust Existing Information Flows: The dynamic information flow analysis needs to adjust the existing information flows if the Realization model graph changes. The removal of information flow edges that are connected to removed nodes is covered by the underlying formalism (*Single Push-Out* [23]) as dangling edges are removed. For each pair of nodes that are no longer connected by a real edge, but still feature an information flow edge, we need to remove the flow edge. This is accomplished by two production rules similar to the simple information flow rules, but with two untyped nodes, a condition that no real edge is present, and the removal of either a flow or noflow edge.

The information flow edges that are based on changed attributes are recomputed if their predicates do not hold anymore. That means, for each information flow rule that introduces an information flow edge based on an attribute condition, such as the VLAN ID attribute dependent rule of Fig. 3c, we have an adjusting production rule that verifies that the attribute condition still holds; if not, it revokes the



(a) Strict Zone Isolation (Negative)

(b) Strict Zone Isolation (Positive)

(c) Compute Placement

(d) Guardian Zone Isolation: Missing Guardian on Path

(e) Guardian Zone Isolation: Wrong Zoning of Guardian

(f) Multiple Disjoint Paths

Figure 4: Security and Operational Policies modeled in *GROOVE* as Graph Matches.

information flow edge. Adjusting the information flow graph based on changes in the Realization model may further influence connectivity-dependent information flow edges, such as the ones produced by the fast-edge portgroup rule. Similar to an adjusting production rule for attribute changes, we also have a production rule that deletes flow edges if their connectivity condition is no longer satisfied.

3.3 Infrastructure Policies as Graph Matches

The final piece of our analysis effort is the specification of security and operational policies. We formalize a wide variety of practical policies, such as isolation of security zones and prevention of single point of failures, as graph matches. Instead of production rules that transform the model, the policy rules only try to match a given graph pattern.

We usually express the security policies as *attack* states, i.e., a state of the topology or configuration that violates the desired security properties. Instead of verifying that a security property holds for the entire infrastructure, we try to find violations. However the formalism and analysis allow for both the specification of positive and negative policies. The analysis stops, i.e., finds a violation, if a propositional formula of the form $AttackPolicy_1 \vee \neg PositivePolicy_1 \vee AttackPolicy_2 \ldots$ is satisfied. That is, an attack state has been found when an attack policy has matched, or a positive policy no longer matches. Attack state policies have an advantage in the root-cause analysis of policy violations, since the analysis returns the matching part of the infrastructure that causes the violation, i.e., the attack state. Whereas for positive policies, the analysis does not provide a reason why a policy rule no longer matches. In the following we present a subset of policies that stem from security requirements of practitioners of infrastructure cloud deployments.

Strict Security Zone Isolation: We represent tenants as *security zones* which group together infrastructure elements, such as virtual machines, into zones. Each security zone is represented as a single vertex of type seczone with directed contains edges, that represent zone membership, to Realization model vertices. The zoning of elements is a policy setup performed by a security operator.

In this policy we require a *strict* isolation, i.e., no information flows, between any pair of zoned infrastructure elements that are not members of at least one common security zone. With the example of VMs as security zone members, we show both a positive and negative specification of this policy. Although we use VMs as an example, any infrastructure element can be grouped into security zones. Fig. 4a shows a negative/attack specification of the policy: We have a policy violation for a pair of zoned VMs that are connected by an information flow path (flow+) if they are *not* members of the same security zone. The statement flow+ is a regular expression on edges and requires at least one flow edge. On the other hand, a positive specification of this policy (Fig. 4b) states that for all zoned VM pairs, which can communicate, there must exists at least one zone that contains both VMs.

We allow elements to be part of multiple security zones, and our policy expects at least one common zone for element pairs with information flow. A problem arises when a multi-zoned element facilitates information flow between single-zoned elements. For example, if a VM acts as a firewall and is part of two security zones, then VMs of one zone may communicate with VMs of the other zone via the firewall VM, which is a violation of strict isolation. We handle inter-zone trusted elements with the *guardian* isolation policy (§3.3.1).

Compute Placement mandates the assignment of computing resources, that is on which physical hosts virtual machines must run. The motivation stems from both performance and availability reasons as well as security and legal requirements. Imagine that VMs must run on hosts from a particular geo-location due to privacy laws and data security requirements. By grouping physical hosts and VMs together into *placement zones*, similar to the previous security zones, this policy is violated if a VM is hosted on a physical server of another placement zone or no zone at all (cf. Fig. 4c). A zoned host can run VMs that are not part of any zone.

The related security concern of side-channel attacks due to VM co-location on the same physical host [22] is covered by the security zone isolation policy. The trust assumption if a particular hypervisor provides strong VM isolation or not is captured in the user-configurable information flow rules (cf. §3.2). From practical security policies we learned that co-locating different tenants is allowed only for a particular set of hypervisor products that are considered trusted.

3.3.1 Policies with Information Flow Path Conditions

We are also dealing with policies that have requirements on the information flow paths. For example, the guardian isolation policy requires that a trusted component, e.g., a firewall, is part of a flow path between elements of different zones. To express such policies, we can no longer rely on the flow+ path construct, because we cannot inspect the found paths. We model an explicit path finding with traversal rules that add vertices to a path vertex with directed edges to denote path membership. The state exploration applies the traversal rules, which perform a graph traversal on flow edges, and constructs all possible paths between pairs of *start* and *end* nodes. We can now express policies that verify conditions on the found paths.

Guardian Security Zone Isolation: Given a pair of elements that are not members of a common security zone and that are connected by an information flow path. It is mandatory that the communication is mandated by a trusted guardian, i.e., a vertex flagged as *guardian* must be part of the information flow path between the pair (cf. Fig. 4d). Additionally, the guardian must share a security zone with each element of the pair (cf. Fig. 4e). The first policy is violated if there exists a path between a pair of VMs, which do not belong to a same security zone, and the path does not contain a VM flagged as *guardian*. The second policy catches the violation that a guardian VM exists on the path, but the guardian does not share a security zone with either the start or end VM. The negative edge labeled with + represents an OR condition for the two negative conditions of the VM and guardian zone matching.

Multiple Disjoint Paths: We define a *dependability* zone as a group of infrastructure elements that require mutually redundant fully disjoint paths. The motivation is to prevent single point of failures between dependent infrastructure elements. Fig. 4f shows the corresponding rule as an attack state matching. We are using a universal quantifier with the ability of counting the number of paths between a pair of nodes of the same dependability zone (depzone). The policy is violated if the paths count is less than two, or any redundancy factor that is required.

3.3.2 Infrastructure Policies Summary

We demonstrated a variety of policies ranging from zone isolation, placement of virtual machines, to the prevention of single point of failures. This covers the policy areas of isolation, operational correctness, and failure resilience that have been introduced for virtualized infrastructure policies [3]. We showed the formalization of those policies as graph matches in *GROOVE*, and further explored different ways to express policies, such as negative and positive matching. Besides an expressive and general-purpose approach, the usability is equally important so that end-users, such as auditors of cloud environments, can specify new policies. *GROOVE* offers a graphical editor to develop new production rules, in fact the policies shown in Fig. 4 have been develop graphically and exported as-is. This provides an intuitive and efficient way of specifying new policies.

4. AUTOMATED ANALYSIS

Weatherman provides an automated analysis of configuration and topology changes in virtualized infrastructures. Its architecture, as shown in Fig. 5, obtains all the necessary inputs for the analysis and invokes *GROOVE* as the graph transformation engine. Based on this architecture, we describe two application scenarios for change management as well as for run-time enforcement of security policies and the mitigation of misconfigurations.

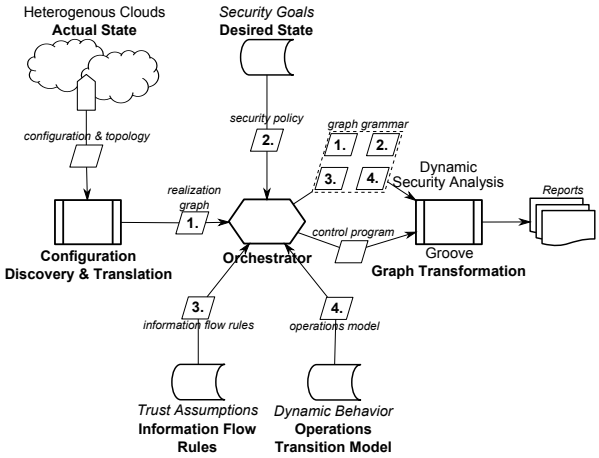


Figure 5: The system architecture consists of i) *Configuration Discovery & Translation* on the left, which extracts the infrastructure configuration and builds the Realization model; ii) the *Orchestrator* in the middle, which prepares the graph grammar for the analysis based on all inputs; iii) and the *Graph Transformation* on the right that employs *GROOVE*.

For obtaining the required inputs, the involvement of the *Weatherman* user is kept to a minimum as we are striving for an automated approach. The rules for the information flow analysis as well as the security policies come as predefined sets, and only in specific cases need to be modified by the user. Security policies may require further input from the user, such that virtual resources need to be assigned to security zones for the zone isolation policy.

Run-time Analysis of Changes enables the automated mitigation of misconfigurations and enforcement of security policies. We introduce an *authorization proxy* that acts as a reverse HTTPS proxy in front of the otherwise shielded management host. The proxy intercepts management operations and inspects them for the analysis. The proxy keeps sessions for each logged in administrators and associates the operations with them. Operations and configuration changes are only forwarded by the proxy to the management host if the *Weatherman* analysis indicates no security policy violation. In a secure deployment (cf. §5.1), it allows to protect virtualized infrastructures from malicious adversaries.

The Policy Decision Point (PDP) of the authorization proxy translates intercepted management operations into a change plan in the *GROOVE* control language. We have translation modules for all covered operations. For instance, from an UpdatePortGroup operation the proxy extracts the host, identifying port group name, new VLAN identifier, as well as new port group name. The PDP then delegates the change plan analysis to *Weatherman*. The Policy Enforcement Point (PEP) only accepts the intercepted operations if they are compliant with the policies; otherwise, they are rejected. The authorization proxy refrains from forwarding the management operation in the reject case, i.e., they are not deployed in the actual infrastructure. It signals an error back to the administrator client with the policy violation.

Change Plan Analysis: The goal of the change plan analysis is to support the planning of complex configuration changes and to verify their security compliance. The focus of this complementary approach lies on the planning of potential changes and perform what-if analyses, whereas the run-time analysis inspects the operations that are currently deployed. In fact, change management, and change plans in particular, are often employed as part of IT infrastructure operation workflows and processes. In our case, an administrator drafts a sequence of desired changes that he wants to be provisioned.

The crucial question is: Will the proposed changes render the infrastructure insecure? To answer this question, the administrator submits the change plan to *Weatherman*, which applies the changes to the graph model of the infrastructure and verifies the resulting infrastructure state against the desired security policies. 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 graph model obtained from the application of the changes violates the security goals, the tool notifies the administrator to reject the proposed change plan and provides the analysis output of the matched policy violation 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.

5. EVALUATION

5.1 Security Analysis

The analysis is based on the system model of §2 and the run-time analysis (§4): *Weatherman* is deployed with an authorization proxy (PEP) that intercepts management operations, forwards them to the policy decision point (PDP) for analysis, and which in turn issues an accept/deny decision. We establish a secure deployment that allows to obtain the integrity property based on a small set of assumptions.

1) *Limited Access* [access]: The adversary accesses the virtualized infrastructure through the management interface only, which can be enforced by placing hosts into *lockdown mode* [28] with no privileges to revoke it. Further, this implies that the adversary does neither have physical or root access on the physical hosts, direct access to the hypervisor nor physical access to network and storage. The adversary does not have access as super_admin, who manages the privileges. *Weatherman* and the authorization proxy are deployed in a hardened configuration and thereby placed under [access]. In practice the hardening can be further achieved by reducing the attack surface of the deployment, e.g., by using a hardened hypervisor, no multi-tenancy, and attestation.

2) *Network Isolation* [netisolation]: The management network is isolated from adversarial access, which implies that the management host cannot be accessed by the adversary directly, but only through the authorization proxy. We call the network between authorization proxy and management host $\mathsf{net}_{\mathsf{sec}}$, either enforced 1) as dedicated physical network, 2) as VLAN in the physical switch, where virtualization administrators do not have access, or 3) as a virtual network with a dedicated VLAN identifier, where the administrators do not have privileges to change it. *Weatherman* and the authorization proxy are deployed in $\mathsf{net}_{\mathsf{sec}}$ and their communication with the management host is covered by [netisolation].

3) *Authentic View and Faithful Model* [authenticview]: *Weatherman* has an authentic view of the topology and configuration of the infrastructure as well as a faithful model of it, including the consequences of management operations. This condition stems from the modeling approach introduced in §3.1: The Realization model provides a faithful graph representation derived from the actual configuration as the structure is encoded there. The operations model captures how individual management operations change the state of the infrastructure and thereby the Realization model.

Definition 4 (Integrity of Run-time Analysis) *If a set of management operations S has been provisioned to the virtualized infrastructure, then* Weatherman *has previously verified S with respect to the specified security goals and issued an* accept *decision and the management host consequently provisioned S.*

Proof Sketch. We pursue the argument by back-tracking starting from a set of management operations S received at the management host. **1.** *Integrity of communication:* We know that the management network $\mathsf{net}_{\mathsf{sec}}$ between management host, authorization proxy and analysis is covered by [netisolation] and gain integrity on S and on topology data. As the management host received S at the management network, it must have been forwarded by the authorization proxy upon an accept decision from the analysis (PDP). The analysis thereby must have verified S under the given security policy and issued an accept decision. **2.** *View equivalence on the topology:* From the assumption [authenticview], we obtain both the faithful Realization model of the topology and representation of consequences in the operations model as necessary conditions. Given that authentic view and faithful model, the tool can only have issued an accept decision, if none of the alarm states defined in the security policy matched the what-if state of the topology amended with the management operations of S. **3.** *View equivalence on S: Weatherman* and the authorization proxy are protected from the adversary's direct influence by [access]. The management operations S are transferred between authorization proxy and *Weatherman* with integrity, by which *Weatherman* analyzes the very same S as staged for provisioning at the authorization proxy. We have that the S received at the management host must have been the same submitted at the authorization proxy and analyzed by *Weatherman*, which could only have been forwarded if a what-if analysis did not match an alarm state. **4.** *Exclusive provisioning through the management host:* Finally, given the [access] condition, we have that management operations can only be provisioned through the management host and that the adversary cannot access hypervisors and physical hosts directly. Thereby, S must have been provisioned by the management host itself after the verification and an accept decision. □

Discussion: The run-time analysis (§4) offers protection against malicious insiders, while the change-plan analysis (§4) offers non-malicious administrators a way to verify changes before provisioning. This approach benefits system availability, since honest administrators can evaluate their change plans pro-actively to gain confidence that their changes will not be denied at run-time. The security analysis hinges on the authentic view of *Weatherman* when it comes to the topology structure and the consequences of management operations. Even though §3.1 seeks to establish a faithful representation and a systematic approach to validate to model against reality, it is still the case that *"the map is not the territory"*. The Realization and operations models are likely to suffer from

subtle differences to the real configuration. Furthermore, the effectiveness of *Weatherman*'s analysis largely depends on the quality of the input specifications: First, the information flow rules represent the trust assumptions on isolation properties and determine which components are assumed to pass on information. Second, *Weatherman* only finds attack states for configuration changes as provided in the specified security policy. Its analysis offers a model checking for these attack states; it does not constitute a security proof.

Security Testing: Besides arguing about the security of our system, we systematically test its ability to detect known violating operations and differentiate them from non-violating ones, on the operations set of §5.2. For each operation, we probabilistically select parameters either from a set of violating or non-violating ones. We issue the operation to the authorization proxy with an expectation that for the violation case we obtain a *reject* decision with a particular policy violation as the reason. Otherwise, for non-violating parameters, the operation should be accepted. *Weatherman* detected all violation cases and behaves as expected. Clearly, security testing and modeling are going hand-in-hand as an iterative process, in which we make the experience that corner cases discovered in security testing serve well to improve the model and to close the maps-territory gap.

5.2 Scalability and Optimizations

In a semi-production environment with 100 VMs, we measured an analysis time of *Weatherman* in the order of 500ms (cf. Fig. 6) for the operations of Table 1, which is suitable for run-time analysis. We further studied the scalability of *Weatherman* with a VMware infrastructure simulator, which is part of the official VMware vCenter server appliance. For a simulated environment with 1000 VMs, which resulted in a Realization model graph with 4121 vertices and 6140 edges, we obtained an overall analysis time of 253s for finding a violation in a UpdatePortGroup operation. This makes our approach suitable for the change plan analysis, but causes a long blocking in a run-time analysis. In a simulated environment with 10000 VMs (41201 vertices, 61400 edges) *GROOVE* ran out of available memory.

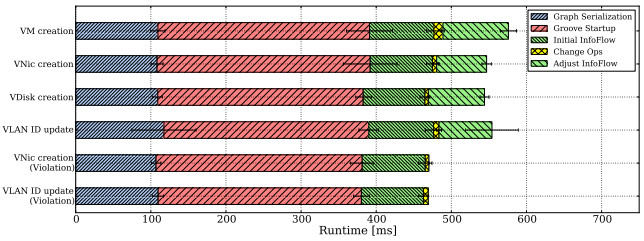


Figure 6: Time measurements for the analysis of a variety of operations, including two violating ones (the last two). We measure the times for the graph serialization, *GROOVE* start-up, initial and adjusting information flow analysis, as well as applying the change operation.

We stress that establishing the models, methodology, and analysis system has been the primary focus of this paper, and not providing an optimized and scalable analysis. We now outline multiple directions of optimizations and scalability improvements. A short-term optimization is to reduce the size of the Realization model graph by removing nodes of types that are not addressed by production rules of the grammar. Possible long-term optimizations are to transform *GROOVE* graph grammars into native code (an approach employed by GrGen [8]) and to exploit a parallel processing of production rules (in particular for rules with universal quantifier and the confluent simple information flow rules).

6. RELATED WORK

Misconfigurations in networks have been a problem in the operation of IT environments for a long time and solutions have been proposed. Mahajan et al. [18] studied misconfigurations in BGP routing configuration changes by listening to changes and assess these. Kim et al. [16] analysed the evolution of network configurations by mining a repository of network configuration files. With the rise of software-defined networking, real-time monitoring and policy checking have been achieved in these environments [13, 14]. In dynamic virtualized infrastructures, *vQuery* [26] monitors configuration changes in VMware environments and assess these changes with regard to performance implications. Schiffman et al. [25] proposed a monitoring system called *Cloud Verifier* that allows to monitor hosts and virtual machines with regard to integrity requirements based on trusted computing mechanisms. Overall, these approaches work in a *reactive* way, i.e., they assess changes in the infrastructure *after* they have happened, whereas we aim for complementary *proactive* mitigation of misconfigurations.

Trustworthy hypervisors, such as sHype [24], offer strong guarantees and mechanisms of isolation between VMs on a single physical system. Our user-configurable information flow rules can capture the different trust assumptions in the isolation of the hypervisor, and can embed the hypervisor isolation into the larger context of virtualized infrastructure isolation. The Trusted Virtual Datacenter (TVDc) [2] offers isolation and integrity by leveraging a trustworthy hypervisor, trusted computing, and automated setup of network isolation. *Weatherman* is complementary to that by i) providing a secondary control mechanisms that checks if the infrastructure is deployed according to a high-level security policy; ii) verifying the changes that are automatically performed by the TVDc system; and iii) providing checks for further policies such as the mitigation of single point of failures. The security of VM images [29] is an important part of the virtualized infrastructure security, but it is orthogonal to our work that focused on the security of the topology.

A model-based approach for configuration management has been proposed in [19] that formalizes network configurations in first-order logic and employs *Alloy* [11] for model finding, in order to detect configuration errors. The model is limited to network configurations, whereas our model covers the entire virtualized infrastructure and provides a fine-grained model of management operations. Similarly, the verification of change operations in the context of statically and dynamically routed networks has been studied [10]. Kikuchi et al. analyze cloud infrastructure changes using Alloy [15], where changes are manually specified. Our approach can automatically analyze changes at runtime using our operations model. Further, we perform an information flow analysis to determine isolation properties. The analysis of firewall policies, e.g., using model-checking [12], provide a complementary approach and covers higher levels of the networking stack.

Graph transformations and in particular *GROOVE* have found applications in other security-related scenarios. A secu-

rity case study has been presented in [9] that deals with the graph-based modeling of physical and digital environments. The modeling and analysis of role-based access control systems has also been achieved using graph transformations [17]. This demonstrates the generality of graph transformations to a wide variety of security application domains.

7. CONCLUSIONS

In this work we address the problem of misconfigurations and resulting security failures in virtualized infrastructures. Our solution consists of a practical tool called *Weatherman* that employs a formal model of cloud management operations, an information flow analysis to determine isolation properties, and a policy verifier in order to proactively assess infrastructure changes with regard to their security impact. For instance, we are able to detect and mitigate changes that would i) break the network isolation of tenants, ii) create virtual machines in the wrong location, and iii) introduce single point of failures. We offer the run-time enforcement of security policies as well as change planning for what-if analyses. While for concreteness we focus in this paper on a particular practical system and goals, we believe that our work is a first step towards a general verification methodology for virtualized infrastructures. One key aspect of our approach is the use of graph rewriting, which offers an expressive and intuitive method for formalizing the operations, information flow analysis, as well as policies.

As part of future work, we consider the integration of access control with our operations transition model, where we extend our existing operations model with required privileges. Given a set of users and their privileges, we can then model-check which operations can be issued by the users that may result in an insecure state.

Acknowledgments

This work is partially supported by the EU H2020 projects SUPER-CLOUD (grant No. 643964) and PrismaCloud (grant No. 644962), and Swiss Secretariat for Education, Research and Innovation (contract No. 15.0025). We thank our shepherd John McDermott and the anonymous reviewers for insightful comments, as well as Arend Rensink for supporting us with *GROOVE*.

8. REFERENCES

[1] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on 1*, 1 (jan.-march 2004), 11 – 33.

[2] Berger, S., Cáceres, R., Pendarakis, D., Sailer, R., Valdez, E., Perez, R., Schildhauer, W., and Srinivasan, D. Tvdc: managing security in the trusted virtual datacenter. *SIGOPS Oper. Syst. Rev. 42* (January 2008), 40–47.

[3] 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.

[4] 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.

[5] Bleikertz, S., Gross, T., and Vogel, C. Cloud Radar: Near Real-Time Detection of Security Failures in Dynamic Virtualized Infrastructures. In *Annual Computer Security Applications Conference (ACSAC 2014)* (Dec 2014), ACM.

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

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

[8] 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* (2006), Springer, pp. 383–397.

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

[10] Hagen, S., Seibold, M., and Kemper, A. Efficient verification of IT change operations or: How we could have prevented Amazon's cloud outage. In *Network Operations and Management Symposium* (April 2012), pp. 368–376.

[11] Jackson, D. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol. 11* (April 2002), 256–290.

[12] Jeffrey, A., and Samak, T. Model Checking Firewall Policy Configurations. In *Proceedings of the 10th IEEE International Conference on Policies for Distributed Systems and Networks* (2009), POLICY'09, IEEE Press, pp. 60–67.

[13] Kazemian, P., Chang, M., Zeng, H., Varghese, G., McKeown, N., and Whyte, S. Real Time Network Policy Checking Using Header Space Analysis. In *10th USENIX Symposium on Networked Systems Design and Implementation* (2013), pp. 99–111.

[14] Khurshid, A., Zou, X., Zhou, W., Caesar, M., and Godfrey, P. B. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *10th USENIX Symposium on Networked Systems Design and Implementation* (2013), pp. 15–27.

[15] Kikuchi, S., and Hiraishi, K. Improving reliability in management of cloud computing infrastructure by formal methods. In *Network Operations and Management Symposium (NOMS), 2014 IEEE* (May 2014), pp. 1–7.

[16] Kim, H., Benson, T., Akella, A., and Feamster, N. The Evolution of Network Configuration: A Tale of Two Campuses. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (2011), IMC '11, pp. 499–514.

[17] Koch, M., Mancini, L. V., and Parisi-Presicce, F. A Graph-based Formalism for RBAC. *ACM Trans. Inf. Syst. Secur. 5*, 3 (Aug. 2002), 332–365.

[18] Mahajan, R., Wetherall, D., and Anderson, T. Understanding BGP Misconfiguration. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2002), SIGCOMM '02, pp. 3–16.

[19] Narain, S. Network Configuration Management via Model Finding. In *Proceedings of the 19th conference on Large Installation System Administration Conference - Volume 19* (2005), LISA '05, pp. 15–15.

[20] Oppenheimer, D., Ganapathi, A., and Patterson, D. A. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4* (2003).

[21] Rensink, A., and Kuperus, J.-H. Repotting the geraniums: on nested graph transformation rules. In *Graph transformation and visual modelling techniques* (2009), vol. 18 of *Electronic Communications of the EASST*.

[22] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), pp. 199–212.

[23] Rozenberg, G., Ed. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*, vol. 1. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

[24] Sailer, R., Jaeger, T., Valdez, E., Caceres, R., Perez, R., Berger, S., Griffin, J. L., and Doorn, L. v. Building a mac-based security architecture for the xen open-source hypervisor. In *Proceedings of the 21st Annual Computer Security Applications Conference* (2005), pp. 276–285.

[25] Schiffman, J., Sun, Y., Vijayakumar, H., and Jaeger, T. Cloud Verifier: Verifiable Auditing Service for IaaS Clouds. In *Proceedings of the IEEE 1st International Workshop on Cloud Security Auditing (CSA 2013)* (June 2013).

[26] Shafer, I., Gylfason, S., and Ganger, G. R. vQuery: a Platform for Connecting Configuration and Performance. *VMware Technical Journal 1*, 2 (Dec. 2012).

[27] VMware. vSphere 5.0 API Reference, Aug 2011. http://pubs.vmware.com/vsphere-50/topic/com.vmware.wssdk.apiref.doc_50/right-pane.html.

[28] VMware. vSphere Security, ESXi 5.5, vCenter Server 5.5 (EN-001164-04), 2013.

[29] Wei, J., Zhang, X., Ammons, G., Bala, V., and Ning, P. Managing Security of Virtual Machine Images in a Cloud Environment. In *Proceedings of the ACM Workshop on Cloud Computing Security* (2009), CCSW '09, ACM, pp. 91–96.