Chapter One

Introduction

The advent of cloud computing coupled with the increasing trend of accessing storage over the network provides us with an opportunity to rethink how we monitor persistent storage. Traditionally, storage devices were tightly coupled with the operating system. The operating system acted as a facilitator of interaction with hardware for userspace applications. This was done primarily for protection and performance. The kernel interacted with storage devices via a driver and enforced constraints which userspace had to obey. This enabled the creation of crash-consistent file systems and robust storage of data culminating with advanced file systems like ZFS [133]. Kernel mediated direct memory access ensured fast reads and writes to persistent storage. Though the storage landscape is entirely different today, we still work with monolithic systems which believe they are tightly coupled to their underlying storage. Especially in virtualized clouds, the virtual machine abstraction, by design, prevents deep understanding of where resources come from.

But this tight coupling is an illusion. Within modern clouds there is either a layer of virtualization between a guest kernel and its storage, or the storage is served over the network. In many cases, both a layer of virtualization *and* a network hop exist between an executing guest and its storage. This reality pervades the enterprise in the form of network attached storage arrays. In the home, media servers are being used to save and serve large files such as movies via network file systems. Thus, it is no longer true that protection and performance are guaranteed by a monolithic kernel. The new reality decouples storage from its executing environment. However, higher performance, better resilience, and stronger protection are achievable if we rethink storage with a modern lens.

Higher performance and better resilience to data loss induced failures are out of the scope of this dissertation. But it is trivial to imagine why they are achievable with modern cloud computing. Clouds horizontally scale services such as storage into large distributed systems. This means individual cloud instances writing into cloud storage can obtain bandwidth far exceeding the bandwidth of a single drive, or even that of a RAID array. By quickly replicating data across cloud datacenters, even large-scale disasters become possible to survive—impossible with the more monolithic, tightly coupled version of storage. Stronger protection is within scope, and the basis upon which a lot of this dissertation was completed. The decoupling of layers in the storage realm enables separation

of mechanism, policy, and enforcement. This reflects lessons learned very early on in the networking and OS communities, but never truly applied to storage which traditionally remained tightly coupled from storage device to kernel driver and file system.

Early on in networking the value of separating mechanism and policy, along with a need for gatekeeping network packets, was clearly recognized. Mogul et al. describe implementing user-level packet filtering and inspection [79, 80]. Their design separated the mechanism—kernel-level hooks—of capturing packets, and the implementation—unprivileged, userspace applications—of policy. The importance of separating policy and mechanism was first described in the context of the Hydra research project by Levin et al. [63]. This early work led to the development of deep packet inspecting (DPI) firewalls in networking. DPI firewalls have three attractive properties. First, they are decoupled from the hosts that they monitor except for a network connection. Thus, they have the luxury of being immune to individual host misconfiguration or compromise. In other words, the firewall gatekeepers continue operating even with multiple misbehaving network hosts. In addition, they can not be turned off by the hosts, and the hosts can not affect the policy specified to the firewall. Importantly, these properties hold for both hosts within the firewall's protection boundary, and also for external hosts.

This dissertation explores adding functionality similar to deep packet inspection into the storage layer of cloud infrastructure. Although we use the word cloud, this rethinking of monitoring storage applies in general to storage whether or not it is virtualized. Access over the network technically makes this task easier to accomplish. In this dissertation, much like network-based firewalls, we propose implementing deep file-level monitoring via interpretation of storage writes. Whether in the absence of, or in addition to, a network hop, hypervisors act as a mediator between guest and storage in clouds. This unique position makes them the best location to implement our mechanism.

By placing file-level monitoring within a layer between kernel and storage we gain three properties which are impossible without such decoupling. First, the monitoring becomes immune to guest misconfiguration or compromise. Should a virus infect a guest, it can not hide its tracks if it touches persistent storage. When file-level monitoring is coupled into a monolithic system, viruses can hide. Rootkits which compromise the kernel have the capability of masking their presence entirely. Should an operator misconfigure a guest, file-level monitoring can fail. For example, the wrong permissions set on a log folder causes log analytics processes to fail and not perform their job. Second, guests become protected from bugs affecting file-level monitoring agents, and malicious monitors. No software is bug free, thus we must plan for bugs within file-level monitors—even those designed to protect such as virus scanners [83, 84]. The effect of such bugs is devastating because virus scanners often have root access to perform their job of finding viruses. Should a file-level monitoring process actually have malicious designs on the host it monitors, the decoupling of policy—what a monitoring agent has access to at a file-level—and mechanism—the method of capturing file-level state—provides protection. In a monolithic system, one has no choice but to run monitoring agents at the privilege level they request with direct access to resources. Third, file-level monitoring implementations become easier to scale across different operating systems and applications. They are easier to implement because they do not have to operate within the

same environment as the monolithic system producing the writes. The file-level monitoring occurs decoupled from the environment generating the writes.

Thus, we believe the time to rethink storage is now upon us. We also believe that the strong guarantees along with the easier implementation of file-level monitoring make decoupling policy and mechanism paramount for scaling management of storage in the future. Monitoring subsumes a large portion of the required upkeep of individual systems, and is a key problem facing administrators of any computing system [69]. Examples of monitoring include routine virus scanning [21], intrusion detection [118], log file analysis [112], and configuration auditing [119]. In the monolithic model, state-of-the-art monitoring is accomplished with generally third-party agents [51, 60] embedded within the system they monitor.

Embedding a potentially untrusted, third-party agent within the boundary of an enterprise increases its attack surface—the agent may be hijacked by an intruder and used maliciously [83, 84]. In addition, agents consume valuable resources, and are unpredictable in their resource consumption [60]. An agent using excessive memory causes memory pressure and potentially paging to disk, slowing down or even halting critical services. Finally, if a system is compromised, the agent may be tampered with or deliberately fed false information. In the case of a compromise or misconfiguration that is initially undetected, agents can not be trusted at all.

1.1 Problem Statement and Scope

This dissertation rethinks monitoring persistent state in the modern era of virtualized cloud computing, and network attached storage. Based off of the observation that storage is accessed either via a hypervisor or the network, this dissertation proposes deep inspection of disk writes for monitoring purposes. Similar to deep packet inspecting firewalls, this dissertation proposes decoupling mechanism and policy in the storage context. It seeks to answer the primary question of *how do we securely monitor file-level state as generally as possible*? In this context, security means guaranteed enforcement of policy both ways between a monitored system and the system performing monitoring. In other words, monitored systems should never fear the monitoring agents, and monitoring agents should never fear the systems they monitor. Generalizable in this context means generalizing across the combinatorial space of operating system types and applications.

This dissertation describes a mechanism and interfaces on top of that mechanism which enables separation of capturing state and the logic for monitoring that state. Thus, monitoring both online and offline persistent state is within the scope of this dissertation. Enforcement of policies is assumed to be a solved problem within cloud computing. Network quarantining and virtual machine suspension are both pre-existing techniques for isolating and stopping misbehaving cloud instances. Thus, enforcement is outside of the scope of this dissertation.

Online persistent state refers to live file systems as they mutate in real-time within virtual disks. Offline persistent state refers to backups or snapshots of file systems. As a guiding principle throughout this dissertation, mechanism and interface designs must operate without the explicit

support of a monitored system. For the remainder of this dissertation, we consider the terms monitored system, guest, cloud instance, and virtual machine to be synonymous and interchangeable.

There are four key challenges enumerated below that this dissertation addresses in answering this fundamental question.

1.1.1 Bridging the Semantic and Temporal Gaps

The normal path for an application system call destined to write to a disk moves from a guest's userspace, to the guest's kernel, and then out to the hypervisor modifying the virtual disk. This path is leveraged by agents as they can hook system calls, register for disk state change notification, trace disk operations, and generally use the guest's kernel to directly accomplish their job. An agentless solution, by definition, does not rely on any guest support whatsoever. Even relying on guest hints makes an agentless method vulnerable to guest misconfiguration or compromise. Agentless methods receive black box operations at the lowest abstraction layer, and must reconstruct or infer higher-level meaning.

The challenge is to bridge this *semantic gap* between black box disk-level operations and their white box file-level interpretation with significantly less context than the guest kernel. In addition, layers often reorder operations for better throughput or latency. File systems generally update data first, metadata second. Thus, there is an additional *temporal gap*–the time between when operations flush to disk, and when they have semantic meaning within the guest file system. Atomic operations, for example, wait until their instructions finish before persisting that their operation finished. The temporal gap is the time between the start of the atomic operation, and the persistence of their completeness.

1.1.2 Bounding Overhead

Monitoring clearly comes at a cost to the system being monitored. We monitor systems for various reasons, but if the monitoring cost exceeds the value of the benefit derived from monitoring, then monitoring does not make sense. Thus, there must be *boundable overhead* for any monitoring solution—agent or agentless. Should monitoring require upgrading the hardware capabilities of all hosts in a production cloud, for example increasing their memory, this cost will more than likely exceed the value of monitoring. In addition, a key metric in virtualized cloud environments is consolidation. Monitoring overheads directly hurt the consolidation factor of a cloud environment by using resources which could have otherwise been devoted to valuable production workloads. In the extreme case, boundable overhead means the capability of completely eliminating overhead. This implies the capability of turning on and off monitoring at will.

1.1.3 Generality

Monolithic design places every component inside the blackbox boundary of the virtual machine. This is attractive because all of the configuration state, necessary libraries, and versions of key

system components are all kept together for a specific application. Indeed, this model has proven wildly successful with the use of virtual appliances for deploying software. However, this means that monitoring agents must have implementations and logic for each operating environment they support. The interfaces to an agentless monitoring platform should not needlessly tie themselves to an OS family or specific environment. Agentless monitoring should provide a single set of interfaces to persistent state that not only scales in the *number* of monitored guests, but also in the *types* of file systems and OS's it can monitor. In other words, agentless monitoring must generalize across OS's and applications.

1.1.4 Storing and Indexing Historic Changes

As promised, this dissertation handles not only online persistent state, but also offline persistent state. Due to modern trends in backup architectures providing low-latency, high-bandwidth access to data [81], we believe a critical feature of modern backup should be search—especially with tunable indexes. Backup subsumes so many different systems that a single indexing scheme seems unlikely to fit all present and future search needs. Thus, a key feature of modern backup should be optimized indexing over stored objects. In this dissertation we rethink storage formats for file-level backups given the existence of our storage monitoring mechanism. In addition, we explore techniques for reducing the time to index in an application-agnostic manner.

1.2 Thesis Statement

This dissertation claims that *agentless monitoring of disk state provides stronger security and correctness guarantees than traditional agent-based approaches, is achievable with modest modifications to modern operating environments, and enables generalizability across the combinatorial space of operating systems, libraries, applications, and their versions.* The cost of realizing agentless monitoring of disk state lies directly with the effort needed to deeply interpret virtual disk write operations—specifically the on-disk layout and data structures of file systems. That is to say, it is a capital intensive endeavor, but one that pays off over time. The upfront costs of writing file-system-specific introspection logic are amortized over a long time period. This is because file systems rarely change their on-disk layouts.

The vision of this thesis for cloud computing is that by slightly weakening the strong isolation between virtual machines and their underlying cloud infrastructure, we can provide a more secure environment by taking advantage of the tension between cloud customer, cloud operator, and monitoring vendors. Cloud customers wish to bound the information that monitoring vendors access. Monitoring vendors wish to maintain correct configurations and views of monitored state. Cloud operators wish to increase revenue by offering valuable services to their customers. Thus, the cloud operator is perfectly poised as a mediator guarding paying customers from monitoring vendors, and providing information independent of customer misconfiguration or compromise to the monitoring vendor. Monitoring vendors benefit with greater name recognition via the cloud marketplace, and interfaces which ease their generalization across cloud customer operating environments.

The research questions this dissertation addresses are: (1) What is the minimal set of extensions to modern environments needed for agentless persistent state monitoring? (2) How does agentless monitoring of persistent state change the implementation of file-level monitoring? (3) How does agentless monitoring of persistent state change the implementation of snapshotting? (4) How can backup systems minimize the time to index all stored objects and their versions?

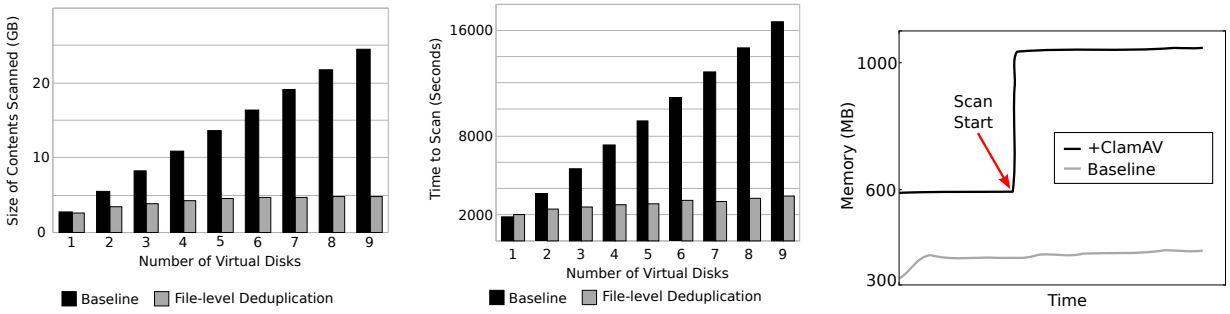
1.3 Contributions

This dissertation challenges the status quo monolithic system design which embeds monitoring agents into the systems they monitor, with an agentless vision separating the mechanism of monitoring from the policies which act upon monitored information. In so doing, this dissertation disrupts a billion dollar industry [40]. We show that the long path towards an agentless world, with all of its benefits, is in fact tractable. It is feasible with most modern hypervisors to pursue today, in some cases with minimal disruption to existing deployments and operations. Should the mechanism described in this dissertation become widely available, switching to agentless monitoring is very difficult to argue against.

Agents can not fundamentally offer the guarantees of an agentless approach. No agent-based solution can guarantee isolation from monitored system misconfiguration or compromise. By implementing policy with mechanism inside the monitored system, corruption of either can directly affect the other. No agent-based solution can guarantee that its bugs will not affect the monitored system. In fact, guaranteeing bug freeness is undecidable without constraints. No agent-based solution easily generalizes across operating environments. Because of the deep coupling of agent and the monitored system, agents must needlessly cater to their environment rather than just their task. Thus, if the technology espoused in this dissertation is widely implemented, it becomes very difficult to justify the further use or development of agent-based monitoring solutions for persistent state. Agentless solutions also enable optimizations which are very difficult to implement with agent-based solutions. Agentless monitoring has the capability of leveraging global knowledge about collections of systems and resource allocations. For example, agentless monitoring can deduplicate across collections of virtual disks and schedule monitoring tasks efficiently with production VM workloads without mixing resource allocations.

Figure 1.1 shows the wins with an agentless model that leverages global knowledge—file-level duplication across VMs—for a file system scanning workload. In this example, traditional scanning agents execute independently inside each VM. It would be non-trivial and also inefficient for each type of scanning agent to implement file-level deduplication independently. As reflected in Figure 1.1(a), a virus scanning agent reads duplicated files multiple times wasting significant read-bandwidth. Processing time is also wasted scanning the same files over and over again as reflected in Figure 1.1(b). Simply running an agent uses resources of the VM—in this case significant amounts of memory as shown in Figure 1.1(c). Instead of wasting this memory n times for VMs which might all have the same files, an agentless implementation could bound the memory usage devoted to virus scanning across the entire cloud. These wins are not limited to file system scanning work-

loads. First, the cloud can separately bound global resources devoted to *any* monitoring workload by decoupling its execution from individual VM execution. Second, deduplication also benefits streaming workloads. For example, agents monitoring changes to configuration files could receive coalesced updates from multiple VMs across the cloud rather than processing duplicate file-level changes individually. These optimizations enhance the scalability of future clouds, and directly lead to lower total cost of ownership and higher revenue for all parties involved.



(a) Aggregated read bandwidth. (b) Aggregated processing time. (c) Memory of a single VM.

Figure 1.1: ClamAV's [21], a file system scanning agent, memory overhead and aggregated resource usage (Figures 1.1(a) and 1.1(b) reconstituted from [124]).

The contributions of this thesis, in addition to answering the aforementioned research questions, are:

- Demonstrates feasible, efficient, generalizable agentless monitoring of disk state (Chapter 2)
  - Full reference implementation released under the Apache v2.0 License
  - Support for the `NTFS`, `ext4`, and `FAT32` file systems
  - API integration into the OpenStack cloud software
- Demonstrates three generalizable interfaces to disk state (Chapters 3, 4, and 5)
  - `cloud-inotify` – for real-time agents (publish-subscribe interface, live writes)
  - `/cloud` – for scanning/batch agents (legacy applications, file system interface, on-disk)
  - `/cloud-history` – for historic data access (file-level deduplicated snapshots, archived)
- A backup study of a research group with 58 unique hosts over 1 year (Chapter 5.2)
  - 3,268 file system snapshots
  - 1.676 billion referenced files
  - 146 TiB of crawled file state

- A log-structured format for `/cloud-history` based on agentless monitoring (Chapter 5.5)
  - Heuristic time-based snapshotting of files
  - Garbage collection of log state
  - Application-agnostic indexing speedup via whole-file deduplication