Automated Information Flow Analysis of Virtualized Infrastructures

Sören Bleikertz[1], Thomas Groß[1], Matthias Schunter[1], Konrad Eriksson[2]

[1] IBM Research - Zurich
sbl,tgr,mts@zurich.ibm.com
[2] InfraSight Labs
konrad.eriksson@infrasightlabs.com

Abstract. The use of server virtualization has been growing steadily, but many enterprises still are reluctant to migrate critical workloads to such infrastructures. One key inhibitor is the complexity of correctly configuring virtualized infrastructures, and in particular, of isolating workloads or subscribers across all potentially shared physical and virtual resources. Imagine analyzing systems with half a dozen virtualization platforms, thousands of virtual machines and hundreds of thousands of inter-resource connections by hand: large topologies demand tool support.
We study the automated information flow analysis of heterogeneous virtualized infrastructures. We propose an analysis system that performs a static information flow analysis based on graph traversal. The system discovers the actual configurations of diverse virtualization environments and unifies them in a graph representation. It computes the transitive closure of information flow and isolation rules over the graph and diagnoses isolation breaches from that. The system effectively reduces the analysis complexity for humans from checking the entire infrastructure to checking a few well-designed trust rules on components' information flow.

1 Introduction

Large-scale virtualized infrastructures and cloud deployments are a common and still growing phenomenon. The goals of server virtualization include high utilization of today's hardware, fast deployment of new virtual machines and load balancing through migration of existing virtual machines. Virtualized infrastructures provide standardized computing, virtual networking, and virtual storage resources. Correspondingly, infrastructure clouds provide simple machine creation and migration mechanisms as well as seemingly unlimited scalability, while the costs incurred are only proportional to the resources actually used.

The growth of IT infrastructures and the ease of machine creation have lead to substantial numbers of servers being created (server sprawl). Furthermore, then led to large and complex configurations that arise by rank growth and evolution rather than by advance planning and design. Indeed, the configuration complexity often exceeds the analysis and management capabilities of human administrators. We depict an example for a mid-size infrastructure in Figure 1. This, by itself, calls for automated security analysis of virtualized infrastructures. The high complexity of an analysis is amplified when considering security properties such as isolation, because then the analysis of individual resources must be complemented with an analysis of their composition.

In addition, virtualization providers often aim at establishing multi-tenancy, that is, the capability to host workloads from different subscribers on the same infrastructure. Also, they provide an open environment, in which arbitrary subscribers can register without trust between subscribers being justified. Therefore, we need to

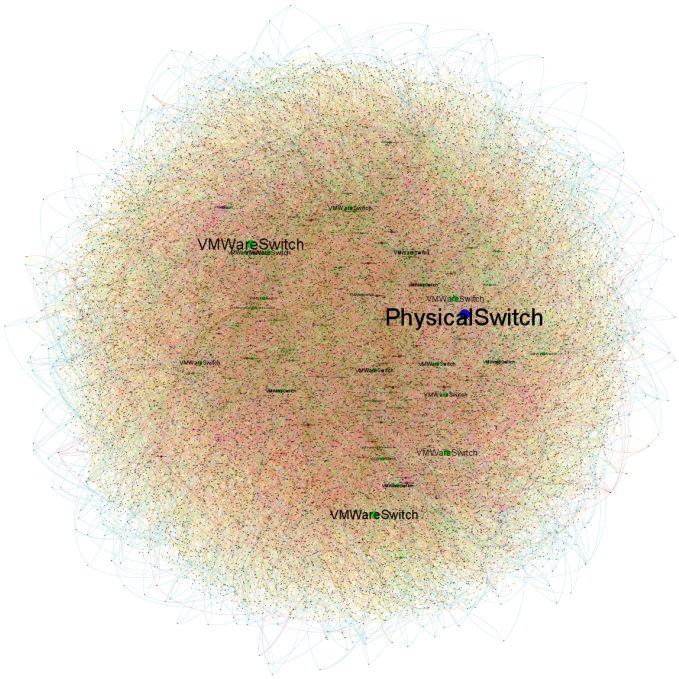


Fig. 1. Illustration of the overwhelming complexity of a mid-size infrastructure with 1,300 VMs.

assume that workloads as well as VMs are under the control of an adversary, and that an adversary will use overt and covert channels in its reach.

Industry partially approaches isolation with automated management and deployment systems constraining the users' actions. However, these mechanisms can fail, lack enforcement, or be circumvented by human intervention.

1.1 Contributions

The goal of this paper is to study automated information flow analysis for large-scale heterogeneous virtualized infrastructures. We aim at reducing the analysis complexity for human administrators to the specification of a few well-designed trust assumptions and leave the extrapolation of these assumptions and analysis of information flow behavior to the tools.

We propose an information flow analysis tool for virtualized infrastructures. The tool is capable of discovering and unifying the actual configurations of different virtualization systems (Xen, VMware, KVM, and IBM's PowerVM[1]) and running a static information flow analysis based on explicitly specified trust rules. Our analysis tool models virtualized infrastructures faithfully, independent of their vendor, and is efficient in terms of absence of false negatives as well as adjustable false positive rates.

Our approach transforms the discovered configuration input into a graph representing all resources, such as virtual machines, hypervisors, physical machines, storage and network resources. The analysis machinery takes a set of graph traversal rules as additional input, which models the information flow and trust assumptions on resource types and auxiliary predicates. It checks for information flow by computing a transitive closure on an information flow graph coloring with the traversal rules as policy. From that, the tool diagnoses isolation breaches and provides refinement

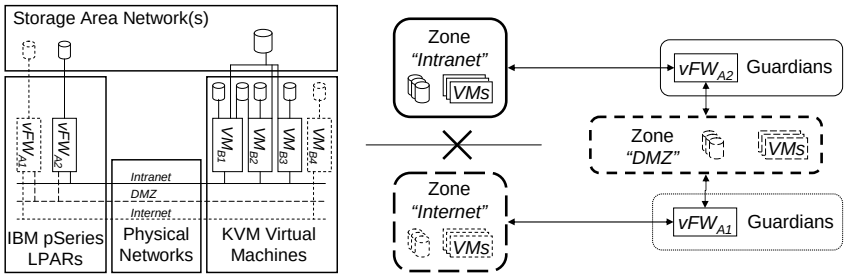
for a root causes analysis. The challenge of information flow analysis for virtualized infrastructures lays in the faithful and complete unified modeling of actual configurations, a layered analysis that maintains completeness and correctness through all stages, and a suitable refinement to infer the root causes for isolation breaches.

Our method applies strict over-abstraction to minimize false negatives. This means that we only assume absence of flows for components that are known to isolate. This enables us to reduce the analysis correctness to the correctness of the traversal rules. As this method accepts an increase in the false positive rate, we allow administrators to fine-tune the trust assumptions with additional traversal rules and constraint predicates to obtain a suitable overall detection rate.

We report on a case study for a mid-sized infrastructure of a financial institution production environment in §7.

1.2 Applications

Our technique is applicable to the isolation analysis of complex configurations of large virtualized datacenters. Such datacenters include different types of server hardware, implementations of virtual machine monitors, as well as physical and virtual networking and storage resources.



(a) Architecture Overview (b) Isolation Zones

Fig. 2. An example setup of a virtualized datacenter with an isolation policy for three virtual security zones.

Let us consider a simplified version of such a configuration in Figure 2(a). This simplified version includes the following hardware: A IBM pSeries[1] server, an x86 server, a virtual networking infrastructure providing VLANs, and a Storage Area Networking providing virtual storage volumes. The virtual resources (networks, storage, machines, and virtual firewalls) are depicted inside these hardware resources. Keep in mind that sizable real-world configurations contain thousands of virtual machines and hundreds of thousands of connections.

Figure 2(b) depicts a desired isolation topology for this example: we have three example virtual security zones "Intranet", "DMZ", and "Internet". Furthermore, we permit communication between Intranet and DMZ that is mediated by a trusted guardian, such as a virtual firewall vFW_{A2}. Similarly, firewall vFW_{A1} moderates and restricts the communication between the Intranet and Internet zones, respectively. The isolation analysis must check that there do not exist components that connect two zones or are shared by two zones, while not being trusted to sufficiently mediate direct and covert information flows.

Note that we focus on validating the virtualized infrastructure's configuration. Once we have guaranteed that no undesired information flow exists except through the specified guardians, we would need to employ techniques from firewall filtering analysis, e.g. [17,18,28], to ensure that the guardians have been configured correctly.

2 Related Work

Virtual systems introduce several new security challenges [7]. Two important drivers that inspired our work are the increase of scale and the transient nature of configurations that render continuous validation more important.

The first area of related work is security of virtual machine monitors. This knowledge is needed to underpin the user's individual decisions whether to trust a given component. Analysis of well-known attacks such as jailbreaks [27] allows one to detect vulnerable configurations. This includes information leakage vulnerabilities of today's infrastructure clouds that allow covert or overt communication between multiple tenants that should be isolated. Examples include co-hosting validation [21] and cache-based side channels [1,20].

To our knowledge, there do not exist any research contributions on the static high-level information flow analysis in virtualized infrastructures. Still, we draw inspiration from information flow analysis such as research in separation [12,23], channel control [22], and non-interference [8,9,16,24]. We discuss these influences on our own definition on structural information control in §3. More often than not, we find research in this space focused on the information flow between high and low variables and not on the information flow in larger topologies.

A second area of related work is *reachability analysis* in networks and the related configuration analysis of firewalls. Our isolation analysis draws from known work on reachability analysis. Analyzing firewalls and complex network infrastructures allows one to decide to what extent two known networks are connected. In particular, Al-Shaer *et al.* [2,13,29] analyze entire network infrastructures including packet filters, transformers, and routers. In [5], it was shown how reachability analysis can be applied to infrastructure clouds. *Firewall configuration analysis* allows the understanding and validation of firewall rules [17,18,28]. While this work focuses on the TCP/IP level, our goal is to ensure 'physical' isolation by ensuring that VLANs and virtual networks are disjoint. This approach is similar to the approach proposed in [14]. If L2 networks are connected while isolation is implemented on the TCP/IP level, we see potential to further extend our work by using these concepts for TCP/IP isolation analysis that is then fed into our analysis concept.

This area of research is also important for modeling the behavior of imperfect guardians. Whereas this paper assumes that guardians always make correct decisions and stop dangerous information flow, reachability and firewall configuration analyses allow one to model imperfections as explicit traversal rules. Guardians with packet inspection and stateful analysis may even discover illegal information flow hidden in legal flows.

Whereas earlier security analyses considered stand-alone elements of a virtualized infrastructure, a tool-supported information flow analysis of a full virtualized environment is still missing, not to speak of complex heterogeneous and large-scale virtualized infrastructures with a diversity of underlying platforms. The research areas of information flow and reachability analysis underpin our efforts, yet so far have not produced a mechanized approach for this problem statement.

3 A Model for Isolation Analysis

In our work, we consider *overt* and *covert* channels. *Covert channels* [15] are not intended for information transfer at all, yet seem to be a common phenomenon in virtualized infrastructures. Requiring the absence of all covert channels from hypervisors, physical hosts and resources will render many resulting system impractical. Therefore, we allow administrators to specify a certain amount of covert channel information flow as tolerable.

In the quest for a suitable requirements definition, we review information flow types [25,15,21,8,9,24,16,10,23,12,11] in Appendix A. At this point, we note i. that we need intransitivity and ii. that *channel control* [22] captures our requirement to specify *exceptions* to the general zoning requirements. Thus, we introduce a property we call *structural information control* that essentially lifts channel control to topology:

Definition 1 (Structural Information Control). *A static system topology provides* structural information control *with respect to a set of information flow assumptions on system nodes if there does not exist an inter-zone information flow unless mediated by a dedicated guardian.*

Observe that we aim at the detection of isolation breaches (information flow traces), which renders our approach loosely similar to model checking, and not at the verification of absence of information flow, which would be similar to theorem proving.

3.1 Modeling Isolation

Modeling Configurations Our static information flow analysis is graph-based. Each element of a virtualization configuration is represented by (at least) one vertex (VMs, VM hosts, virtual storage, virtual network). Connections between elements are represented by edges in the graph and model *potential* information flow. Note that our approach requires completeness of the edges: While not all edges may later actually constitute information flows, we require that all relations that allow information flow are actually modeled as an edge.

The vertices of the graph are typed: our model distinguishes VM nodes, VM host nodes, storage and network nodes, etc.

Definition 2 (Graph Model). *Let $\mathbb{T} \subset \Sigma^+$ a set of vertex types and $\mathbb{P} \subset \Sigma^+$ a set of vertice properties. A virtualization graph model contains a set of typed vertices $V \subset \mathbb{V} := (\Sigma^+ \times \mathbb{T} \times \mathbb{P})$ and a set of edges $E \subseteq (V \times V)$. A vertice v is a triple of label, type and properties set $(l, t, p) \in \mathbb{V}$. An edge e is a pair of start and end vertice $(v_i, v_j) \in (V \times V)$. A set of edges E is called* valid *with respect to a set of vertices V', if $E \subseteq (V' \times V')$. A graph $(\bar{V}, \bar{E})$ is called a* valid subgraph *of graph (V, E), if $\bar{V} \subseteq V$ and $\bar{E} \subseteq E$ is valid with respect to $\bar{V}$. An edge set $\boldsymbol{E} \subseteq E$ is called a* path *if the edges and their respective vertices form a connected valid sub-graph of (V, E).*

We represent complex structures of the virtualization infrastructure by subgraphs of multiple vertices. For instance, we construct guardians such as firewalls with complex information flow rules by a firewall vertex connected to multiple port vertices.

Information is output at one or more *information source* nodes, propagates according to *traversal rules* along the nodes and edges of the graph, and is consumed at an *information sink*. We treat information sources as independent and information as untyped and unqualified.

Definition 3 (Information Sources and Sinks). *For a set of vertices V, we define a set of information sources $\hat{V} \subseteq V$ and a set of information sinks $\check{V} \subseteq V$. A vertice $\hat{v} \in \hat{V}$ is called* information source*, a vertice $\check{v} \in \check{V}$* information sink*.*

Modeling Information Flow Assumptions A *traversal rule* models an assumption on information flow from one vertex type to another vertex type. For instance, a traversal rule will specify that if a VM host is connected to a storage provider, this edge constitutes a direct information flow and is to be traversed. Also, a traversal

rule may specify that if two VMs are connected to the same VM host, this implies the risk of covert channel communication and, therefore, constitutes an information flow.

Definition 4 (Traversal Rules). *For the set of vertice types $\mathbb{T} \subset \Sigma^+$ and a set of vertice properties $\mathbb{P} \subset \Sigma^+$, the* traversal rules *are a propositional function of source type, destination type, source properties, and destination properties over a type relation R and a predicate P:*

$$f_{\mathbb{T},\mathbb{P}} : (\mathbb{T} \times \mathbb{T} \times \mathbb{P} \times \mathbb{P}) \to \{\mathsf{stop}, \mathsf{follow}\} :$$
$$f_{\mathbb{T},\mathbb{P}}(t_i, t_j, p_i, p_j) := \begin{cases} (t_i, t_j) \in R \wedge P(p_i, p_j) & \mathsf{follow} \\ (t_i, t_j) \notin R \vee \neg P(p_i, p_j) & \mathsf{stop} \end{cases}$$

We call traversal rules simple*, if P is always true.*

Definition 5 (Completeness). *For the set of vertice types $\mathbb{T} \subset \Sigma^+$ and a set of vertice properties $\mathbb{P} \subset \Sigma^+$, traversal rules $f_{\mathbb{T},\mathbb{P}}$ are called* complete *if R and P associated to $f_{\mathbb{T},\mathbb{P}}$ are complete. We call a default rule a* completion *of incomplete traversal rules $f_{\mathbb{T},\mathbb{P}}$, if it maps all undetermined cases to either* stop *or* follow*. We call non-default rules* explicit*.*

Whereas completeness is a property of a set of traversal rules, we define *coverage* as in how far a set of traversal rules determines the analysis of a graph deterministically without invoking the default rule.

Definition 6 (Coverage). *For the set of vertice types $\mathbb{T} \subset \Sigma^+$ and a set of vertice properties $\mathbb{P} \subset \Sigma^+$, consider a virtualization graph (V, E) as in Def. 2 and the subset of edges $E' \subseteq E$ that are matched by explicit traversal rules $f_{\mathbb{T},\mathbb{P}}$. We call the quotient of number of explicitly matched edges to total number of edges* coverage*: $c = |E'| \,/\, |E|$*

Observe that a complete coverage, that is, $c = 100\%$ is significant for achieving a low false-positive rate.

The traversal rules specify general assumptions on information flow in virtualized environments and, thereby, embodies a part of the overall trust assumptions. The specification of traversal rules is therefore orthogonal to the isolation policy of a system. Whereas our system comes with a root set of traversal rules as base line trust assumptions, we allow users to specify multiple sets of *user-defined traversal rules* and thereby *user-defined trust assumptions.*

Similar to the tainted variable method for static information flow analysis, we employ the metaphor of color propagation. We associate colors to information sources $\hat{v} \in \hat{V}$ and to vertices that have received information flow from a certain source by the evaluation of traversal rules $f_{\mathbb{T},\mathbb{P}}$. The total information flow of a system is the *transitive closure* of the graph traversal governed by the traversal rules $f_{\mathbb{T},\mathbb{P}}$. This means, that the information flow from any source to any sink can be efficiently statically analyzed by a reachability analysis between source and sink.

We define graph coloring recursively.

Definition 7 (Graph Coloring). *Let traversal rules $f_{\mathbb{T},\mathbb{P}}$, a graph (V, E) and an information source $\hat{v} \in \hat{V} \subseteq V$ with color c be given. Then, $\hat{v}$ is* colored *with c by definition. A vertice $v \in V$ is* colored *with c, if there exists an edge $e = (\cdot, v) \in E$, which is* colored *with c. An edge $e = (v_s, v_d) \in E$ with $v_s = (\cdot, t_s, p_s)$ and $v_d = (\cdot, t_d, p_d)$ is* colored *with c iff (i) v_s is* colored *with c and (ii) $f_{\mathbb{T},\mathbb{P}}(t_s, t_d, p_s, p_d) =$* follow*.*

4 Isolation Analysis of Virtual Infrastructures

We apply the foundations from the preceding section to virtualized infrastructures. Our approach (see Figure 3) consists of four steps organized into two phases: 1) building a graph model from platform-specific configuration information and 2) analyzing the resulting model. The graph model is formally defined in Def. 2.

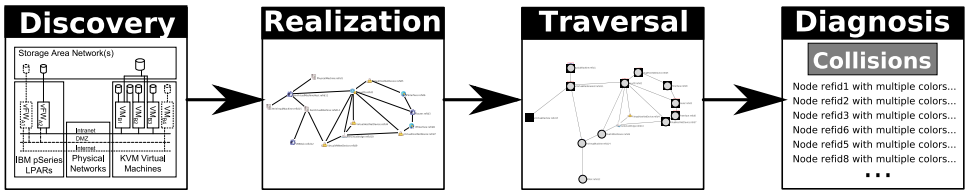


Fig. 3. Overview over the analysis flow.

The first phase of building a graph model is realized using a discovery step that extracts configuration information from heterogeneous virtualized systems, and a translation step that unifies the configuration aspects in one graph model. For the subsequent analysis, we apply the graph coloring algorithm defined in Def. 7 parametrized by a set of traversal rules and a zone definition. The assessment of the resulted colored graph model enables a diagnosis of the virtualized infrastructure with respect to isolation breaches.

4.1 Discovery

The goal of the discovery phase is to retrieve sufficient information about the configuration of the target virtualized infrastructure. To this end, platform-specific data is obtained through APIs such as VMware VI, XenAPI, or libVirt, and then aggregated in one discovery XML file. The target virtualized infrastructure, for which we will discover its configuration, is specified either as a set of individual physical machines and their IP addresses, or as one management host that is responsible for the infrastructure (in the case of VMware's vCenter or IBM pSeries's HMC). Additionally, associated API or login credentials need to be specified.

For each physical or management host given in the infrastructure specification, we will employ a set of discover probes that are able to gather different aspects of the configuration. We realized multiple hypervisor-specific probes for Xen, VMware, IBM's PowerVM, and LibVirt. Furthermore, if the management VM is running Linux[1], we also employ probes for obtaining Linux-specific configuration information. Currently, we do not discover the configuration of the physical network infrastructure. However, the framework easily be extended beyond the existing probes or use configuration data from a third-party source.

4.2 Transformation into a Graph Model

We translate the discovered platform-specific configuration into a unified graph representation of the virtualization infrastructure, the *realization model*. The realization model is an instance of the graph model defined in Def. 2. It expresses the low-level configuration of the various virtualization systems and includes the physical machine, virtual machine, storage, and network details as vertices. We generate the realization model by a translation of the platform-specific discovery data. This is done by so-called *mapping rules* that obtain platform-specific configuration data

and output elements of our cross-platform realization model. Our tool then stitches these fragments from different probes into a unified model that embodies the fabric of the entire virtualization infrastructure and configuration.

For all realization model types in $\mathbb{T}$ (cf. Def. 8), we have a mapping rule that maps hypervisor-specific configuration entries to the unified type and, therefore, establishes a node in the realization model graph. We obtain a complete iteration of elements of these types as graph nodes. The mapping rules also establish the edges in the realization model.

This approach obtains a complete graph with respect to realization model types. Observe that configuration entries that are not related to realization model types are not represented in the graph. This may introduce false negatives if there exist unknown devices that yield further information flow edges. To test this, we can introduce a default mapping rule to include all unrecognized configuration entries as dummy nodes.

4.3 Coloring through Graph Traversal

The graph traversal phase obtains a realization model and a set of information source vertices with their designated colors as input. According to Def. 7, the graph coloring outputs a colored realization model, where a color is added to a node if permitted by an appropriate traversal rule. We use the following three types of traversal rules (see Def. 4 and the definition of traversal rules below) that are stored in a ordered list. We apply a first-matching algorithm to select the appropriate traversal rule for a given pair of vertices.

Flow rules model the knowledge that information can flow from one type of node to another if an edge exists. For example, a VM can send information onto a connected network. These rules model the "follow" of Def. 4. *Isolation rules* model the knowledge that certain edges between trusted nodes do not allow information flow. For example, a trusted firewall is known to isolate, i.e., information does not flow from the network into the firewall. These rules model the "stop" of Def. 4. *Default rule* means that ideally, either isolation or else flow rules should exist for all pairs of types and all conditions, that is, we want to achieve complete coverage according to Def. 6: For any edge and any two types, the *explicit traversal rules* should determine whether this combination allows or disallows flow. In practice, the administrator may lack knowledge for certain types. As a consequence, we included a default rule as *completion*. Here, we establish a *default flow rule*: whenever two types are not covered by an isolation or flow rule, then we default to "follow". To be on the safe side, i.e., reducing false negatives, we assume that flow is possible along this unknown type of edges.

Given this set of rules, we then traverse the realization model by applying the set of traversal rules and color the graph according to information flows from a given source. The traversal starts from the information sources and computes the transitive closure over the traversal rule application to the graph.

4.4 The Traversal Rules

The graph coloring algorithm requires a set of traversal rules that model information flows, isolation properties, and trust assumptions. We will propose a set of rules and explain their purposes, and leave the correctness argument to the security analysis in §5.2 and a detailed discussion in Appendix B.2.

Definition 8 (Traversal Rule). *Let F be a set of follow types* $\{\mathsf{stop}, \mathsf{follow}\}$*,* $\mathbb{T}' \subset \mathbb{T}$ *be a set of realization model types* {*Port, NetworkSwitch, PhysicalSwitch,*

Table 1. Traversal Rules

#	Type	Flow	Condition +Color Modification
		Trust Rules	
1	stop	$PhysicalSwitch \Rightarrow Port$	Has any *vlan* color
2	stop	$ManagementOS \Leftrightarrow any$	
3	stop	$PhysicalMachine \Leftrightarrow PhysicalDevice$	
4	stop	$VirtualMachine \Leftrightarrow VirtualMachineHost$	
		Network Switches	
5	stop	$Port \Leftrightarrow NetworkSwitch$	Port is disabled
		VLAN	
6	follow	$Port \Rightarrow NetworkSwitch$	Port has VLAN tagging with tag $VLAN + Create *vlan-$VLAN*
7	follow	$NetworkSwitch \Rightarrow Port$	Port's VLAN tag matches color's one +Remove *vlan-$VLAN*
8	follow	$NetworkSwitch \Rightarrow Port$	Port is trunked
9	stop	$NetworkSwitch \Rightarrow Port$	Port's VLAN tag mismatches color's one
10	stop	$NetworkSwitch \Rightarrow Port$	Has any *vlan* color
		Storage	
11	stop	$StorageController \Rightarrow PhysicalDisk$	
12	stop	$FileSystem \Rightarrow File$	
		Default	
13	follow	$any \Leftrightarrow any$	

ManagementOS, PhysicalDevice, VirtualMachine, VirtualMachineHost, StorageController, PhysicalDisk, FileSystem, File, any$\}$*, and* D *be a set of flow directions* $\{\Rightarrow, \Leftarrow, \Leftrightarrow\}$*, where* $\Rightarrow$ *and* $\Leftarrow$ *denote a unidirectional, and* $\Leftrightarrow$ *a bi-directional flow. A traversal rule is a tuple* (f, t_0, t_1, d, P, g) *with* $f \in F$*,* $t_0, t_1 \in \mathbb{T}'$*,* $d \in D$*,* P *is a predicate over properties and colors of the realization model, and* g *is a color modification function. During graph coloring (see Def. 7),* g *can transform the color* c *of a colored vertex* $\hat{v}$ *while coloring a new vertex* v*, i.e.,* $c(v) = g(c(\hat{v}))$*.*

The traversal rules specified in Table 1 are a ordered list of rules (as defined in Def. 8). In case the condition is left empty, a *true* predicate is assumed, and in case the color modification is empty, g is the identity function.

Definition 9 (Matching Rule). *Given a traversal rule* (f, t_0, t_1, d, P, g) *as defined in Def. 8 and a source and destination vertex from the graph traversal:* v_s *and* v_d *respectively. The rule matches iff i)* $(t_0 = t(v_s) \vee t_0 = any) \wedge (t_1 = t(v_d) \vee t_1 = any)$ *where* $t(v)$ *denotes the type of a given vertex* v*, ii)* $d \in \{\Rightarrow, \Leftrightarrow\}$*, iii)* $P = true$*.*

The first-matching algorithm iterates over the ordered list of traversal rules (Table 1) and applies the matching rule defined in Def. 9. If the matching evaluates to true, the iteration stops and the matched rule is returned. The matching of the traversal rules induces a function representation of the traversal rules as defined in Def. 4.

Our trust assumptions are specified in the rules namely, Rule 1, Rule 2, Rule 3, and Rule 4. These model that VLANs are isolated on physical switches, that the privilege VM and the physical machine are trusted and do not leak information, and that we exclude cross-VM covert channels (see §5.2).

Rule 5 simply stops an information flow if a network port is disabled. Rule 6 and Rule 7 model the VLAN en- and decapsulation of network traffic. Traffic with a VLAN tag is modeled as a new color *vlan* with the VLAN tag appended, which is created in case of encapsulation and removed in case of decapsulation. In the case of VMware, the VLAN tag for a VM is modeled as a non-zero *defaultVLAN* property

of the port. Rule 8 specifies that if a port is marked as trunked, which is required in the case of VMware to allow traffic from the VMs to the physical network interface, the VLAN traffic is also allowed to flow. Otherwise, if the *vlan* color tag mismatches the port's VLAN tag, we isolate and stop the information flow (see Rule 9). This also applies to Rule 10, which is the default isolation rule for VLAN traffic, if one of the previous rules did not match.

On the storage side, we model the behavior of the storage controller not to leak information from one disk to another with Rule 11. Furthermore, the filesystem will not leak information from one file to another (Rule 12).

The default rule Rule 13 allows any information flow that was not handled by a previous rule due to the first-matching algorithm.

We make three observations about the traversal rules: *First*, administrators can modify existing and specify further traversal rules, for instance, to relax trust assumptions or to model known behavior of specific components. *Second*, traversal rules serve as generic interface to include analysis results of other information flow tools into the topology analysis (e.g., firewall information flow analysis). *Third*, the behavior of explicit guardians (see Def. 1) is introduced by traversal rules specific to these nodes. For instance, the guardians in the exemplary Figure 2, §1.2, would receive a stop-rule.

4.5 Detecting Undesired Information Flows

The goal of the detection phase is to produce meaningful outputs for system administrators. For detecting undesired information flows, we color a set of information sources that mark types of critical information that must not leak. The idea of the color spill method is to introduce nodes called 'sinks' (see Def. 3). Each sink is colored with a subset of the colors corresponding to the information that it is allowed to receive. In practice, the administrator provides a list of clusters or zones that shall be isolated, and we add/mark sources and sinks according to the isolation policy with respect to these zones. In our example from Figure 2, §1.2, we would mark nodes from the zones "Intranet" ($VM_{B1}, VM_{B2}, VM_{B3}$) and "Internet" ($VM_{B4}$) as sources and the guardians of the opposite zones (vFW_{A1}, vFW_{A2}) as sinks, to determine isolation breaches in both directions. After the transitive closure of the traversal rules, we check whether any additional colors "spilled" into a given sink. If a sink gets connected to an additional color, then we have found a potential isolation breach. You could imagine the dedicated color sinks as a honey pot, waiting for colors from other zones to spill over.

Observe that the detection of a color spill only indicates the existence of a breach and between which zones (source-sink pairs) it has occurred. The color flow can be visualized and of some use for administrators to fix the problem. In addition, we study different refinement methods for root-cause analysis, in order to pinpoint critical edges responsible for the information flow in a industry case study (§7).

5 Security Analysis of the Automated Information Flow Analysis

Definition 10 (System Assumptions). *We assume correctness of discovery/-translation and isolation behavior as defined in §4.4.*

- Completeness of Discovery*: We assume that the configuration output of virtualized infrastructures contains all elements that might solicit information flow (cf. §4.1)*
- Correct Translation Modules*: We assume that the discovery modules analyzing concrete systems are capable to correctly identify configuration elements that translate to vertices and edges in the realization model (cf. §4.2).*

– Hypervisor Separation*: We assume that the hypervisor sufficiently prevents cross-VM information flow through covert channel down to a tolerable level (cf. Rule 4, §4.4).*[3]
– VLAN Separation by Physical Switches*: We assume that physical switches isolate different VLANs from each other (cf. Rule 1, §4.4).*

5.1 Reduction to Correctness of the Traversal Rules

The graph coloring establishes the following events through a transitive closure of traversal rules $f_{\mathbb{T},\mathbb{P}}$ over a graph (V, E) with sources $\hat{V}$ and sinks $\breve{V}$.

Definition 11 (Events). Wlog., *we model admissible colors of an information sink $\breve{v} \in \breve{V}$ as colors associated with $\breve{v}$. Then we have:*

– *We call an event* B *an* isolation breach *if a information sink $\breve{v} \in \breve{V}$ is colored with an inadmissible color of a information source $\hat{v} \in \hat{V}$ such that $\hat{v} \neq \breve{v}$.*
– *We call an event* A *an* alarm *if an isolation audit reports an information flow between a distinct information source $\hat{v} \in \hat{V}$ and information sink $\breve{v} \in \breve{V}$.*
– *We call the set of events* $\neg\mathsf{A} \wedge \mathsf{B}$ *a* false negative.
– *We call the set of events* $\mathsf{A} \wedge \neg\mathsf{B}$ *a* false positive.

Corollary 1 (Structural Information Control). *Under the assumptions from Def. 10 and correct traversal rules $f_{\mathbb{T},\mathbb{P}}$, from the absence of false negatives in an isolation analysis ($\neg\mathsf{A} \wedge \mathsf{B} = \emptyset$) follows that a breach of structural information control is indicated by an alarm event* A.

Because we modeled the mediation by dedicated guardians explicitly by traversal rules and inter-zone information flow by B events, this is by construction.

Note that the goal of the analysis system is to detect isolation breaches, that is, breaches of structural information control. We cannot prove an absence of information flow, i.e., verify structural information control, but only detect attack states. We optimize the detection thereof by minimizing the false negative rate through reduction to correct traversal rules (making sure we miss as few breaches as possible) and maximizing the Bayesian detection rate through mitigation of false positives (finding the actual needles in the haystack).

Theorem 1 (Reduction to Traversal Rules, proven in Appendix B.1). *The correct isolation modeling by traversal rules $f_{\mathbb{T},\mathbb{P}}$ implies absence of false negatives.*

5.2 Correctness of the given Traversal Rules

The correctness of the traversal rules from Table 1, §4.4 remains to be shown, where we need to analyze on two levels: i. correctness of individual rules and ii. correctness of their composition.

Individual Rules We examine the traversal rules in detail in Appendix B.2 and highlight the most important points here.

– *Network*: We model correct implementation of physical switches (Rule 5), VLAN en- and decapsulation and lift the properties of cryptographic secure channels (e.g., [6,19]) to VLAN tags (Rules 1, and 8, to 10).

[3] This assumption is modeled by the hypervisor traversal rules and can be explicitly specified by administrators.

– *Physical Machine, Hypervisor, VM Stack*: We claim secure isolation by management OS and physical machine (Rules 2 and 3) as well as cross-VM isolation (Rule 4). The former rules are elementary for virtualization security, the latter rule is arguable as it models the hypervisor's multi-tenancy capability and needs to be reconsidered depending on the actual environment (cf. [21,1] and discussion in Appendix B.2).
– *Storage*: We model secure separation by physical disks as well as by the file system (Rules 11 and 12), where the latter rule is systematically enforced by virtualization vendors (e.g., [26]) and can be checked automatically [30].

Correct Traversal Rule Composition The composition establishes the following robustness principles:

– *Explicit Knowledge Model*: The explicit traversal rules model all and only known facts about information flow and isolation. Thus, traversal rules focus on preventing false negatives introduced by invalid assumptions.
– *Strict Over-abstraction*: When in doubt, the traversal rules must be a conservative estimate towards information flow, that is, model a super-set of potential information flow. By that, traversal rules will never introduce false negatives at the cost of additional false positives.
– *Default-Traversal Behavior*: The default rules establishing completion on the traversal rules must all be *default-follow rules*, that is, evaluate undetermined cases to 1 and log such results. Thus, the completion will only introduce false positives but never false negatives.

We conclude that the traversal rule robustness principles are all lined up to fence off false negatives, yet at the cost of false positives. Whereas this trade-off benefits a conservative security analysis, it impacts its effectiveness, as becomes manifest in its *overall detection rate*.

5.3 Overall Detection Rate

The overall detection rate of an analysis system establishes a relation between alarm and breach events, A and B, as defined in Def. 11. We analyze the effectiveness of the analysis system, in particular with respect to rejection of *false positives*, whose influence through the base-rate fallacy rate was established by Axelsson [3] in the area of intrusion detection systems.

Definition 12 (Detection Rates). *The* detection rate *is* $\mathsf{P}[\mathsf{A}|\mathsf{B}]$*, alarm contingent on breach, obtainable by testing the analysis system against scenarios known to constitute a* B *event. The* false alarm rate *is* $\mathsf{P}[\mathsf{A}|\neg\mathsf{B}]$*, the false positive rate, obtainable analogously. The* false negative rate *is* $\mathsf{P}[\neg\mathsf{A}|\mathsf{B}] = 1 - \mathsf{P}[\mathsf{A}|\mathsf{B}]$*. The* Bayesian detection rate *is* $\mathsf{P}[\mathsf{B}|\mathsf{A}]$*, that is the rate with which an alarm event implies an actual breach event. The* all-is-well rate *is* $\mathsf{P}[\neg\mathsf{B}|\neg\mathsf{A}]$*, that is the rate with which the absence of an alarm implies that all is well.*

Our goal is to maximize the Bayesian detection rate and the all-is-well rate, which we determine with Bayes Theorem:

$$\mathsf{P}[\mathsf{B}|\mathsf{A}] = \frac{\mathsf{P}[\mathsf{B}] \cdot \mathsf{P}[\mathsf{A}|\mathsf{B}]}{\mathsf{P}[\mathsf{B}] \cdot \mathsf{P}[\mathsf{A}|\mathsf{B}] + \mathsf{P}[\neg\mathsf{B}] \cdot \mathsf{P}[\mathsf{A}|\neg\mathsf{B}]}$$

If we assume that the rate of breaches is low compared to the rate of non-breaches, $\mathsf{P}[\mathsf{B}] \ll \mathsf{P}[\neg\mathsf{B}]$, we see that the false positive rate $\mathsf{P}[\mathsf{A}|\neg\mathsf{B}]$ dominates the denominator of the Bayesian detection rate. This analysis asks for caution. Even though the focus on absence of false negatives implies a conservative security analysis, the presence of false positives can diminish the effectiveness of the analysis system easily.

Conjecture 1. The correctness of the traversal rules determines the absence of false negative events. The coverage of explicit correct traversal rules determines the absence of false positive events.

Although the absence of false negatives is important for the system's security, effectiveness requires the absence of false positives, as well. This is to ensure that administrators are able to find the actual breaches in the set of all alarm events. Therefore, administrators need to fine-tune the traversal rules to maximize coverage and, thus, the Bayesian detection rate.

5.4 Discussion

The transitive closure over the graph coloring securely lifts the isolation analysis to an analysis of the traversal rules $f_{\mathbb{T},\mathbb{P}}$. Therefore, the correctness of the traversal rules becomes a make-or-break criterion for the analysis method and impacts the detection rate.

We observe a *complexity reduction*: the simple traversal rules have a complexity of their relation $R \subseteq \mathbb{T} \times \mathbb{T}$. In practice, $|\mathbb{T}| \ll |V|$ as well as $|\mathbb{T}|^2 \ll |E| \leq |V|^2$, with the number of properties set for $f_{\mathbb{T},\mathbb{P}}$ being small. Therefore, the complexity of analyzing the traversal rules $f_{\mathbb{T},\mathbb{P}}$ is much smaller than the complexity of isolation analysis. This allows administrators to explicitly model and thoroughly and efficiently check their knowledge and trust assumptions about information flow and isolation.

Because our traversal rules base on the principle of *over-abstraction*, that is, resort to default-traversal in undetermined cases, the method excludes false negatives, at the cost of additional false positives. The method is therefore always on the conservative side, even though we are well aware that the false positive rate impacts the overall detection rate [3]. We provide the general analysis framework and offer user-defined traversal rules to fine-tune the analysis method to reduce false positives and maximize the Bayesian detection rate. Also, we experiment with refinement methods for a subsequent root-cause analysis to pinpoint critical information flow edges.

In principle, our tool is in a similar situation as the first intrusion detection systems. There do not exist standardized data sets to quantify and calibrate false positive and false negative rates. We approach this situation by obtaining real-world data from third parties and are currently testing the analysis method in sizable real-world customer deployments, such as the case study discussed below, to establish the detection rates.

6 Implementation

We have implemented a prototype of our automated information flow analysis in Java[1] that consists of roughly twenty thousand lines of code. Furthermore, we have additional scripts written in Python that perform post-processing for visualization purposes and refinement for root-cause analysis. The prototype consists of two main programs, that is, the discovery, and a processing and analysis program. The result of the discovery is written into an XML file and is used as the input for the analysis.

6.1 Discovery

The functionality of the discovery and its different probes were already outlined in §4.1. There exist different ways to implement a discovery probe. A probe can establish a secure console (SSH) connection to the virtualized host or the management console where commands are executed and the output is processed. Typically, the output is either XML, which is stored in the discovery XML file directly, or the output has

to be parsed and transformed into XML. As alternative to the secure console, a probe can connect to a hypervisor-specific API, such as a web service, that provides information about the infrastructure configuration.

We illustrate the discovery procedure with VMware as example. Here, the discovery probe connects to *vCenter* to extract all configuration information of the managed resources. It does so by querying the VMware API with the retrieveAllTheManagedObjectReferences() call, which provides a complete iteration of all instances of ExtensibleManagedObject, a base class from which other managed objects are derived. We ensure completeness by fully serializing the entire object iteration into the discovery XML file, including all attributes.

6.2 Processing

The *processing* program consists of the transformation of the discovery XML into the realization model, the graph coloring, and the analysis of the colored realization graph.

The realization model is a class model that is used for generating Java class files. During the transformation of the XML into the realization model, instances of these classes are created, their attributes set, and linked to instances of other classes according to the mapping rules (cf. §4.2). Again, we illustrate this process for VMware. Each mapping rule embodies knowledge of VMware's ontology of virtualized resources to configuration names, for instance, that VMware calls storage configuration entries storageDevice. The edges are encoded implicitly by XML hierarchy (for instance, that a VM is part of a physical host) as well as explicitly by Managed Object References (MOR). The mapping rules establish edges in the realization model for all hierarchy links and for all MOR links between configuration entries for realization model types.

The traversal rules used for the graph coloring (cf. §4.3 and §4.4) are specified in XML. Intermediate results, such as the paths of the graph coloring, can be captured and used for further processing, i.e., visualization. We implemented Python scripts that generate input graphs for the *Gephi visualization framework* [4], such as illustrated in Figure 1.

7 Case Study

We launched a case study with a global financial institution for a performance evaluation and for further validation of detection rates and behavior in large-scale heterogeneous environments. The analyzed virtualized infrastructure is based on VMware and consists of roughly 1,300 VMs, the corresponding realization model graph of 25,000 nodes and 30,000 edges. The production system has strong requirements on isolation between clusters of different security levels, such as high-security clusters, normal operational clusters, backup clusters and test clusters. In addition, we can work with a comprehensive inventory of virtualized resources that serves as specification of an ideal state (machine placement, zone designation and VLAN configuration) and as basis for alarm validation.

We examine preliminary lessons learned, where we first consider the operation of the tool itself. The phases discovery, transformation to realization model and graph coloring executed successfully. The visualization of all results presented a challenge as a 25,000-node/30,000-edge graph overburdened the built-in visualization of the tool.

From a performance perspective, the discovery of the infrastructure using the VMware probe in combination with vCenter requires about *seven minutes*, and

[4] `http://gephi.org/`

results in a discovery XML file with a size of *61MB*. The discovery was performed in a production environment, where network congestion and other tools using the same vCenter can have a negative effect on the discovery performance. The overall analysis of the infrastructure using the discovery XML file requires *53 seconds*, where *46 seconds* are spent on the graph coloring. This demonstrates a reasonable performance for the discovery and analysis of a mid-sized infrastructure, such as the one in our case study.

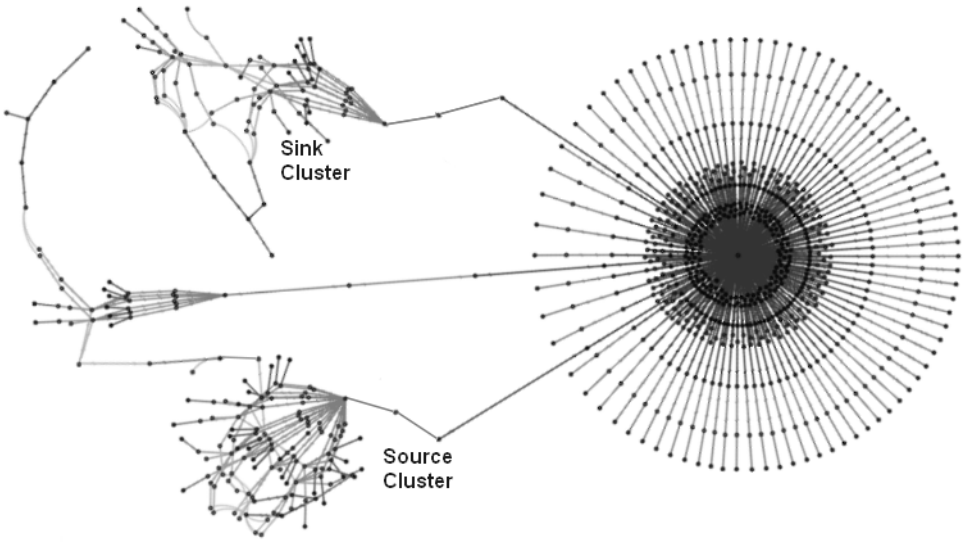


Fig. 4. Root-cause analysis of a source cluster with information flow to a sink cluster. The tree refinement derives only the sub-graphs relevant for an isolation breach. The "flower" is a large-scale switch.

From a security perspective, the tool indeed found several realistic isolation breaches, which we highlighted by adding virtual edges between breached clusters. All isolation breaches constituted potential information flows. By that we could show actual breaches between high-security, normal operational and test clusters. We have furthermore shown that the documentation of the permitted flows was incomplete: One breach that the system identified violated the initial policy given by the customer and was fixed by augmenting the policy.

Root-cause analysis answers the question which edges and nodes are ultimately responsible for the breach. We found that color spill after a traversal to a new cluster may hamper the subsequent root-cause analysis. We therefore introduced multiple automated refinement mechanisms after the graph-coloring phase to support the elimination of classes of potential root causes. *First*, we benefited greatly from a process of elimination, that is, to exclude, for instance, that information has flown over storage edges. *Second*, it was helpful to allow partial coloring, in particular to stop color propagation after detecting a breach to another cluster. *Third*, we introduced a reverse flow tree that captured which path information flow took as prelude to a breach. Figure 4 depicts an example of such a color tree: the tree is a subgraph highlighting a cross-cluster information flow path. *Fourth*, we further refined this tree by extracting critical edges, such as passed VLANs, to pinpoint routes of information flow.

In conclusion, we added a refinement phase driven by reusable Python scripts. We obtained multiple realistic alarms and could trace their root causes. The graph

export to Gephi enabled the efficient visualization of root causes and information flows for human validation.

8 Conclusions & Future Work

We demonstrated an analysis system that discovers the configuration of complex heterogeneous virtualized infrastructures and performs a static information flow analysis. Our approach is based on a unified graph model that represents the configuration of the virtualized infrastructure and a graph coloring algorithm that uses a set of traversal rules to specify trust assumptions and information flow properties in virtualized systems. Based on the colored graph model, the system is able to diagnose isolation breaches, which would violate the customer isolation requirements in multi-tenant datacenters. We showed in our security analysis that we can reduce the correctness and detection rate to the correctness and coverage of the graph traversal rules. Based on existing research and systems knowledge, we submit that the present traversal rules are natural and correct.

To make this a comprehensive solution for validating virtual infrastructures in practice, there are several open questions. The first area of future work is the kind of information flows the system can model. Whereas it is currently restricted to a binary decision, i.e., whether information flows or not, future work may include different flow types (traffic types) and the bandwidth of covert channels. We also pursue research on topological covert channels, i.e., to what extent the composition of components with associated information flow rules exceeds the information flow risks of individual components. Part of this area is the modeling of imperfect guardians. For instance, firewalls let certain kinds of traffic pass and block others. We can build on the results of firewall and reachability analyses discussed in §2 to capture the behavior of such guardians and transform it into detailed guardian- and flow-type-specific traversal rules.

The second open problem is a platform-independent language to express security requirements and trust assumptions. In our current system, isolation assumptions and rules are provided in XML format, and a domain-specific language could ease the specification of requirements for customers. Bleikertz and Groß [4] proposed a first formal language to that end.

A third area of future work is to extend our approach towards other configuration properties, such as dependability. Today, misconfiguration of redundant components (network, disks, machines) often is only detected if the main component fails. We believe that our approach can be used to validate the correct configuration of such backup components to ensure correct fail-over. In addition, dynamic analysis becomes more important with increasing size of the topology and change frequency. Our current approach performs a static analysis of a given configuration state. A dynamic analysis can be emulated by running multiple static analyses and comparing the resulting realization models. However, a truly dynamic analysis needs to analyze small configuration changes and efficiently determine their effect on the topology.

Acknowledgments

We would like to thank Ray Valdez, Michael Steiner, Stefan Berger, and Dimitrios Pendarakis of the IBM Watson Research Center for valuable feedback and productive collaboration during our research. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n257243 (TClouds project: `http:/www.tclouds-project.eu`).

References

1. Aciiçmez, O.: Yet another microarchitectural attack: exploiting i-cache. In: CSAW '07: Proceedings of the 2007 ACM Workshop on Computer Security Architecture. pp. 11–18. ACM (2007)
2. Al-Shaer, E., Marrero, W., El-Atawy, A., ElBadawi, K.: Global Verification and Analysis of Network Access Control Configuration. Tech. rep., DePaul University (2008)
3. Axelsson, S.: The base-rate fallacy and the difficulty of intrusion detection. ACM Trans. Inf. Syst. Secur. 3(3), 186–205 (2000)
4. Bleikertz, S., Groß, T.: A virtualization assurance language for isolation and deployment. In: Proceedings of the 12th IEEE International Symposium on Policies for Distributed Systems and Networks (IEEE POLICY 2011). IEEE (2011)
5. Bleikertz, S., Schunter, M., Probst, C.W., Pendarakis, D., Eriksson, K.: Security audits of multi-tier virtual infrastructures in public infrastructure clouds. In: Proceedings of the 2010 ACM Workshop on Cloud Computing Security. pp. 93–102. CCSW '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1866835.1866853
6. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels (extended abstract). In: Advances in Cryptology: EUROCRYPT 2002. vol. 2332, pp. 337–351. Springer (2002), extended version in IACR Cryptology ePrint Archive 2002/059, http://eprint.iacr.org/
7. Garfinkel, T., Rosenblum, M.: When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. In: HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems. pp. 20–20. USENIX Association, Berkeley, CA, USA (2005)
8. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy. pp. 11–20. IEEE (1982)
9. Gray, III, J.W.: Toward a mathematical foundation for information flow security. In: IEEE Symposium on Security and Privacy. pp. 21–35. IEEE (1991)
10. Haigh, J.T., Young, W.D.: Extending the non-interference version of MLS for SAT. In: IEEE Symposium on Security and Privacy. pp. 60–60. IEEE (1986)
11. Jacob, J.: Separability and the detection of hidden channels. Inf. Process. Lett. 34, 27–29 (February 1990), http://portal.acm.org/citation.cfm?id=79804.79852
12. Kelem, N.L., Feiertag, R.J.: A Separation Model for Virtual Machine Monitors. In: IEEE Symposium on Security and Privacy. pp. 78–86. IEEE (1991)
13. Khakpour, A.R., Liu, A.: Quarnet: A Tool for Quantifying Static Network Reachability. Tech. Rep. MSU-CSE-09-2, Department of Computer Science, Michigan State University, East Lansing, Michigan (January 2009)
14. Krothapalli, S.D., Sun, X., Sung, Y.W.E., Yeo, S.A., Rao, S.G.: A toolkit for automating and visualizing VLAN configuration. In: SafeConfig '09: Proceedings of the 2nd ACM Workshop on Assurable and Usable Security Configuration. pp. 63–70. ACM (2009)
15. Lampson, B.W.: A note on the confinement problem. Communications of the ACM 16(10), 613–615 (1973)
16. Mantel, H.: Information flow control and applications - bridging a gap. In: Oliveira, J.N., Zave, P. (eds.) FME. Lecture Notes in Computer Science, vol. 2021, pp. 153–172. Springer (2001)
17. Marmorstein, R., Kearns, P.: A Tool for Automated iptables Firewall Analysis. In: ATEC '05: Proceedings of the USENIX Annual Technical Conference. pp. 44–44. USENIX Association, Berkeley, CA, USA (2005)
18. Mayer, A., Wool, A., Ziskind, E.: Fang: A Firewall Analysis Engine. In: SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy. p. 177. IEEE, Washington, DC, USA (2000)
19. Mödersheim, S., Viganò, L.: Secure pseudonymous channels. In: Backes, M., Ning, P. (eds.) ESORICS. Lecture Notes in Computer Science, vol. 5789, pp. 337–354. Springer (2009)
20. Percival, C.: Cache missing for fun and profit. http://www.daemonology.net/papers/htt.pdf (May 2005)
21. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: CCS '09: Proceedings of the 16th ACM conference on Computer and Communications Security. pp. 199–212. ACM, New York, NY, USA (2009)

22. Rushby, J.: Design and verification of secure systems. In: Proceedings of the eighth ACM Symposium on Operating Systems Principles. pp. 12–21. SOSP '81, ACM, New York, NY, USA (1981), http://doi.acm.org/10.1145/800216.806586
23. Rushby, J.: Proof of separability a verification technique for a class of security kernels. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming. Lecture Notes in Computer Science, vol. 137, pp. 352–367. Springer (1982)
24. Rushby, J.: Noninterference, transitivity, and channel-control security policies. Tech. rep., SRI International (Dec 1992), http://www.csl.sri.com/papers/csl-92-2/
25. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21, 2003 (2003)
26. VMware: Providing LUN Security. Available at http://www.vmware.com/pdf/esx_lun_security.pdf (March 2006)
27. Wojtczuk, R.: Adventures with a certain Xen vulnerability (in the PVFB backend). http://invisiblethingslab.com/pub/xenfb-adventures-10.pdf (October 2008)
28. Wool, A.: Architecting the Lumeta Firewall Analyzer. In: SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium. pp. 7–7. USENIX Association, Berkeley, CA, USA (2001)
29. Xie, G., Zhan, J., Maltz, D., Zhang, H., Greenberg, A., Hjalmtysson, G., Rexford, J.: On static reachability analysis of IP networks. In: INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. vol. 3, pp. 2170 – 2183 vol. 3. IEEE (13-17 2005)
30. Yang, J., Twohey, P., Engler, D., Musuvathi, M.: Using model checking to find serious file system errors. ACM Trans. Comput. Syst. 24, 393–423 (November 2006), http://doi.acm.org/10.1145/1189256.1189259

Notes

[1]IBM, PowerVM and pSeries are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both. Java is a registered trademark of Oracle and/or its affiliates. Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries. Other product and service names might be trademarks of IBM or other companies.

A Information Flow Types

We borrow concepts from information flow analysis, where we focus on information flow as the deterministic propagation of discrete units of information throughout a system.

A.1 Flow Types

Information flow analysis of multi-tenant configurations in virtualized environments analyzes *overt* and *covert* channels.[5] An *overt channel* is intended for communication; a principal can read or write on that channel within the limits of some access control policy.

Lampson [15] introduced the term *covert channel* as a channel not intended for information transfer at all. Consider a malware in VM Alice which attempts to transfer information to another instance of the malware in VM Bob, both hosted on the same hypervisor. The malware on VM Alice can, for instance, monopolize a resource[6] to transmit a bit observed by the malware on VM Bob in performance or through-put decrease.

[5] This is similar to the analysis of explicit and implicit information flow on high and low variables [25].

[6] Examples include reserving a bus, launching expensive computations, flooding a cache, sending many network packets.

We perceive covert channels to be a common phenomenon in virtualized infrastructures. Requiring the absence of all covert channels from hypervisors, physical hosts and resources, will render many resulting system impractical. Therefore, we allow administrators to specify a certain amount of covert channel information flow as tolerable.

Requirement Definition We informally stated our security goal as *isolation* between zones, which sounds similar to *non-interference* [8,9]. This requirement enforces that actions in one zone do not have any effect on subsequent behavior or outputs in another zone.

The transitivity of non-interference renders it, however, unsuitable to model our setting, in which information flow via guardians may be permitted, whereas the corresponding direct flow is disallowed. Agreeing to the arguments of Rushby [24] and Mantel [16], we would need *intransitive non-interference* [10] to start with. Furthermore, the existing definitions are based on traces of steps and, thus, inherently dynamic[7], whereas we aim at a high-level static information flow analysis (its topology and communication links). Therefore, we preclude a step/trace-based non-interference analysis.

Another candidate is the analysis for *separation*, e.g. [23,12]: one removes all guardians from the system and verifies that the remaining parts are perfectly separated; however this approach was criticized by Jacob [11].

The concept of *channel control* [22] sounds interesting, as it captures our requirement to specify *exceptions* to the general zoning requirements. For instance, two zones should not communicate with each other *unless* a guardian mediates and filters the communication. In our case, however, we are not studying single channels, but a complex topology of channels.

B Security of Information Flow Analysis

B.1 Reduction to Traversal Rules

Theorem 2 (Reduction to Traversal Rules, Theorem 1). *The correct isolation modeling by traversal rules $f_{\mathbb{T},\mathbb{P}}$ implies absence of false negatives.*

We prove Theorem 1 by contradiction and induction over the length n back-trace graph traversal. The proof by itself is straight-forward as graph coloring (Def. 7) is a recursive definition.

Proof. Let sets of types $\mathbb{T}$ and properties $\mathbb{P}$, a valid graph (V, E) and information sources $\hat{V}$ and sinks $\check{V}$ be given.

Suppose a false negative event $N \in (\neg \mathsf{A} \wedge \mathsf{B}) \neq \emptyset$. By definition, there exists a breach, that is a sink $\check{v} \in \check{V}$ for which holds that it is colored by a source $\hat{v} \in \hat{V}, \hat{v} \neq \check{v}$.

Initialize a set $\boldsymbol{E} = \emptyset$.

Induction start $n = 1$**:** the sink $\check{v}$ is colored because of the breach event B.

Assume the induction statement true for $n - 1$**:** A colored vertice v_{n-1} could only have been colored if

(a) v_{n-1} is source $\hat{v}$ with the corresponding color (then we are done and output $\boldsymbol{E}$) or

[7] To that end, Haigh and Young [10] have shown that it is necessary to analyze the complete trace of actions subsequent to a given action to validate that the action is allowed to interfere with another zone.

(b) there exists an edge $e = (v_n, v_{n-1})$ with $v_n = (\cdot, t_n, p_n)$ and $v_{n-1} = (\cdot, t_{n-1}, p_{n-1})$ for which holds: v_n is colored and the traversal rules $f_{\mathbb{T},\mathbb{P}}(t_n, t_{n-1}, p_n, p_{n-1})$ evaluate to follow. Accumulate $\boldsymbol{E} := \boldsymbol{E} \cup \{e\}$.

If the induction succeeds, then $\boldsymbol{E}$ is a construction of a path between source $\hat{v}$ and sink $\check{v}$, thus an alarm event, A. We obtain a contradiction against $N \in \neg\mathsf{A}$.

Consequently, for any case in which no sink-source path can be constructed, there exists an edge e for which the traversal rules $f_{\mathbb{T},\mathbb{P}}$ evaluate to stop. This reduces the false negative to the correctness for the traversal rules.

B.2 Inspection of Individual Traversal Rules

First, let us analyze the rules for network switches and VLAN traffic. Rule 5 assume a correct implementation of an isolation by network switches for switched-off ports. Rules 6 and 7 establish the VLAN en- and decapsulation by network switches and are interesting for the security analysis. The rules assign a VLAN-specific color to information flow for in-ports with VLAN tagging and only allow information traversal at out-ports with matching VLAN tags. This models the VLANs' traffic separation by encryption lifted to VLAN tags as well as a cross-session key separation assumption, standard for secure channels: messages encrypted under one VLAN tag cannot interfere with messages encrypt under other VLAN tags and can only be decrypted under the same VLAN tag. We can derive these properties from research on secure channels and their parallel composition (in cryptography for instance Canetti and Krawczyk's work on UC secure channels [6]; in formal methods for instance Mödersheim and Viganò's formalization of secure pseudonymous channels [19].) Rule 9 stops information flow at ports with non-matching VLAN tags accordingly. Rule 8 has information flow follow through for trunked VLAN ports. Otherwise, we assume that the network and physical switches securely configure and implement VLAN traffic isolation for flows from switch to port (Rules 10 and 1). We conclude that these assumptions are natural and model correct network behavior.

Second, let us consider the stack of physical machine, hypervisor and VMs. Rules 2 and 3 make the assumptions that a management OS and physical host provide secure isolation and that all information flow is accounted for explicitly. These assumptions are necessary for virtualization security, as information leakage from these components can subvert the entire system's security, and model standard trust assumptions. Rule 4 is interesting as it assumes that hypervisors sufficiently separate VMs against each other, that is, that information flow through cross-VM covert channels can be neglected. Research results exist that highlight cross-VM covert channels, for instance [21,1]. Therefore, this trust assumption on the hypervisor's multi-tenancy capability must be subject to thorough debate.[8] Whereas the isolation assumptions on physical machine and management OS are natural and well founded, we conclude that the modeling of covert channels is a key trust decision for the hypervisor model.

Third, let us consider the information model for storage. Rule 11 models that the storage controllers are capable of separating information flow to physical disks, whereas Rule 12 establishes that the file system prevents cross file information flow through its access control enforcement. The former property is systematically enforced by virtualization vendors, such as VMware [26] that do not allow reconfiguration of storage back-ends for VMs. The latter property found attention in research and can be checked with tool support [30]. We conclude that both assumptions are natural to make.

[8] For high-security environments, we recommend to set this rule to follow and therefore only relying on physical separation, yet dismissing hypervisor multi-tenancy.