Towards Non-Intrusive Software Introspection and Beyond

Apoorve Mohan*, Shripad Nadgowda‡, Bhautik Pipaliya*, Sona Varma*,
Sahil Suneja‡, Canturk Isci‡, Gene Cooperman*, Peter Desnoyers*, Orran Krieger†, Ata Turk§

*Northeastern University, †Boston University, ‡IBM T.J. Watson Research Center §State Street Corporation

***Abstract*—Continuous verification and security analysis of software systems are of paramount importance to many organizations. The state-of-the-art for such operations implements agent-based approaches to inspect the provisioned software stack for security and compliance issues. However, this approach, which runs agents on the systems being analyzed, is vulnerable to some attacks, can incur substantial performance impact, and can introduce significant complexity. In this paper, we present the design and prototype implementation of a general-purpose approach for Non-intrusive Software Introspection (NSI). By adhering to NSI, organizations hosting in the cloud can as well control the software introspection workflow with reduced trust in the provider. Experimental analysis of real-world applications demonstrates that NSI presents a lightweight and scalable approach, and has a negligible impact on the performance of applications running on the instance being introspected.**

***Index Terms*—Software Introspection, Storage Dissagregation, IaaS Cloud**

I. Introduction

Many organizations deploy security agents alongside the provisioned software stack to verify sanity, i.e., to ensure that the software stack is not compromised. These agents run just like any other application on the system while implementing their respective security functions. IaaS-cloud providers even offer deployable agent-based introspection solutions [1] and image templates already "baked" with such security agents [2] [3], which report to a provider-managed central monitoring service. These agents periodically scan the file systems, memory, and running processes of the systems within their scope in order to compare their findings against databases of known vulnerabilities[1] or malware, and send reports to the system administrator summarizing their findings.

Over the decades, these practices of implementing security functions through local agents have become a standard in data centers and clouds, but this approach has several shortcomings (see Fig. 1). *First*, security agents themselves may become vulnerable and lead to new attack vectors into one's system, as reported through a recent "DoubleAgent" attack that turns one's antivirus into malware or/and hijacks the system [6]. *Second*, as an agent is required to be installed separately on every system, the operation and maintenance of these agents becomes challenging at cloud-scale. For example, introspecting 10K bare-metal instances would require deploying and managing the same number of agents, and a virtualized

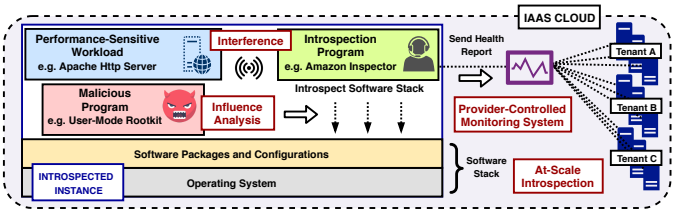


Fig. 1: Visualizing agent-based introspection.

environment with 64 virtual machines per host would require 640K such agents. *Third*, while performing periodic security inspection, these agents consume system resources (e.g., CPU cycles, memory, network resources, etc.), which may impact the performance of the primary workloads that the systems are catering to. While the system may tolerate or avoid these overheads, this may, in turn, lead to performance instabilities or resource inefficiencies [7]. *Finally*, a provider-managed introspection service is intrusive for privacy and security-sensitive organizations such as federal agencies, financial institutions, hospitals, etc. Despite the economic benefits of operating in a public cloud, they refrain from using public cloud offerings due to the lack of control and trust in the provider. For such organizations to use public cloud offerings, they would want to enjoy private datacenter-like properties (e.g., control, privacy, security, etc.), even when operating in a public IaaS-cloud [8].

Several efforts have been made to overcome the shortcomings of agent-based introspection [9] [10] [11] [12] [13] for specific instance types. But unfortunately, none of these offers a solution that is *non-intrusive, general-purpose*[2] *and tenant-controllable* at the same time. These efforts either intercept I/O requests of an instance provisioned to a remote virtual disk and recreate the filesystem state to perform introspection, or they propose provider-managed solutions to snapshot virtual machines at the host-layer and use the snapshot to perform introspection. For example, Banikazemi et al. [9] propose intrusion detection techniques for instances provisioned to a SAN target; however, this can lead to interference within the SAN controller's I/O path – this not only violates the requirements for non-intrusive introspection but also requires specialized

[1]Non-profit organizations (e.g., Mitre Corporation [4], National Institute of Standards and Technology [5], etc.) maintain and continuously update open-source databases of vulnerable software and software configurations for public use.

[2]In the context of this paper the phrase *"general-purpose"* refers to instance-type (i.e. virtual machine or bare-metal server) and operating-system agnostic. Furthermore, a general-purpose introspection system should not require specialized hardware (e.g., SAN) or support from the hypervisor (in case the of virtual machines).

hardware. In another proposal, Richter et al. [11] propose recreating the disk state of virtual instances by intercepting the I/O between the hypervisor and the disk emulator, which is not a general-purpose solution as it would not work for bare-metal instances. Oliveira et al. comes closest to achieving non-intrusive introspection by snapshotting running virtual machines and exposing the snapshots as read-only pseudo-devices for out-of-band introspection, however, not only is this approach instance-specific; it still requires resources and access to the host where the VMs are running. Furthermore, these systems require the tenant to trust the cloud provider fully.

In this paper, we explore the answers to the following questions:

- *Can we develop an agentless introspection technique that works for both virtual machines and bare-metal servers?*
- *Can we develop a system that has limited complexity for small-scale deployments, so it can be deployed in private clouds and modest clusters?*
- *Can we reduce the overhead and complexity of security introspection for cloud-scale systems?*
- *Can we support tenants that don't want to trust the provider (e.g., those that encrypt their storage)?*

Disaggregated storage allows us to address these questions. The most critical aspect of non-intrusive introspection is to have continuous and non-intrusive access to the state of the provisioned software stack. Disaggregating the persistent state of a running instance to a distributed storage system (e.g., Ceph [14], Lustre [15]) provides a clean separation of the provisioned software stack from the running instance. This disaggregation also enables a simple and practical mechanism for non-intrusive access to the state of the provisioned software stack, which can be exploited by different introspection and inspection APIs to overcome the shortcomings mentioned above for agent-based approaches. Until recently, storage disaggregation has been primarily limited to virtualized instances. However, recent research has demonstrated the possibility to use storage disaggregation for provisioning bare-metal instances with negligible performance overheads [16] [17] [18] [19] [20] [21] [22] [23].

In this paper, we present the design and prototype implementation of a general-purpose approach for Non-intrusive Software Introspection (NSI). NSI exploits the capability offered by distributed storage to create a cheap copy-on-write snapshot from a remote server and mounts the snapshot as a read-only volume on the remote server to introspect the latest state of the provisioned software stack. This approach has several advantages over previous ones. Since it uses a standard OS with support for arbitrary file systems, we can mount any volume, requiring none of the special purpose techniques developed in the hypervisor or SAN. Moreover, as the OS of the introspecting server is isolated (not exposed to the internet), it is simpler to ensure that the introspecting system is not compromised. Furthermore, the introspection is now decoupled from the tenant instance and is running on a remote server, thus avoiding performance impact on the workloads executing on the tenant's instance. NSI also enables tenants to create their own introspection servers to perform introspection on volumes encrypted by tenant-controlled keys, avoiding the need to trust to the provider; the only capability this requires is to ensure that the tenant has a way of mounting her own volumes.

Key contributions of this work include:

- We propose the design and prototype implementation of NSI, a general-purpose approach for non-intrusive software introspection. We also present a brief discussion on other security analytics use cases that can be aided by NSI. NSI consists of a set of microservices, reducing its deployment complexity and enabling it to scale-out or scale-up quickly. By opting for NSI's modular design, IaaS-providers can enable security-sensitive tenants to control and verify components required to introspect their provisioned software stack.
- A prototype implementation for NSI utilizing open-source components is provided[3]. NSI's modular microservice-based structure lends itself to replacing the underlying components – enabling system administrators to replace any underlying component to keep up with the changing technology. The prototype implementation is straightforward. We implemented introspection capability into an existing bare-metal provisioning system. All we needed was a way to snapshot and mount volumes. Although our prototype implementation is based on a bare-metal provisioning system, it works with virtual machines as it is.
- Our prototype implementation shows that the complexity of performing non-intrusive software introspection is small; i.e., all needed functionality can be contained in a simple set of services deployed in a VM. Deployment is straightforward for an enterprise and public clouds, and it can scale up or scale out quickly with small management overhead. Our evaluation shows that one can dedicate a modest amount of infrastructure to introspect on many computers; e.g., in our analysis, one system with 32 cores, 64 GB memory, and 10 Gbps NIC can perform basic software introspection every 5 minutes for up to 463 other computers. Based on the evaluation results, we deduce that for a 10K server cluster (i.e., a typical Borg cluster [24]), it would take 23 servers to perform basic software introspection, which is a trivial amount of infrastructure to set up and manage, rather than reserving resources for and coordinating with introspection agents running on all 10K servers.
- We also evaluate the performance impact of NSI's prototype implementation on real-world applications running on introspected instances and compare it with agent-based approaches. While agent-based introspection can lead to up to $\sim$12% performance degradation NSI is observed to be scalable with a negligible impact on the performance

[3] https://github.com/CCI-MOC/abmi

of the workloads. Our evaluation also demonstrates the possibility of reserving resources with high confidence for a particular introspection mechanism even across different applications.

The remainder of this paper is organized as follows. We present relevant background and the related work in Section II, discuss the NSI architecture and prototype implementation in Section III, present a runtime analysis for NSI and its performance impact in Section IV, and conclude in Section V.

II. Background and Motivation

In this section, we first list the requirements for employing a general-purpose introspection system and then present some existing work on introspection to explain/justify our need for a new non-intrusive introspection solution. We then discuss the advances in technology that enable us to perform agentless introspection in the cloud.

A. Requirement for Non-intrusive Software Introspection

We first review the key desired features of a non-intrusive software introspection, and how the requirements for supporting those features suggest the design of an agentless solution. In this section, we will briefly discuss those requirements:

a) Separation of Introspector and Introspected: Integrity of security introspectors is critical. To establish confidence in their security assessment (about the introspected system), security introspectors should be impervious to compromises. In the case of an agent-based system, the agents (i.e., the security introspectors) execute as just another application on the system, potentially alongside any malicious software and exposes themselves to compromise [6], doing more harm than good. Moreover, the agents could also transitively get influenced by the existing vulnerability in the system. For instance, some trojan malware could override the *ps* utility on your system to hide certain processes from reporting, and then any monitor using that utility to detect suspicious processes on your system will become ineffective [25]. Hence, it is important that introspectors are separate from the introspected system, i.e., the integrity of the entity performing the security introspection should be verifiable independently from the introspected system.

b) Managing Introspection at Cloud-Scale: It is essential to keep the operational and maintenance cost associated with security assessment in clouds to a minimum. In the existing agent-based approach, a separate instance of security software is required to be installed on every server. The complexity of managing them is directly proportional to the scale of introspected instances in the cloud. For example, introspecting 10K bare-metal instances would require to manage the same number of agents, and the management complexity would worsen even further if several virtual machines were running on each of the bare-metal instances. Common-routine administrative tasks such as monitoring the health of agents, collection of assessment reports, and rolling of agent updates to all servers become challenging in this approach as the scale grows. Furthermore, when operating at a cloud-scale, the networking between introspected systems and the reporting center becomes more complex as well. Therefore, there is a need for a solution wherein the instances in the cloud can scale independently, and the cost of implementing their security introspection can be contained and managed efficiently.

c) Non-intrusive Introspection: Periodic introspection should not have an impact on the performance of the workloads running on the introspected systems. Agent-based solutions require the co-location of the agent with the running workloads. When the agent periodically executes, it contends with the running workloads for resources, potentially causing jitter in the system and impacting the performance of the running workloads [26]. This impact can be mitigated by either (i) reserving exclusive resources for the agent on each server, or (ii) running introspection when the server is under-utilized. However, both of these approaches have side-effects, since either (i) the reserved exclusive resources for the agent will remain idle when the agent is not running (leading to resource inefficiency), or (ii) there may be large time windows between introspections, and malicious agents can exploit this feature to avoid introspection. It is preferable to employ an introspection mechanism that does not have a performance impact on the workloads running on the introspected systems.

B. Introspection Mechanisms

In this section, we present an overview of some of the well-known introspection mechanisms. They can be broadly classified into two categories:

a) Vulnerability Detection: Vulnerabilities are weaknesses that can be exploited by a malicious entity to perform unauthorized actions. Heartbleed (OpenSSL-based) [27], Shellshock/Bashdoor (Unix Bash shell-based) [28], and GHOST (Linux glibc-based) [29] are examples of the most infamous software vulnerabilities seen over the last decade. Various tools such as FlawFinder [30], RATS [31], ITS4 [32], and Foster [33] have been developed to detect software vulnerabilities. These tools employ techniques such as pattern matching, lexical analysis, data flow analysis, taint analysis, model checking, fault injection, fuzzing testing, etc., to detect vulnerabilities [34].

b) Malware Detection: Malware is an ill-intentioned software designed to conceal its identity and cause damage to the computer system that it runs on. Types of malware include Viruses [35], Rootkits [36], Keyloggers [37], etc. Tools such as `chkrootkit` (for detecting the presence of Rootkits) [38], Linux Malware Detect (Linux system scanner to detect threats in shared hosted environments) [39], and ClamAV (an antivirus engine for detecting trojans, viruses, etc.) [40] are examples of well-known malware detection tools. Such tools employ techniques like anomaly-based detection, specification-based detection, and signature-based detection to identify the existence of malware in a system [41].

Since there exist different tools and mechanisms to introspect for various software related security issues, we designed

NSI such that it is compatible with different existing introspection mechanisms.[4] NSI's default introspection mechanism performs vulnerability detection. When evaluating NSI, we demonstrate its compatibility with rootkit analysis and virus scan mechanisms. (See Section IV for details.)

C. Related Work

With the increase in the number, complexity, and code-base size of deployed software systems, the number of threats that can lead to security breaches increased as well [42], [43], [44]. Thus, introspection has become a key requirement for organizational IT deployments in cloud settings [1], [13], [45], [46], [47], [48], [49], [50], [51] and large-scale introspection systems have been developed to address these requirements.

Introspection systems can be classified into two types: Agent-based and Agentless introspection systems. In agent-based introspection systems, the introspection agent/software runs on each server and the agent periodically executes the desired introspection mechanisms (e.g., ClamAV [40], chkrootkit [38], Linux malware detect [39], etc.) on the server to forward the introspection results to a centralized statistics collection system. Amazon Inspector [1], IBM BigFix [52], Symantec Endpoint Protection [53], and Tanium Threat Response [54] are examples of agent-based introspection systems.

Agent-based introspection has a number of drawbacks such as being amenable to intrusion and creating interference, and these issues have been previously reported in connection with production deployments [26]. For virtual machines and containers, agentless introspection systems such as OpVis [13], Anchore [55], Clair [56], AquaSec [57], and Twistlock [58] have been proposed to overcome these drawbacks. These solutions exploit the non-intrusive capabilities available in virtualized/containerized systems that enable snapshotting of the target file-systems/images. Specifically, the open-source tool OpVis [13] snapshots virtualized instances at the host-level and introspects the snapshots using filesystem tree introspection techniques such as Columbus [59]. Anchore [55] and Clair [56] use image-scanning capabilities to introspect container images. AquaSec [57], and Twistlock [58] offer proprietary agentless container introspection solutions.

There exist a few studies that focus on agentless introspection of bare-metal servers that are provisioned to Storage Area Network (SAN) targets. Banikazemi et.al. [9] proposes intrusion detection techniques at the SAN target level. Unfortunately, the introspection they do at the SAN-level leads to interference within the SAN controller's I/O path [10]. IDStor [10] avoids this problem through a network-based intrusion detection approach. It intercepts every iSCSI request between the server and the SAN target, re-creates the filesystem state identifying an inverse mapping between blocks and the inodes of files (external to the SAN), and introspects the re-created filesystem state. However, it cannot detect threats

[4]NSI is compatible with introspection mechanisms for which access to the filesystem is sufficient to detect threats.

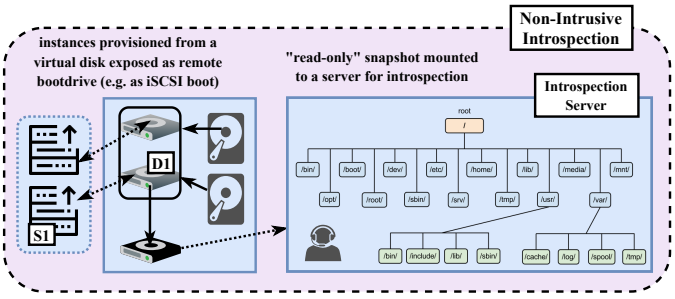


Fig. 2: Non-intrusive introspection of instance S1 provisioned to virtual disk D1.

caused by software that has not yet been accessed by the server. Moreover, IDStor approach is not sustainable as it requires developing special-purpose software to identify block-inode inverse mapping for every filesystem type. Unlike these approaches, the NSI design separates the image-control and introspection services and thus avoids interference. Furthermore, it can view the entire filesystem, and can detect vulnerabilities located in the unused parts of the image. Finally, NSI is not specific to SAN solutions, which require specialized hardware to operate.

D. Enabling Technologies

a) Data center resource disaggregation: The advances in data center networking capabilities (such as the common use of 10-40 Gbps NIC's on hosts, full bisection networks, improved Ethernet latency, redundant network switches, higher network bandwidth, link aggregation on bare-metal instances to handle failures [20], [60], etc.) have led to an ongoing paradigm shift towards data center resource disaggregation. By fabricating the same system resource-types on standalone blade servers that are interconnected via a network fabric, high-capacity, low-overhead disaggregated services can be offered [61]. Prominent examples includes, growing preference to use a distributed remote storage over local-disk solutions for storage for reasons of reliability, cost and scalability [21], [62], [63], [64], ability to improve aggregate resource usage [65], etc.

b) Unified system for instance provisioning and image management: So far, such disaggregations were primarily limited to virtualized systems and were not considered for bare-metal instances. However, solutions such as M2 [17], [66], Bolted [18], OpenStack-Ironic [67], etc., exploit the above-mentioned technological advancements and offer rapid bare-metal provisioning in a fashion similar to virtual-machine provisioning. The bare-metal instances are provisioned from remote disks stored on distributed storage. Due to the aforementioned technological advances, they deliver comparable performance to applications that run over locally mounted disks [17], [68]. Furthermore, they offer image management functionalities such as rapid snapshotting and cloning for the bare-metal instance images. The advent of these systems has exemplified the possibility of a *general-purpose* provisioning

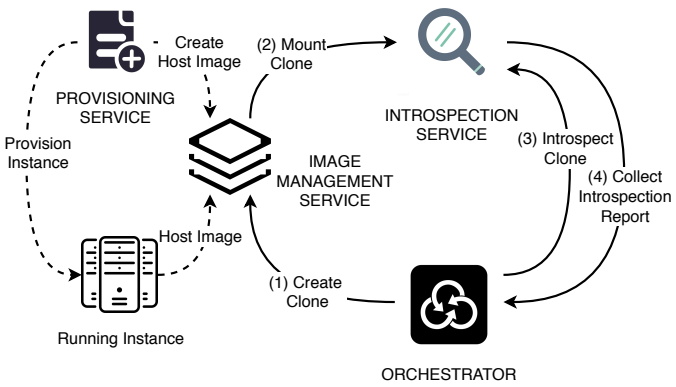


Fig. 3: NSI workflow design

and image management system for both virtual and bare-metal instances in the cloud.

III. NSI Overview

This section presents an overview of NSI's design, components, and workflow; and our prototype implementation. We also briefly discuss some of the other use cases that can be addressed using the proposed design.

A. Design Philosophy

The key goals of NSI are: (1) to enable non-intrusive software introspection in the cloud; (2) the introspection solution should be agnostic to cloud instance type (i.e., virtual machine or bare-metal); and (3) cloud providers should be able to support security-sensitive tenants. These goals have a number of implications in NSI's design. Fig. 2 present a high-level overview of of the proposed non-intrusive introspection.

NSI relies on diskless provisioning of cloud instances to access the state of the provisioned software stack. By doing so, NSI is not required to execute an agent on the introspected bare-metal instance or the host (in the case of virtual machines). Furthermore, using the feature-set offered by today's modern distributed storage systems, NSI can discreetly create lightweight read-only clones of the current software stack, which can be accessed remotely without interfering in the I/O path of the remotely provisioned instance. The read-only clone can be mounted remotely and be introspected non-intrusively – irrespective of the instance-type. Modern distributed storage implementations expose block device interfaces for operations such as lightweight cloning, remote mounting, etc. By provisioning instances in a diskless manner, tenants workloads are decoupled from the software stack, opening the possibility for discreetly snapshotting the current state of the software stack and performing out-of-band introspection on the latest snapshot. Thus, while performing periodic introspection, NSI avoids any performance impact on the workloads executing on an the introspected instance. NSI opts for a microservice-based architecture for software introspection to enable independent control and management of different components required for software introspection. Tenants can either rely on the provider to implement all these components or bring in their own implementation of a particular component. Tenants can also set up their own introspection servers to perform introspection on volumes encrypted by tenant-controlled keys, avoiding having to trust the provider; the only capability this requires is to ensure that the tenant has a way of mounting her own volumes.

B. Components and Workflow

NSI consists of four microservices. Fig. 3 presents the workflow design for NSI between the four microservices are: (a) Provisioning Service, (b) Image Management Service, (c) Introspection Service, and (d) Orchestration Service.

a) Provisioning Service: This service provisions cloud instances from the remote-mountable image disks made available by the Image Management Service. For newly provisioned instances, the service calls the Image Management Service to create a new host image, and then the cloud instance is provisioned from that host image. A server can be released by a tenant, and later the tenant can restart the server by pointing it to the same image. We call this re-provisioning. If the instance is being re-provisioned, the Provisioning Service uses the existing image hosted at the Image Management Service for re-provisioning.

b) Image Management Service: This service hosts the remote-mounted images for cloud instances and provides APIs to rapidly snapshot or clone a provisioned cloud instances' image. The remote provisioning service heavily relies on the image management service to perform its activities, but NSI mainly uses the cloning capabilities of this service to clone the host image of the instance to be introspected.

c) Introspection Service: This service first mounts a clone (lightweight snapshot) created by the Image Management Service for the instance to be introspected, mounts the clone as a as a standard filesystem, and then scans the mounted image to check for vulnerabilities. It maintains a database of known vulnerabilities. This database is populated and periodically updated by querying vulnerability databases (open- or close-source based on availability). The introspection service can run rootkit analysis and/or software vulnerability analysis over the mounted volume. For security-sensitive tenants, this service should also have a mechanism to manage secrets (encryption keys) to be used when mounting encrypted disks. We note that introspection operations that need memory analysis are not immediately supported with this design.

d) Orchestration Service: This service controls the remote introspection workflow among the different components. It also offers an interface for tenantss to introspect the instance they own. Tenants can submit requests to this interface for introspecting the instance they control. This allows them to control and change the introspection periodicity on demand, and hence it allows them to control the cost of introspection of their instance.

As shown in Fig. 3, when performing a remote introspection, the Orchestration Service: (1) creates a clone of the

provisioned instances' remote disk via the Image Management Service; then (2) mounts the filesystem present on the clone; and (3) invokes the Introspection Service to perform vulnerability analysis on the mounted image. Finally, (4) it collects the vulnerability analysis results and reports the data back to the tenant that initiated the introspection.

C. Extended Scope

NSI is an extensible system that can be used to support the enforcement of various Security and Compliance Industry Standards. This includes government regulatory standards like FedRAMP [69], NIST [5], and other industry standards like PCI-DSS [70], Center for Internet Security (CIS) [71]. These are IT/Cloud standards required across Financial, Payment Card industries. While we believe many of these requirements can be implemented on top of NSI, in the scope of the paper, we discuss *two* specific requirements that can be satisfied by our system.

a) Configuration Analytic: Software misconfiguration has been a major source of availability, performance, and security problems. In a virtualized and multi-tenant cloud environment, it is non-trivial to ensure correctness when configuring and cross-configuring such components. Moreover, configuration checks are also part of the different regulatory compliance obligations [71] [5] [69]. There are existing agent-based solutions [52] [72] to facilitate configuration management for VMs and bare-metal systems, and there are also agentless solutions [73] [74] [75] for containers. The unique storage dis-aggregation framework in NSI allows us to bring agentless configuration analytic capabilities for VMs and bare metals to scans application and system configuration settings to test them for compliance and best-practices adherence from a security perspective.

b) Integrity Assurance: Another critical security capability for the system is to identify the tampering of critical system management artifacts. These artifacts include configuration files, credentials, secrets-keys, or any other confidential data from your system. This is again regulated under the Chapter 11.5 of PCI-DSS standard. By periodically introspecting the system state and comparing it to the previous state(s), we can identify the file modifications in the system. These file modifications are further semantically analyzed to determine higher-level user action causing the change. For example, if we identify "/etc/passwd" and "/etc/shadow" files are modified, by comparing the content change, we can determine user add/remove action in the system. These changes are then compared against whitelisted actions to realize if unauthorized actions are triggered in the system. State-of-art integrity assurance solutions requires an active monitoring agents like $inotify$ or $auditd$ into system. Operating these agents in context has proven to be detrimental to the application performance. The agentless framework in NSI can be very efficiently leveraged to provide the integrity compliance assurance for VMs and bare metals.

D. Prototype Implementation

Since NSI follows a service-based approach, it allows administrators to replace the solutions used for any of the underlying services with the solutions that they prefer.

In our implementation, we employed M2 [17] as the Provisioning Service, and M2 in conjunction with Ceph [76] as the Image Management Service.

M2 is an open-source, multi-tenant, diskless provisioning service. It provisions instances to a remote disk residing on an image store backed up by a distributed file system, and uses the Hardware Isolation Layer [77] tool to isolate the provisioned servers. In our implementation, images of the instances provisioned by M2 reside in Ceph. M2 employs an iSCSI-based [78] network-booting [79] approach to provision instances. We used the Linux SCSI Target Framework (TGT) [80] for iSCSI-based network booting.

Ceph is an open-source storage platform that implements a highly reliable and scalable object storage on a distributed cluster. M2 uses Ceph's block storage interface for managing and snapshotting instance disk images.

For the Introspection Service, we extended M2 to support software introspection. This implementation maintains a database of software vulnerabilities populated using Canonical Ubuntu Security Notices [81][5]. Inherently, it uses IBM's open-source agentless crawler (IASC) [82] for scanning snapshots. IASC crawls through the filesystem tree present on a snapshot and generates frames corresponding to different OS and software package details. IASC stores the generated frames in a JavaScript Object Notation (JSON) [83] file. The vulnerability detection component of the Introspection Service then reads each frame and compares them against the blacklist present in the pre-populated database. After comparing each frame against the database, a list of vulnerabilities is generated and returned to the Orchestration Service. We are working on upstreaming the Introspection Service to M2. Note that our introspection implementation can be extended with other vulnerability detection services such as chkrootkit [38], OSCAP [84], or Linux Malware Detect (LMD) [39].

We also implemented our own Orchestration/Coordination Service, which coordinates among the services for introspection. It is implemented as a RESTful web-service [85]. Upon receiving an introspection request for a server, the Orchestration Service first creates a snapshot of the server's disk image, then maps the snapshot as a block device using Ceph's block device interface, followed by mounting the block device to a target directory. Now that the snapshot has been mounted for introspection, the Orchestration Service invokes the Introspection Service, passing the path to the target directory as an argument. After the Introspection Service returns the list of vulnerabilities, the Orchestration Service returns that list as the response to an introspection request.

[5]It is also possible to periodically query other open-source databases maintained by various non-profit organizations such as Mitre Corporation's Common Vulnerabilities and Exposures [4], National Institute of Standards and Technology's National Vulnerability Database [5], etc., and update the vulnerability database.

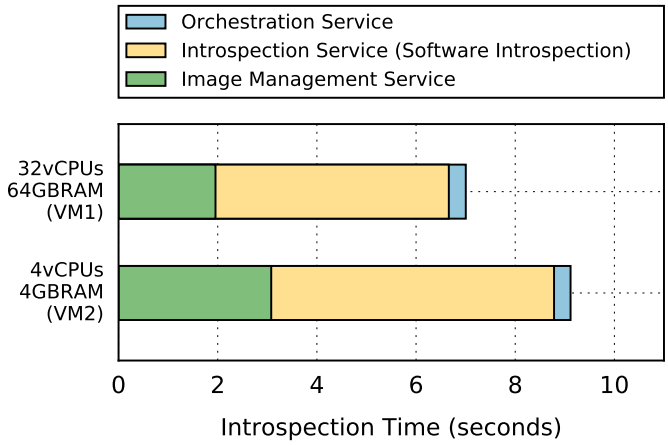


Fig. 4: Dissection of time taken to process a single Software Introspection request across different NSI services considering two different VM configurations to host NSI.

Before returning the list of vulnerabilities, the Orchestration Service also cleans up the state. i.e., it unmounts the mapped block device, unmaps the disk image snapshot, and deletes the snapshot.

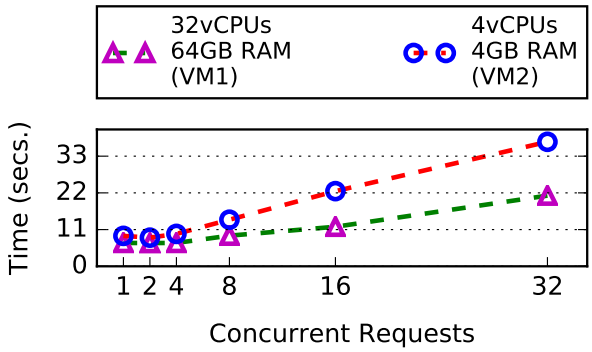
IV. Evaluation

In this section, we evaluate NSI. We first present the experimental setup. We then analyze NSI and its service's runtime under an increasing load. We then present the end-to-end introspection times of different vulnerability analysis mechanisms. Finally, we present the performance impact of introspection via NSI on workloads running on the introspected bare-metal servers.

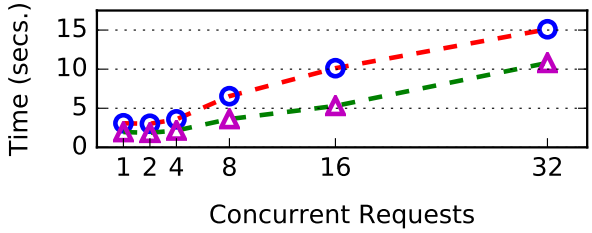
A. Experimental Setup

To evaluate NSI, on-demand bare-metal instances from the Massachusetts Open Cloud [86] were used. The bare-metal instances used during the evaluation have two 8-core Intel(R) Xeon(R) CPU E5-2650 v2 @2.60GHz (32 hyperthreaded cores), 64GB RAM and two dual-port 10 Gigabit Intel 82599 NIC's. All of the bare-metal servers to be introspected were running the RHEL 7.5 OS (provisioned from a 50GB base image). The servers had no local disks attached and the application data was stored on NFS drives mounted on the bare-metal servers which was not being introspected.

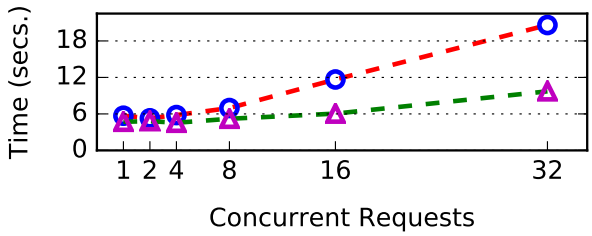
A three-node 98 TB Ceph storage cluster with 27 OSD's and 10 Gbit external and internal NIC's was used as the Image Management Service. Each experiment presented from here on is repeated five times, and the average of the five values is plotted. For all the experimental results presented in this section, none of the NSI components (responsible for introspection) were running on the introspected bare-metal instance. Therefore, the performances of applications running on the introspected bare-metal instances are not affected – irrespective of their compute intensities.



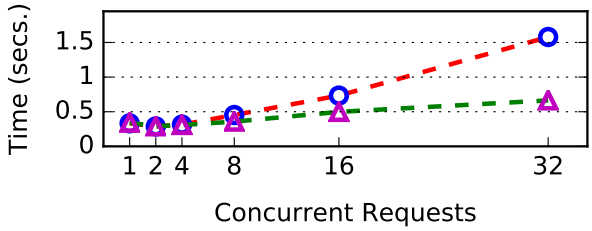
(a) NSI's runtime under increasing load.



(b) Runtime of Image Management Service under increasing load.



(c) Runtime of Introspection Service under increasing load.



(d) Runtime of Orchestration Service under increasing load.

Fig. 5: Runtime analysis of NSI and its services.

B. Runtime Analysis of NSI and its Components

In this section, we present an analysis of NSI and its components' runtimes. For these analyses, two different VM configurations are used to host the Provisioning Service, Image Management Service client, Introspection Service, and Orchestration Service. These VMs run CentOS 7.4. The first configuration (VM1) had 32 vCPUs and 64 GB RAM allocated, whereas the second configuration (VM2) had 4 vCPUs and 4 GB RAM allocated. No workloads were running on

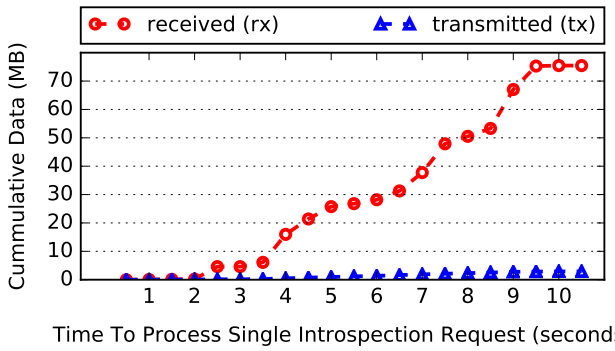


Fig. 6: Network traffic between Ceph Client and Ceph Cluster when processing single introspection request.

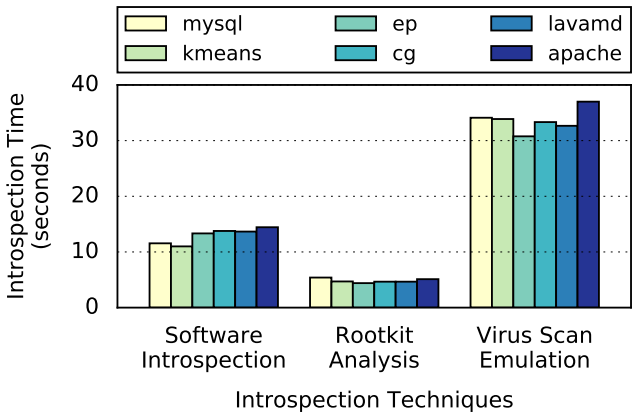


Fig. 7: Software Introspection, Rootkit Analysis, and Virus Scan times while running various workloads on the introspected server.

the bare-metal servers while they were being introspected. *Note that* the reason to conduct this experiment with two VM configurations is to provide insight to tenants operating at different scales. For example a tenant operating at a small scale should not worry about reserving many resources for introspection; whereas a tenant opreating at a larger scale should be able to estimate the the amount of resources required to introspect its massive infrastructure.

Fig. 4 presents a runtime dissection of the introspection of a single server across NSI services. As seen in the figure, in total, NSI requires from 7 to 9 seconds to process a single introspection request when deployed on VM1 and VM2, respectively. In both configurations, the introspection time is dominated by the Introspection and Image Management Service times, with the Orchestration Service taking the least amount of time.

Fig. 5 presents how the runtime of NSI and its services behave as the number of concurrent introspection requests issued to them is increased from 1 to 32. Fig. 5a shows the total introspection time of NSI under different numbers of concurrent introspection requests, whereas Figs. 5b, 5c, and 5d show the time consumed by the Image Management, Introspection, and Orchestration Services, respectively.

As seen in Fig. 5a, the time to process introspection requests is almost flat initially but starts to increase when trying to process more concurrent requests (8 in the case of VM1, and 4 in the case of VM2). As the number of concurrent requests increases, the runtime increase in the Image Management Service is more prominent. This increase is due to the increased network communication overhead between the Ceph client and the Ceph cluster. As seen in Fig. 5a, even for VM2 with 32 concurrent requests, the introspection time is less than 40 seconds, indicating that NSI can be deployed with modest resource requirements in production systems with a large number of servers that need to perform fast introspection. Also, the runtime differences between VM1 and VM2 indicate that NSI can benefit from vertical scaling – especially when the number of expected concurrent requests increase. Fig. 6 shows the cummulative network traffic between ceph client and ceph cluster when processing single introspection request. It was observed that while processing a single introspection request, ∼75 MBs and ∼3MBs data was received (rx) and transmitted (tx) respectively between the Ceph Client and Ceph Cluster.

Note that the communication between the Ceph client and the backend (i.e. the Image Management Service) causes the increase in end-to-end introspection time. The memory bandwidth is much higher than the network bandwidth and the blocks are fetched on-demand, thus memory bandwidth is not increasing the end-to-end introspection time. The Ceph client and backend are deployed with vanilla settings in our modest setup. Parameters for enabling read-ahead, minimum active clients, client-side caching, etc. were left unmodified/untouched. This can be one of the reasons why there was an increase of 5 seconds as the number of concurrent requests handled by the Image management Service increases from 16 to 32.

With the above results at hand, now let's consider an example for where we want to check for vulnerabilities in the kernel and installed software packages for a typical Borg cluster consisting of 10K bare-metal servers [24]. As shown in Fig. 5a, it takes ∼22 seconds to concurrently process 32 such requests with VM2, which indicates that a server with the same CPU-cores and memory can process ∼436 such requests every 5 minutes. Therefore, to introspect 10K servers every 5 minutes for basic vulnerabilities, 23 such servers should suffice. Furthermore, the network bandwidth usage to process each request (i.e., Fig. 6) shows that a 10 Gbps NIC should be sufficient for such an introspection server.

C. Different Introspection Mechanisms

Fig. 7 presents the Software Introspection, Rootkit Analysis, and Virus Scan introspection times with NSI while running various workloads on the introspected server. The basic Software Introspection scans for vulnerable OS and

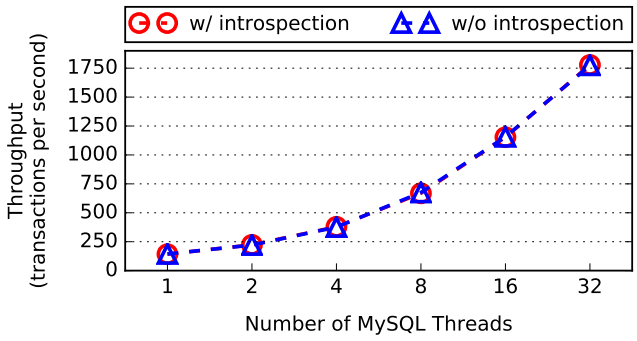


Fig. 8: Impact of periodic introspection on MySQL workload performance with increasing compute intensity.

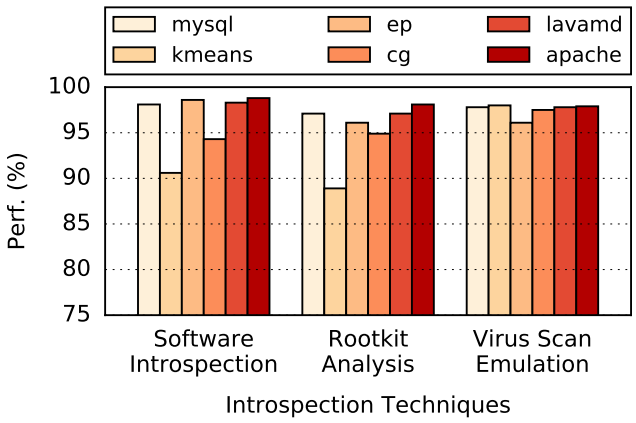


Fig. 9: Impact of agent-based introspection on performance of applications running on the bare-metal server being introspected.

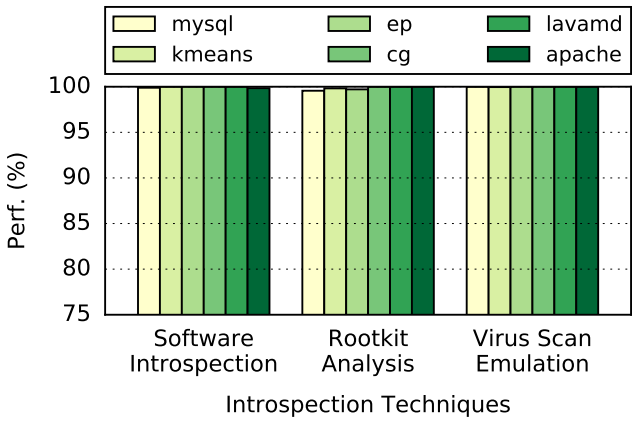


Fig. 10: Impact of agentless introspection on performance of applications running on the bare-metal server being introspected.

software packages on a server[6]. The Rootkit Analysis checks for the presence of rootkits using the `chkrootkit` [38] software tool. A rootkit is a software program that tries to gain unauthorized access to the system without being detected, The Virus Scan Emulation emulates the behavior of anti-virus software, i.e., to scan the contents of the entire filesystem. The content scan option of IBM's Agentless System Crawler [82] was used for this experiment.

In Fig. 7, the three introspection mechanisms were applied while the servers were running six different applications: the MySQL [87] database server; the K-means clustering and LavaMD n-body simulation applications from the Rodinia benchmark suite [88]; the Apache web server [89]; and the Embarrassingly Parallel (EP) and Conjugate Gradient (CG) applications from the NASA Advanced Supercomputing Parallel Benchmark (NPB) suite [90] (class C). In this experiment, the applications were configured to use all of the 32 hyperthreaded cores available on the bare-metal server. The Sysbench [91] and ApacheBench [92] benchmarking tools were used to generate workloads for the MySQL database server and Apache web server, respectively. Both benchmarks were running on the introspected bare-metal instances during the experiments. NSI was running on a VM2 configuration.

As expected, the time taken to introspect a server with different introspection mechanisms varied a lot, since the different introspection mechanisms were performing significantly different analyses on the filesystem. As seen in Fig. 7, the time taken to introspect a server was similar across applications. The slight introspection time variance observed was due to the differences in installed software and filesystem content across different applications. *Note that* as the perturbations across different applications are minor, we can estimate the resources we need for the introspection with high confidence; if it varied by a lot for different workloads, such estimate would have been difficult.

[6]NSI's Introspection Service (presented in Section III-D) was used for detecting vulnerable OS and software packages

D. Impact of NSI on Application Performance

We studied the impact of periodic agentless introspection on a single application when running with different application intensities and on different applications. In both cases, the server was introspected three times during the lifetime of each application and at an interval of one minute. Introspection of each application started after the bootstrap of that application, and application runtimes averaged around three minutes. VM2 configuration was used to host NSI services, and the applications and benchmarking tools used were the same as those presented in Section IV-C.

Fig. 8 presents the MySQL database server's OnLine Transaction Processing (OLTP) throughput (Transactions-Per-Second (TPS)) for two cases (with and without introspection) as the number of MySQL server threads were varied from 1 through 32. The first case (circles with dashed red line) presents the recorded TPS when the bare-metal server was

not introspected and the second case (triangles with dashed blue line) shows the TPS when the bare-metal server was periodically introspected. As seen in the figure, the two lines match perfectly and there is no visible degradation in the OLTP throughput due to introspection.

Figs. 9 and 10 respectively present the impact of agent-based and agentless introspection on application performance. For both figures, the reported performance percentages are normalized against the application performance observed when there is no introspection. For Fig. 9, introspection mechanisms presented in Section IV-C are installed as agents on the bare-metal servers and they are again configured to periodically introspect the servers three times during the lifetime of each application and at one-minute intervals.

As seen in Fig. 9, agent-based introspection causes up to 12% performance degradation. On the other hand, as seen in Fig. 10 agentless introspection has negligible impact on the application performance.

V. Conclusion

In this work, we proposed a general-purpose approach for non-intrusive software introspection, NSI, and presented its design and prototype implementation. NSI assumes that cloud instances are provisioned to remote virtual drives. By introspecting snapshots of these boot drives, NSI (i) mitigates the issue of trusting a potentially compromised agent during vulnerability analysis, (ii) eases the management of introspection by performing introspection through a centralized service instead of at each node, and (iii) provides a noninvasive introspection mechanism for bare-metal servers that does not impact their performance.

Our experimentation shows that NSI has negligible introspection overhead on the application performance, whereas agent-based systems can lead up to 12% performance degradation in application performance. Through NSI's prototype implementation and experimentation we demonstrate (a) how trivial it is to add non-intrusive introspection capability to an existing diskless provisioning infrastructure, (b) resources we need to dedicate for a general-purpose non-intrusive introspection are modest, (c) ease of scaling-up and scaling-out in enterprise and cloud-scale deployments, and (d) how tenant's can minimize trust in the provider for introspection.

VI. Acknowledgment

We gratefully acknowledge Mihir Borkar and Aditya Mohan Sharma for their contributions in development and documentation of NSI, and Naved Ansari and Radoslav Nikiforov Milanov for their assistance in reserving and setting up the evaluation environments. We would also like to thank the industry partners of the Mass Open Cloud (MOC), including Red Hat, Two Sigma, and Intel. Partial support for this work was provided by National Science Foundation Grant OAC-1740218.

References

[1] Amazon, "Amazon Inspector," https://aws.amazon.com/inspector/.

[2] A. E. C. Cloud, "Amazon web services," *Retrieved November*, vol. 9, p. 2011, 2011.

[3] D. Aguado, T. Andersen, A. Avetisyan, J. Budnik, M. Criveti, A. Doroiman, A. Hoppe, G. Menegaz, A. Morales, A. Moti *et al.*, *A practical approach to cloud IaaS with IBM SoftLayer: Presentations guide*. IBM Redbooks, 2016.

[4] "Common Vulnerability Exposures," https://cve.mitre.org/.

[5] "National Vulnerability Database," https://nvd.nist.gov/.

[6] "The clever 'DOUBLEAGENT' attack turns antivirus into malware," https://www.wired.com/2017/03/clever-doubleagent-attack-turns-antivirus-malware/.

[7] W. Yan and N. Ansari, "Why anti-virus products slow down your machine?" in *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*. IEEE, 2009, pp. 1–6.

[8] "Creating a Classified Processing Enclave in the Public Cloud |IARPA," https://www.iarpa.gov/index.php/working-with-iarpa/requests-for-information/creating-a-classified-processing-enclave-in-the-public-cloud, 2017.

[9] M. Banikazemi, D. Poff, and B. Abali, "Storage-based intrusion detection for storage area networks (sans)," in *Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on*. IEEE, 2005, pp. 118–127.

[10] M. Allalouf, M. Ben-Yehuda, J. Satran, and I. Segall, "Block storage listener for detecting file-level intrusions," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–12.

[11] W. Richter, "Agentless cloud-wide monitoring of virtual disk state," in *Proceedings of the 2014 workshop on PhD forum*. ACM, 2014, pp. 15–16.

[12] H. Zhou, H. Ba, J. Ren, Y. Wang, Y. Li, Y. Chen, and Z. Wang, "Agentless and uniform introspection for various security services in iaas cloud," in *Information Science and Control Engineering (ICISCE), 2017 4th International Conference on*. IEEE, 2017, pp. 140–144.

[13] F. Oliveira, S. Suneja, S. Nadgowda, P. Nagpurkar, and C. Isci, "Opvis: extensible, cross-platform operational visibility and analytics for cloud," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*. ACM, 2017, pp. 43–49.

[14] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation, (OSDI)*, USA, 2006, pp. 307–320.

[15] "About the lustre file system." [Online]. Available: http://lustre.org/about/

[16] A. Turk, R. S. Gudimetla, E. U. Kaynar, J. Hennessey, S. Tikale, P. Desnoyers, and O. Krieger, "An experiment on bare-metal bigdata provisioning," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. Denver, CO: USENIX Association, Jun. 2016. [Online]. Available: https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/turk

[17] A. Mohan, A. Turk, R. S. Gudimetla, S. Tikale, J. Hennesey, U. Kaynar, G. Cooperman, P. Desnoyers, and O. Krieger, "M2: Malleable metal as a service," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, April 2018, pp. 61–71.

[18] A. Mosayyebzadeh, A. Mohan, S. Tikale, M. Abdi, N. Schear, T. Hudson, C. Munson, L. Rudolph, G. Cooperman, P. Desnoyers, and O. Krieger, "Supporting security sensitive tenants in a bare-metal cloud," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 587–602. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/mosayyebzadeh

[19] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina *et al.*, "Rack-scale disaggregated cloud data centers: The dredbox project vision," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 690–695.

[20] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 249–264. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[21] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown, A. Klimovic, A. Slowey, and A. Rowstron, "Understanding rack-scale disaggregated storage," in *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*. Santa Clara, CA: USENIX Association, 2017. [Online]. Available: https://www.usenix.org/conference/hotstorage17/program/presentation/legtchenko

[22] A. Klimovic, C. Kozyrakis, E. Thereska, B. John, and S. Kumar, "Flash storage disaggregation," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: ACM, 2016, pp. 29:1–29:15. [Online]. Available: http://doi.acm.org/10.1145/2901318.2901337

[23] H. M. M. Ali, A. Q. Lawey, T. E. El-Gorashi, and J. M. Elmirghani, "Energy efficient disaggregated servers for future data centers," in *2015 20th European Conference on Networks and Optical Communications-(NOC)*. IEEE, 2015, pp. 1–6.

[24] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[25] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Vmm-based hidden process detection and identification using lycosid," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2008, pp. 91–100.

[26] "Amazon. summary of the october 22,2012 aws service event in the us-east region." https://aws.amazon.com/message/680342.

[27] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey *et al.*, "The matter of heartbleed," in *Proceedings of the 2014 conference on internet measurement conference*. ACM, 2014, pp. 475–488.

[28] T. Fox-Brewster, "What is the shellshock bug? is it worse than heartbleed," *The Guardian*, 2014.

[29] R. H. Inc., "GHOST: glibc vulnerability (CVE-2015-0235)," https://access.redhat.com/articles/1332213.

[30] D. A. Wheeler, "Flawfinder," 2011.

[31] C. C. Security, "Rough Auditing Tool for Security," https://security.web.cern.ch/security/recommendations/en/codetools/rats.shtml.

[32] J. Viega, J.-T. Bloch, Y. Kohno, and G. McGraw, "Its4: A static vulnerability scanner for c and c++ code," in *Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference*. IEEE, 2000, pp. 257–267.

[33] J. S. Foster and A. S. Aiken, "Type qualifiers: lightweight specifications to improve software quality," Ph.D. dissertation, Citeseer, 2002.

[34] W. Jimenez, A. Mammar, and A. Cavalli, "Software vulnerabilities, prevention and detection methods: A review1," *Security in Model-Driven Architecture*, p. 6, 2009.

[35] P. Szor, *The art of computer virus research and defense*. Pearson Education, 2005.

[36] M. Davis, S. Bodmer, and A. LeMasters, *Hacking Exposed Malware and Rootkits*. McGraw-Hill, Inc., 2009.

[37] A. Emigh, "The crimeware landscape: Malware, phishing, identity theft and beyond," *Journal of Digital Forensic Practice*, vol. 1, no. 3, pp. 245–260, 2006.

[38] "checkrootkit," http://www.chkrootkit.org/.

[39] "Linux Malware Detect," https://www.rfxn.com/projects/linux-malware-detect/.

[40] "Clam AntiVirus," https://www.clamav.net.

[41] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, vol. 48, 2007.

[42] Y. Shin and L. Williams, "Is complexity really the enemy of software security?" in *Proceedings of the 4th ACM workshop on Quality of protection*. ACM, 2008, pp. 47–50.

[43] T. Mccabe, "More complex= less secure," *McCabe Software, Inc*, p. 12, 2014.

[44] D. Williams, R. Koller, and B. Lum, "Say goodbye to virtualization for a safer cloud," in *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. Boston, MA: USENIX Association, 2018. [Online]. Available: https://www.usenix.org/conference/hotcloud18/presentation/williams

[45] R. Koller, C. Isci, S. Suneja, and E. De Lara, "Unified monitoring and analytics in the cloud," in *7th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[46] H. wook Baek, A. Srivastava, and J. Van der Merwe, "Cloudvmi: Virtual machine introspection as a cloud service," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 153–158.

[47] F. Yao and R. H. Campbell, "Cryptvmi: Encrypted virtual machine introspection in the cloud," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 977–978.

[48] L. Jia, M. Zhu, and B. Tu, "T-vmi: Trusted virtual machine introspection in cloud environments," in *Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on*. IEEE, 2017, pp. 478–487.

[49] S. Nadgowda, C. Isci, and M. Bal, "DÉjÀvu: Bringing black-box security analytics to cloud," in *Proceedings of the 19th International Middleware Conference Industry*, ser. Middleware '18. New York, NY, USA: ACM, 2018, pp. 17–24. [Online]. Available: http://doi.acm.org/10.1145/3284028.3284031

[50] S. Nadgowda and C. Isci, "Drishti: Disaggregated and interoperable security analytics framework for cloud," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '18. New York, NY, USA: ACM, 2018, pp. 528–528. [Online]. Available: http://doi.acm.org/10.1145/3267809.3275470

[51] S. Suneja, R. Koller, C. Isci, E. de Lara, A. Hashemi, A. Bhattacharyya, and C. Amza, "Safe inspection of live virtual machines," in *ACM SIGPLAN Notices*, vol. 52, no. 7. ACM, 2017, pp. 97–111.

[52] IBM, "IBM bigFix: A collaborative endpoint management and security platform," .

[53] S. Corp., "Symantec Endpoint Protection," https://www.symantec.com/smb/endpoint-protection.

[54] Tanium, "Platform for endpoint management and security."

[55] "Open source tools for container security and compliance." https://anchore.com.

[56] "Automatic container vulnerability and security scanning for appc and docker," https://coreos.com/clair/docs/latest/.

[57] "Container security - docker, kubernetes, openshift, mesos." https://www.aquasec.com/.

[58] "Docker security and container security platform." https://twistlock.com.

[59] S. Nadgowda, S. Duri, C. Isci, and V. Mann, "Columbus: Filesystem tree introspection for software discovery," in *Cloud Engineering (IC2E), 2017 IEEE International Conference on*. IEEE, 2017, pp. 67–74.

[60] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 350–361.

[61] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: scalability and commoditization," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. ACM, 2008, pp. 57–62.

[62] A. Klimovic, C. Kozyrakis, E. Thereska, B. John, and S. Kumar, "Flash storage disaggregation," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 29.

[63] E. K. Lee and C. A. Thekkath, "Petal: Distributed virtual disks," in *ACM SIGPLAN Notices*, vol. 31, no. 9. ACM, 1996, pp. 84–92.

[64] A. Warfield, R. Ross, K. Fraser, C. Limpach, and S. Hand, "Parallax: Managing storage for a million machines." in *HotOS*, 2005.

[65] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed OS for hardware resource disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 69–87. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/shan

[66] A. Turk, R. S. Gudimetla, E. U. Kaynar, J. Hennessey, S. Tikale, P. Desnoyers, and O. Krieger, "An Experiment on Bare-Metal BigData Provisioning," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, 2016.

[67] "OpenStack Bare Metal Provisioning Program," https://wiki.openstack.org/wiki/Ironic.

[68] D. Clerc, L. Garcés-Erice, and S. Rooney, "Os streaming deployment," in *Performance Computing and Communications Conference (IPCCC), 2010 IEEE 29th International*. IEEE, 2010, pp. 169–179.

[69] "Federal risk and authorization management program." [Online]. Available: https://www.fedramp.gov

[70] "Payment card industry security standards council." [Online]. Available: https://www.pcisecuritystandards.org

[71] "Center for internet security." [Online]. Available: https://www.cisecurity.org

[72] "Hcl appscan," https://www.hcltechsw.com/wps/portal/products/appscan.

[73] O. Tuncer, N. Bila, S. Duri, C. Isci, and A. K. Coskun, "Confex: Towards automating software configuration analytics in the cloud," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018, pp. 30–33.
[74] S. Baset, S. Suneja, N. Bila, O. Tuncer, and C. Isci, "Usable declarative configuration specification and validation for applications, systems, and cloud," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*. ACM, 2017, pp. 29–35.
[75] T. Chiba, R. Nakazawa, H. Horii, S. Suneja, and S. Seelam, "Confadvisor: A performance-centric configuration tuning framework for containers on kubernetes," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 168–178.
[76] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.
[77] J. Hennessey, S. Tikale, A. Turk, E. U. Kaynar, C. Hill, P. Desnoyers, and O. Krieger, "Hil: designing an exokernel for the data center," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 2016, pp. 155–168.
[78] K. Z. Meth and J. Satran, "Design of the iscsi protocol," in *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*. IEEE, 2003, pp. 116–122.
[79] H. P. Anvin and M. Connor, "X86 network booting: Integrating gpxe and pxelinux," in *Linux Symposium*. Citeseer, 2008, p. 9.
[80] T. Fujita and M. Christie, "tgt: Framework for storage target drivers," in *Proceedings of the Linux Symposium*, vol. 1. Citeseer, 2006, pp. 303–312.
[81] "Ubuntu Security Notices," https://usn.ubuntu.com/.
[82] "Agentless System Crawler," https://github.com/cloudviz/agentless-system-crawler.
[83] T. Bray, "The javascript object notation (json) data interchange format," Tech. Rep., 2017.
[84] "Open Security Content Automation Protocol," https://www.open-scap.org/tools/openscap-base/.
[85] R. Battle and E. Benson, "Bridging the semantic web and web 2.0 with representational state transfer (rest)," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 61–69, 2008.
[86] "Mass Open Cloud," https://massopen.cloud/.
[87] "mysql relational database managament system," https://www.mysql.com/.
[88] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 2009, pp. 44–54.
[89] "Apache HTTP Server Project," https://httpd.apache.org/.
[90] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The nas parallel benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.
[91] A. Kopytov, "Sysbench: a system performance benchmark," *URL: http://sysbench. sourceforge. net*, 2004.
[92] "ab - Apache HTTP server benchmarking tool," https://httpd.apache.org/docs/2.4/programs/ab.html.