RZ 3786 (# Z1008-003) 08/26/2010
Computer Science 8 pages

Research Report

Automating Security Audits of Heterogeneous Virtual Infrastructures

S. Bleikertz*, T. Gross*, M. Schunter*, K. Eriksson‡

*IBM Research – Zurich

‡InfraSight Labs

Research
Almaden • Austin • Beijing • Delhi • Haifa • T.J. Watson • Tokyo • Zurich

Automating Security Audits of Heterogeneous Virtual Infrastructures

Sören Bleikertz Thomas Groß
Matthias Schunter
IBM Research - Zurich
{sbl,tgr,mts}@zurich.ibm.com

Konrad Eriksson
InfraSight Labs
konrad.eriksson@infrasightlabs.com

ABSTRACT

The use of server virtualization has steadily been growing — but many enterprises are still reluctant to migrate critical workloads to such private cloud infrastructures. One critical inhibitor is the complexity of correctly configuring virtualization technology to implement enterprise security policies, such as isolation of workloads or customers, across all potentially shared physical and virtual resources.

To mitigate these concerns, this article describes a virtual infrastructure validation and assurance system for a heterogeneous datacenter. It supports different types of server hardware and virtual machine monitors as well as both virtual and non-virtual resources, such as networking and storage.

We use customer isolation as a running example to illustrate our approach to validating configurations. We focus on virtual infrastructures and provide a detailed view on internal resource allocations and configurations. We demonstrate the utility of our framework through evaluation and visualization of information flow graphs that are used to validate customer isolation. Our system discovers the actual configuration of the virtualization infrastructure (Xen, VMware, KVM, and IBM's PowerVM) and unifies the configuration data into a joint generic data model. This data model is then used to derive a data flow graph that allows us to automatically determine whether two subscribers share any resources that are not trusted in order to achieve proper isolation.

1. INTRODUCTION

The use of server virtualization has been growing substantially. It enables better utilization of today's server hardware, faster deployment, and load balancing through migration of virtual machines. Virtualized infrastructures provide standardized computing, virtual networking, and virtual storage resources. They enable simple creation of new servers and load balancing through migration while increasing physical server utilization and decreasing power consumption. In addition, the corresponding business model of infrastructure clouds provides variable payment and seemingly unlimited scalability.

The rapid growth of IT infrastructures as well as the ease of machine creation enabling self-service portals has lead to substantial numbers of servers being created (the so-called "server sprawl"). This often leads to large and complex configurations that exceed the complexity that can be handled and validated by human administrators. As the resulting virtual infrastructure evolves in an ad-hoc manner, it commonly violates best practices and corporate policies. Furthermore, the introduced virtualization layer and extensive physical resource sharing may lead to new side-channel vulnerabilities.

This issue is partially resolved by automated management systems that constrain the users' actions. In reality, however, these systems can fail, their security policies can lack proper enforcement, or human intervention can lead to additional errors in such complex environments. For these reasons, virtual server technology is often constrained to uncritical workloads or configured to run a single workload type that does not require isolation. We believe that critical applications would benefit from trustworthy evidence that the resulting configuration complies with corporate policies and best practices.

One approach for generating such evidence is to allow external auditors to examine and validate the hidden implementation as well as the management systems and processes in use. While this approach allows verification of a fixed set-up, it does not guarantee security over time for a rapidly changing configuration.

In this article, we describe a novel approach for validating the correct configuration of virtual infrastructures. Our systems allows stakeholders to automatically examine the correctness of the current configuration of such an infrastructure cloud. We use the common requirement of isolation as a running example, that is, different and mutually untrusting customers must be properly isolated even if they share resources. This means that no information flow must be possible whenever two customers share a resource, such as a host and its memory, storage, or networks.

In principle, this required absence of any information flow precludes direct information flow as well as covert channels. While the absence of direct information flow can be enforced, there is a fundamental trade-off between the desirable sharing

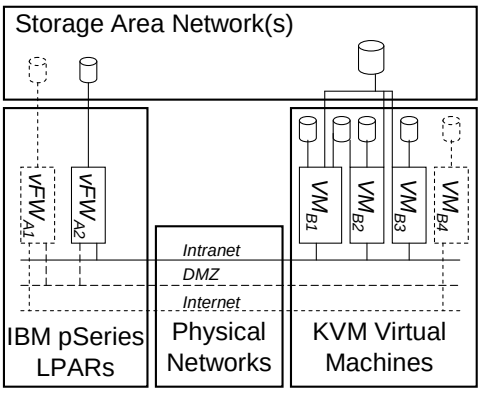

Figure 1: An example setup of a virtualized datacenter.

of resources and the absence of covert channels. To model this fundamental policy decision, we allow users to specify assumptions as to which resources are sufficiently isolating (i.e., that the capacity of the remaining covert channels is acceptable) and which resources produce unacceptable information flow if shared. This allows users to specify stronger isolation for critical workloads while enabling more sharing for uncritical workloads.

Contributions. We introduce a novel concept for automated security validation of virtualization configurations across multiple physical machines and potentially using different virtualization techniques. We contribute a graph abstraction and unification of virtualization configurations, that are used for the analysis of security requirements. We report on the implementation of a prototype, that demonstrates the feasibility and scalability of the particularly complex multi-tenant isolation requirement.

Outline. In Section 2, we introduce our high-level objectives as well as a high-level architecture view of a virtual datacenter to illustrate and motivate our approach. In Section 3, we outline existing related concepts. In Section 4, we start by defining the detailed security objectives and algorithms that underpin our multi-tenant isolation analysis. In Section 5, we generalize this and explain how our system can validate a wide range of security properties for cloud computing infrastructures. In Section 6, we then evaluate this prototype to demonstrate the feasibility and scalability of our approach. In Section 7, we conclude this article and outline future work.

2. OBJECTIVES

Our goal is the security validation of complex configurations of a virtualized datacenter. This datacenter includes different types of server hardware, implementations of virtual machine monitors, and physical and virtual networking and storage resources.

Figure 1 depicts a simplified version of such a configuration. This simplified version includes the following hardware: A IBM pSeries server, an x86-server, a physical networking infrastructure providing VLANs, and a Storage Area

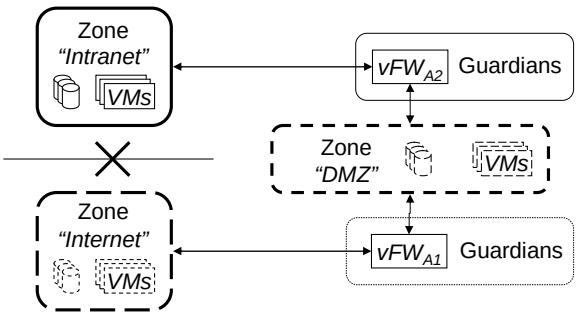

Figure 2: Example policy for isolating three virtual security zones.

Networking providing virtual storage volumes. The virtual resources (networks, storage, machines, and virtual firewalls) are depicted inside these hardware resources.

Our system aims at specification and validation of complex configuration properties of such virtual infrastructures. In this paper, we focus on validating isolation properties. For this example, Figure 2 depicts the desired isolation topology: Logically, the example virtual datacenter is supposed to isolate 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 firewall vFW_{A2}. Similarly, firewall vFW_{A1} moderates and restricts the communication between the Intranet and Internet zones, respectively. In this example, our goal is to validate that no unauthorized information flow is possible between these zones. From a configuration perspective, this means that there are no components that connect two zones or are shared by two zones while not being trusted to sufficiently mediate covert and overt information flow.

Note that we focus on validating the virtual infrastructure's configuration. Thus, once we have guaranteed that no undesired information flow exists except through the specified guardians, existing concepts for firewall validation need to be used to ensure that firewalls implementing the guardians have been correctly configured and do not permit undesired information flow between zones.

3. RELATED WORK

Virtual systems introduce several new security challenges [3]. Two important drivers mentioned that inspired our work is the increase of scale as well transient nature of configurations that render continuous validation more important.

The first area of related work is security of virtual machine monitors. Analysis of well-known attacks such as jailbreaks [22] allows detecting 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. Example include co-hosting validation [16] and cache-based side channels [15, 1]. A final type of virtualization vulnerabilities is caused by the potential to roll-back machines and their pseudo-number generators [17], which may result in predictable keys in cryptographic protocols.

A second area of related work is *reachability analysis* in networks and the related *configuration analysis of firewalls*, in particular the analysis of whole networks including packet filters, transformers, and routers, are [2, 9, 24]. *Firewall configuration analysis* allows understanding and validation of firewall rule [13, 14, 23]. 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 [10]. 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.

A third area of related work is the usage of *visualization* of security that has become increasingly popular in the recent years [5]. Visualization of firewall policies, reachability and attack graphs are published in [19, 20, 21]. Our approach to simplify complex multi-host configuration into a logical model and then visualize this model provides a visualization approach that is tailored to detecting isolation breaches in complex infrastructures.

4. ISOLATION VALIDATION

4.1 Information Flow

We borrow concepts from information flow analysis. Whereas formal information theory is a wide field, we focus on information flow as the determinate propagation of discrete units of information throughout a system. The concept of *channel control* is particularly interesting to us. Whereas the common intuition is that there should be no information flow between different zones, this requirement is too strong for our purposes. In particular, it must be possible 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. Before we shed light at possible requirement definitions, we discuss information flow types.

Flow Types. We cover *explicit* and *implicit* information flow (sometimes also called direct and indirect). Explicit information flow occurs when a variable is directly assigned or, in a system, if a read or write access occurs. For instance, if a VM Alice has read/write rights for a storage provider, this constitutes a possibility of explicit information flow. Implicit information flow occurs, when a variable is determined by the program control flow, for instance, by the result of a condition evaluation.

For multi-tenant configurations in virtualized environments, covert channels are an important case of indirect flow. Lampson [11] 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[1] to transmit a bit observed by the malware on VM Bob in performance or through-put decrease. Similar methods combined with external observation of an honest VM's performance can determine co-location [16].

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

Requirement Definition. We informally stated our security goal as *isolation* between zones, which sounds similar to the classical requirement of *non-interference* [4, 8]. It states that actions in one zone do not have any effect of subsequent behavior or outputs in another zone.

The standard non-interference definition is rather strict. In particular, its transitivity renders it unsuitable to model our setting where information flow via guardians may be permitted while the corresponding direct flow is disallowed. Agreeing to the arguments of Rushby [18] and Mantel [12], we need *intransitive non-interference* to start with. Furthermore, the existing definitions are based on traces of steps and, thus, inherently dynamic.[2]

As we start from a system's configuration snapshot and do not inspect the behavior of VMs, we need to concentrate of static information flow analysis of the system (its topology and communication links) and preclude a step/trace-based non-interference analysis. We therefore introduce a property we call *structural non-interference*:

Definition 1 (Structural Non-Interference). *A static system topology provides* structural non-interference *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.*

Even though this definition is very relaxed compared to classical non-interference, it allows us to derive meaningful statements on absence of information flow for configurations of virtualized infrastructures.

4.2 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. Thus, the formal model contains a set of typed vertices $V = \{v_i = (label_i, type_i)\}$ and a set of edges $E = \{(v_i, v_j)\}$. We represent complex structures of the virtualization infrastructure by sub-graphs of multiple vertices. For instance, we construct guardians such as firewalls with complex informa-

[2]To that end, Haigh and Young [6, 7] have shown that it is necessary to analyze the complete trace of actions subsequent of to a given action to validate that the action is allowed to interfere with another zone.

tion flow rules by a firewall vertex connected to multiple port vertices.

We annotate a subset of the nodes as information sources or information sinks. 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. We apply traversal rules according to the type declaration of the nodes.

Modeling Information Flow Assumptions. A *traversal rule* defines 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.

A set of traversal rules specifies 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. For instance, one set of traversal rules could assume absence of covert channels, another one include intra-VM host covert channels, one set of traversal rules may assume isolation by storage providers, another one information flow through a provider's storage devices. By making the traversal rules orthogonal, we allow virtualization configurations to be evaluated against many different traversal rules to observe information flow under different 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 and to vertices that have received information flow from a certain source by a traversal rule. The total information flow of a system is the transitive closure of the graph traversal governed by the traversal rules. 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.

Validating Isolation. We pursue our goal of non-interference between zones by the following high-level algorithm:

1. For each zone, specify (at least) one information source with the color.
2. For each zone, specify (at least) one information sink.
3. Derive the transitive closure over the traversal of information flow starting from each respective information source.
4. Obtain a non-interference property, if and only if a zone's information sinks only obtained the color of this very zone's information source.
5. Any information sink with an alien color highlights at least one isolation breach.

Note that this algorithm not only determines the existence of a breach but also the paths the information flow takes.

4.3 Determining Information Flow

Starting from this high-level algorithm, we provide three methods for determining information flow.

Color Collision. We first mark information sources for all zones. We mark all nodes of a zone as sink. After obtaining the transitive closure of the traversal rules of the graph, we test whether there exists a color collision in for any vertex of the graph. That is, when a vertex is colored by more than one color, then we infer that there is an information flow between source and sink of the involved colors and, thus, isolation breached. This method produces *false positives* as color collisions may occur at non-sink vertices.

Color Spill. We specify information sources for the zones and explicit sinks with a dedicated color. For instance, we want to make sure that a certain zone receives a specific target color. After the transitive closure of the traversal rules, we check whether any color "spilled" in an alien-color sink. If a sink of one color gets connected to a different color, then we have an isolation breach. You could imagine the dedicated color sinks as a honey pot, waiting for colors from other zones to spill over.

Transformation to Logical Model. We first mark information sources in all zones. For each source, we determine the transitive closure of the traversal rules and thereby obtain a colored sub-graph. This discovered sub-graph determines the actual zoning that is realized by the virtualization configuration under the traversal rules. The *logical model* is a set of graphs where each graph contains the machines, storage, and network corresponding to one of these actual zones. If the different sub-graphs corresponding to isolated customers are disjoint we have isolation, otherwise we have a breach.

The logical model hides all details such as low-level networking elements or inter-machine connections to allow a user to focus on and understand the potential information flows. This model is particularly useful for a manual analysis: we visualize the zones of the logical model together with the connected machines and provide a quick overview of the system's overall state.

5. ANALYSIS FLOW

We structure the analysis process in four phases, which also constitute the main modules of our framework: i. configuration discovery, ii. realization model, iii. graph traversal and coloring, and iv. diagnosis. We depict this flow in Figure 3 and show the examples corresponding to this flow in Appendix A and B.

5.1 Discovery

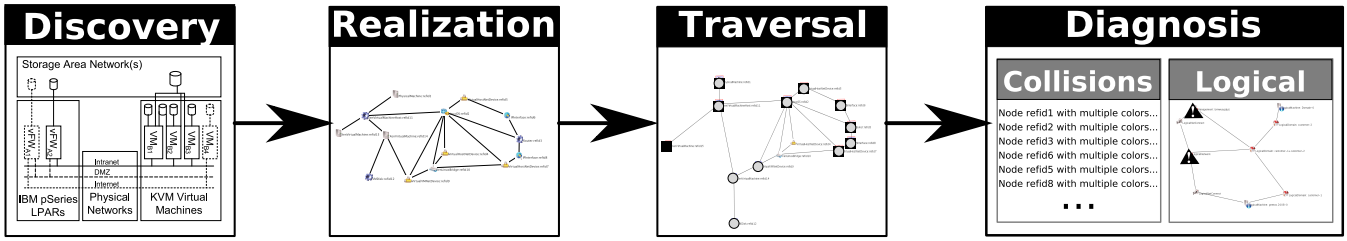

Figure 3: Overview over the analysis flow.

The discovery phase obtains a set of VM host addresses and associated API or login credentials as input and outputs a discovery XML file with outputs of all probes that represents the actual configuration.

We employ discovery modules to discover the configuration of the virtualization infrastructure. We realized multiple hypervisor-specific probes (Xen, VMware, IBM's PowerVM, or LibVirt).[3] The framework can be easily extended beyond the existing probes or use configuration data from a third-party source. We design the discovery to refrain from machine introspection as risk mitigation against data leakage.

5.2 Realization Model

The transformation into a realization model obtains a representation of the actual configuration as input and outputs a typed graph representation of the infrastructure that is unified for all hypervisor types.

We analyze the topology of the discovered configuration to derive a unified graph representation of the virtualization infrastructure, the *realization model*. The realization model expresses the low-level configuration of the various virtualization systems. It models the physical machine, virtual machine, storage, and network details as vertices. We exemplify the structure of the realization model in Appendix A. The physical machine part of the model, *e.g.*, represents the type of machine and its virtual machine monitor. The virtual machine part describes the virtual machines and their virtual resources. The storage realization model contains the virtual storage devices and their connection to physical storage. The network finally contains physical and virtual networking devices and their connections. We represent all connections between resources as edges in the graph, *e.g.*, if a VM is hosted by a hypervisor, we include an edge between VM and VM host.

We generate the realization model by a translation from the discovery data from the hypervisor-specific probes. Our architecture includes one adapter for each of the supported system types that is able to understand and translate the system-specific low-level configuration into the common realization model. Our tool stitches these fragments from different probes into a unified model that embodies the fabric of the entire virtualization infrastructure and configuration.

[3]While Xen, VMWare, and LibVirt-based systems provide an API for gathering configuration data, IBM pSystems are controlled by a centralized administration host from which the configuration of all managed pServers can be retrieved.

5.3 Graph Traversal

The graph traversal phase obtains a realization model, a set of traversal rules, and a set of sources and sinks as input. It outputs a colored version of the representation model encoding the potential information flows.

We traverse the realization model by applying a set of traversal rules and color the graph according to information flows. The traversal starts from the information sources and computes the transitive closure over the traversal rule application to the graph. The colored realization model can already be visualized for manual information flow analysis, as color paths and color collisions will be valuable indicators of isolation problems similar to canaries in a coal mine.

For example a node with a certain Property X should have the seed color blue and a node with Property Y always red. The traversal rules determine how the colors are propagated from these two seed nodes to the rest of the graph, *e.g.*, always retain the color when traversing from node with property X to a node with property Z.

5.4 Diagnosis

The diagnosis phase obtains a colored realization model and auxiliary data for the diagnosis modules as input and outputs a visualization of isolation breaches that is specific to the diagnoses module used.

Our framework enters a diagnosis phase, which is a refinement step to the colored realization model poised at producing meaningful outputs for admins. This phase allows multiple problem analysis modules, where we highlight two kinds of modules.

On the one hand, we have automated analysis modules, which determine collisions between colorings and output an information flow and isolation breach error log. This method produces still digestible outputs even for massive virtualization infrastructures (thousands of VM hosts/VMs).

On the other hand, we developed a module that transforms the realization model into the *logical model*, a set of sub-graphs, which encode the actual information flow zones of the infrastructures. This model provides a highly-simplified visualization for smaller systems (hundreds of VMs). The rule of the logical model is: whenever two resources have any information flow in the transitive closure of the traversal rules, they belong to the same sub-graph. Thus, a sub-graph in the logical displays all machines that can somehow "talk to each other" as a connected clique and all machines that are separated from them in a separate sub-graph. By that,

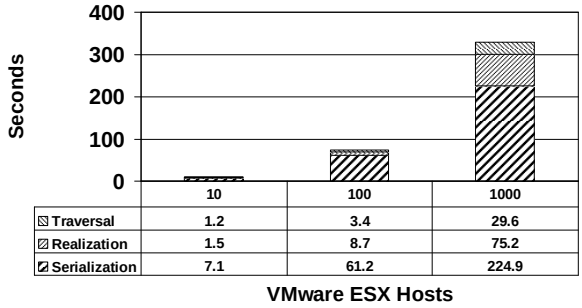

Figure 4: Performance of the three main stages of an isolation analysis.

admins can see a breach of isolation at one glance. We depict a simple case of a logical model with an isolation breach corresponding to the realization example of Appendix A in Appendix B.

6. EVALUATION & SCALABILITY

We prototyped and evaluated our concept using VMware ESX 4, IBM pSeries, and Xen virtual infrastructures. For VMware, the discovery of a single server with 8VMs required 20 Seconds. Since servers can be discovered independently, we expect that medium sized datacenters can also be discovered in this time. In the worst case, performance of discovery will be linear.

To assess the large-scale performance and scalability of our system, we emulated the configuration of 10, 100, 1000 VMware ESX4 hosts that each host 9 VMs. The corresponding configuration data was translated into our realization model using a ThinkPad T43p (Pentium M 750, 2GHz, 2GB RAM) running the Sun Java JDK 1.6.0.20. Figure 4 shows the resulting scalability evaluation of our non-optimized prototype that read raw discovery data from disk, translated and combined it into a realization model, and finally performed graph traversal and conflict detection.

We expected that discovery and translation into the realization model is linear in the number of VMs: Both only depend on the hosts and the elements within each host. Similarly, the graph traversal is linear in the number of nodes and the number of colors (each color can propagate to each node at most once). The observed overall speed is sufficient in practice and does not contradict that the expected performance is linear. One potential area for substantial improvement that we discovered is model serialization to XML and writing it on hard disk. Of the 300 sec spent to transform 1000 hosts from discovery data into the realization model, approximately 225 sec were spent on serializing and writing the resulting model to disk.

7. CONCLUSIONS & FUTURE WORK

In this article we demonstrated the ability to discover and efficiently validate the security of complex virtual infrastructures as well as infrastructure clouds. We have used the important and complex customer isolation requirement as our initial proof-point and demonstrated its feasibility by building and evaluating a demonstrator.

To make this a comprehensive solution for validating virtual infrastructures in practice, there are several open questions. The first open problem is a language for enabling customers to express their 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.

A second 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 when 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.

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. This project was partially supported by the MASTER research project funded by the European Commission's FP7 programme.

8. REFERENCES

[1] Aciiçmez, O. Yet another microarchitectural attack: exploiting i-cache. In *CSAW '07: Proceedings of the 2007 ACM workshop on Computer security architecture* (New York, NY, USA, 2007), ACM, pp. 11–18.

[2] Al-Shaer, E., Marrero, W., El-Atawy, A., and ElBadawi, K. Global Verification and Analysis of Network Access Control Configuration. Tech. rep., DePaul University, 2008.

[3] Garfinkel, T., and 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* (Berkeley, CA, USA, 2005), USENIX Association, pp. 20–20.

[4] Goguen, J. A., and Meseguer, J. Security policies and security models. In *IEEE Symposium on Security and Privacy* (1982), pp. 11–20.

[5] Goodall, J. R. Introduction to Visualization for Computer Security. In *VizSEC* (2007), pp. 1–17.

[6] Haigh, J. T., and Young, W. D. Extending the non-interference version of MLS for SAT. In *IEEE Symposium on Security and Privacy* (1986), pp. 60–60.

[7] Haigh, J. T., and Young, W. D. Extending the noninterference version of MLS for SAT. *IEEE Trans. Software Eng. 13*, 2 (1987), 141–150.

[8] III, J. W. G. Toward a mathematical foundation for information flow security. In *IEEE Symposium on Security and Privacy* (1991), pp. 21–35.

[9] Khakpour, A. R., and 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.

[10] Krothapalli, S. D., Sun, X., Sung, Y.-W. E., Yeo, S. A., and 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 (New York, NY, USA, 2009), ACM, pp. 63–70.

[11] Lampson, B. W. A note on the confinement problem. *Communications of the ACM 16*, 10 (1973), 613–615.

[12] Mantel, H. Information flow control and applications - bridging a gap. In *FME* (2001), J. N. Oliveira and P. Zave, Eds., vol. 2021 of *Lecture Notes in Computer Science*, Springer, pp. 153–172.

[13] Marmorstein, R., and Kearns, P. A Tool for Automated iptables Firewall Analysis. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), USENIX Association, pp. 44–44.

[14] Mayer, A., Wool, A., and Ziskind, E. Fang: A Firewall Analysis Engine. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2000), IEEE Computer Society, p. 177.

[15] Percival, C. Cache missing for fun and profit, May 2005.

[16] 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 *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security* (New York, NY, USA, 2009), ACM, pp. 199–212.

[17] Ristenpart, T., and Yilek, S. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *Proceedings of Network and Distributed Security Symposium – NDSS '10* (2010).

[18] Rushby, J. Noninterference, transitivity, and channel-control security policies. Tech. rep., SRI International, dec 1992.

[19] Tran, T., Al-Shaer, E., and Boutaba, R. PolicyVis: Firewall Security Policy Visualization and Inspection. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–16.

[20] Williams, L., Lippmann, R., and Ingols, K. An Interactive Attack Graph Cascade and Reachability Display. In *VizSEC* (2007), pp. 221–236.

[21] Williams, L., Lippmann, R., and Ingols, K. GARNET: A Graphical Attack Graph and Reachability Network Evaluation Tool. In *VizSec '08: Proceedings of the 5th international workshop on Visualization for Computer Security* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 44–59.

[22] Wojtczuk, R. Adventures with a certain Xen vulnerability (in the PVFB backend). `http://invisiblethingslab.com/pub/xenfb-adventures-10.pdf`, October 2008.

[23] Wool, A. Architecting the Lumeta Firewall Analyzer. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2001), USENIX Association, pp. 7–7.

[24] Xie, G., Zhan, J., Maltz, D., Zhang, H., Greenberg, A., Hjalmtysson, G., and Rexford, J. On static reachability analysis of ip networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE* (13-17 2005), vol. 3, pp. 2170 – 2183 vol. 3.

APPENDIX

A. EXAMPLE REALIZATION MODEL

We shed light at different aspects of our framework's realization model. The realization model is a unified abstraction over different virtualization configuration types. It incorporates an abstract data model for virtualization primitives, physical primitives, storage and network. We exemplify its structure with a class diagram of the networking abstraction in Figure 5.

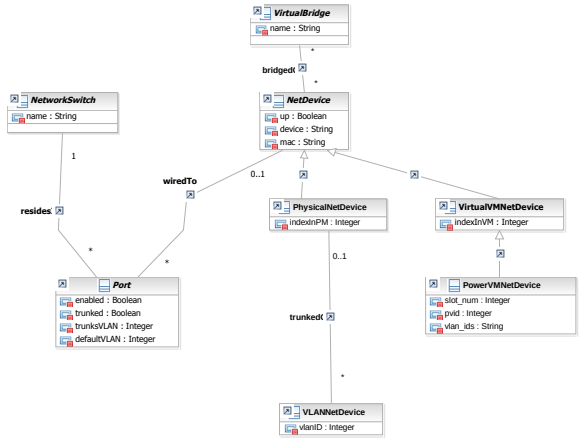

Figure 5: Class diagram of the realization model's networking abstraction.

The realization model constitutes a typed graph representation of the virtualization system, which is further annotated with the information flow coloring derived from the traversal rules. We depict such a colored version of a very simple realization model in Figure 6.

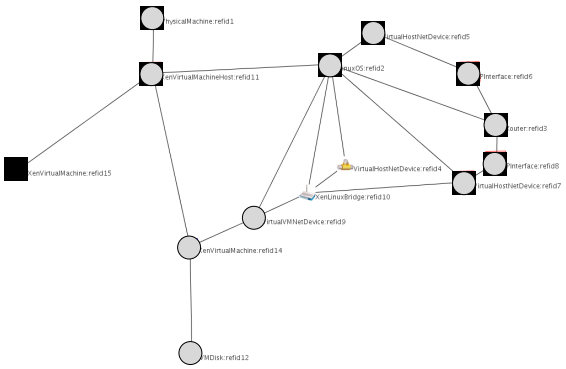

Figure 6: Example graph of a simple realization model with information flow coloring for two information sources.

B. EXAMPLE OF THE LOGICAL MODEL

The logical model shows the essence of an isolation analysis. It visualizes the actual zones of a configuration as either independent subgraphs (for correct isolation) or merged graphs with problem annotation (for isolation breach). We depict a logical model with breached isolation in Figure 7.

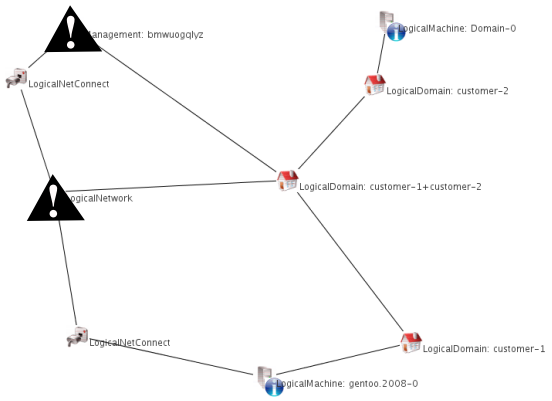

Figure 7: Example of a logical model with isolation breach between two zones.