Scalable Cloud Security via Asynchronous Virtual Machine Introspection

Sundaresan Rajasekaran, Zhen Ni, Harpreet Singh Chawla, Neel Shah, Timothy Wood
George Washington University

Emery Berger
University of Massachusetts Amherst

Abstract

Software will always be vulnerable to attacks. Although techniques exist that could prevent or limit the risk of exploits, performance overhead blocks their adoption. Services deployed into the cloud are typically customer facing, leaving them even more exposed to attacks from malicious users. However, the use of virtual machines, and the economy of scale found in cloud platforms, provides an opportunity to offer strong security guarantees to tenants at low cost to the cloud provider. We present ScaaS, a security Scanning as a Service framework for cloud platforms that uses frequent virtual machine checkpointing coupled with memory introspection techniques to detect bugs and malicious behavior in real time. By buffering VM outputs (i.e., outgoing network packets and disk writes) until a scan has been completed, ScaaS gives strong guarantees about the amount of damage an attack can do, while minimizing overheads.

1 Introduction

Despite decades of advances in areas ranging from testing to static analysis and verification, all large real-world software is deployed with errors. Because this software is either written in or underpinned by unsafe languages, errors often translate to security vulnerabilities. While techniques exist that could prevent or limit the risk of exploits, costly performance overheads block their adoption, leaving today's systems open to attack.

Security is an especially important topic in cloud environments, since they are often used for running complex services that may be open to the public. Such software is an appealing target to adversaries, since breaking the weakest link in a distributed application can often give attackers access to other components. As a result, security has been consistently cited as one of the foremost problems for IT professionals, ranking above areas such as QoS and resource efficiency that are often the focus of academic research [1]. Cloud providers have an increasing array of products to help customers improve the performance and monitor the reliability of their applications. Despite this, security features remain limited, typically restricted to network issues like firewalls and VPNs. What is needed is a way for cloud platforms to provide security functionality as a service, similar to what they provide today for virtual machine performance management.

Towards this end we are developing ScaaS, a Scanning as a Service framework for cloud data centers. ScaaS takes advantage of recent advances in memory forensics and VM checkpointing to provide an efficient and scalable platform that can scan for a wide range of attacks within both applications and the operating system. Unlike past approaches based on expensive VM record and replay [2], ScaaS uses an asynchronous checkpointing mechanism to replicate a VM's memory onto a Scanner host dozens of times per second. The Scanner then uses VM introspection techniques to study the memory of the virtual machine, efficiently targeting the changed memory pages for analysis while retaining context of the full system. The VM runs on the primary as normal between checkpoint intervals, but all of its outputs (e.g., potentially malicious network packets and disk writes) are buffered until the end of a checkpoint. At that time, the Scanner provides a status message indicating whether the VM has been compromised, or if it is safe to release the outputs and continue running. This provides strong guarantees that attacks will be detected within tens of milliseconds, and that an attack can cause no externally visible effects.

In this paper we describe our preliminary design for ScaaS and analyze the feasibility of our approach. We focus on the overheads incurred by checkpointing and scanning to illustrate the platform's scalability in a cloud environment where a small number of Scanner hosts can be multiplexed for a large number of protected VMs.

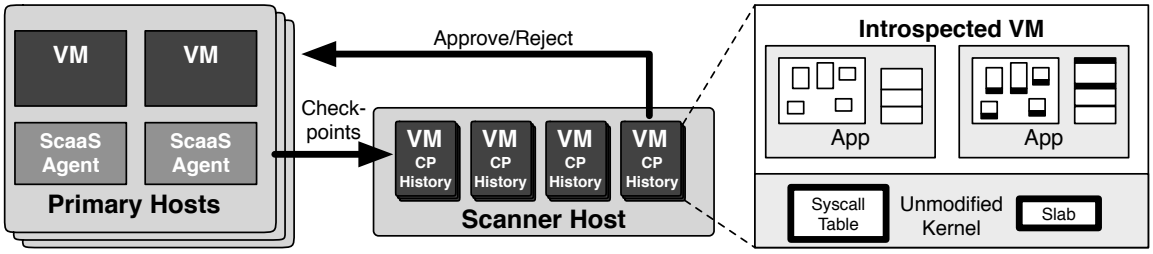

Figure 1: Virtual machines run on ordinary cloud hosts, periodically sending checkpoints to the Scanners for analysis. A Scanner host uses VM introspection techniques to search for evidence of vulnerabilities, such as validating canaries in the OS and applications, or ensuring the integrity of key kernel data structures.

2 Background and Related Work

Virtualization platforms have grown in popularity primarily because they facilitate application deployment and server consolidation. The hypervisor's abstraction layer also allows new services to be transparently provided to virtual machines, often by treating the VM as an opaque black box. Here we describe some of the prior work that used this abstraction layer to provide security services, as well as the introspection techniques that let the hypervisor understand the internals of a running VM.

Prior work explored how the hypervisor can offer security services such as virus scanners, honeypots, and intrusion detection services [3–5]. The AfterSight system is most closely related to our work, and used deterministic record and replay techniques to analyze a second copy of a VM for security issues in real time [2]. However, record-replay has very high cost both on the primary (especially to ensure non-determinism for multi-core virtual machines) and on the secondary (which must fully re-execute every instruction from the primary). Thus if a VM on the primary is using four CPU cores, each at 75% load, the secondary machine will also consume at least as many resources in order to replay the VM and analyze its state. In fact, the record-replay feature in the VMware hypervisor used to build AfterSight was discontinued in 2011. ScaaS avoids this high cost by using checkpoint analysis, rather than record-replay. While this still consumes memory resources on the secondary server, it makes the analysis of large numbers of protected virtual machines much more efficient.

ScaaS takes advantage of the Remus high availability system [6, 7] included in recent versions of Xen to create checkpoints. Remus continuously transmits checkpoints of a virtual machine to a secondary host. We extend this to perform security analysis on the checkpoints as they are saved to the backup.

Analysis of checkpoints is achieved through Virtual Machine introspection (VMI), which exploits knowledge of a VM's operating system to interpret its internal state [5, 8]. Given a symbol map of the precise kernel version being run in a VM, tools such as libVMI can easily locate kernel data structures within the VM's memory and translate addresses from guest to physical memory [9]. VMI can be used in an offline mode for forensic analysis/debugging a crashed virtual machine, or it can be performed on a live VM. A natural challenge when using introspection for security purposes is that a malicious agent might manipulate the kernel state to block introspection techniques. ScaaS assumes that the system is started in a clean state and that such attacks can be detected before they resolve, but more advanced techniques such as semantic view reconstruction could also be used to avoid this problem [3].

3 ScaaS Platform Overview

The structure of ScaaS is shown in Figure 1. Each host in the cloud platform runs one or more virtual machines, each of which is managed by a ScaaS Protection Agent. The ScaaS Agent is responsible for creating checkpoints of the virtual machine and transmitting them to a Scanner host. Each Scanner host can be multiplexed to receive checkpoints from many different hosts, providing high scalability.

ScaaS works by breaking a VM's execution down into scan intervals. In the simplest form (which we further optimize in the following sections), a VM runs as normal between each security scan with no interruption. At the end of the interval, the VM is paused and its memory state is copied by the ScaaS Agent to the Scanner host for analysis. If the Scanner does not find any errors in the checkpoint, it notifies the VM's Agent, which begins execution of a new interval.

If scan intervals are on the order of hours or days, this approach is similar to how a virus or malware scanner typically works on a PC. However, this clearly leaves the target vulnerable for long periods between scans, during which time an attacker may cause substantial harm. The goal of ScaaS is to reduce the scan interval from hours to milliseconds, allowing an attack to be promptly caught. We achieve this by performing asynchronous checkpoints as the VM runs, allowing dozens of scans to be performed per second at low cost. Further, by buffer-

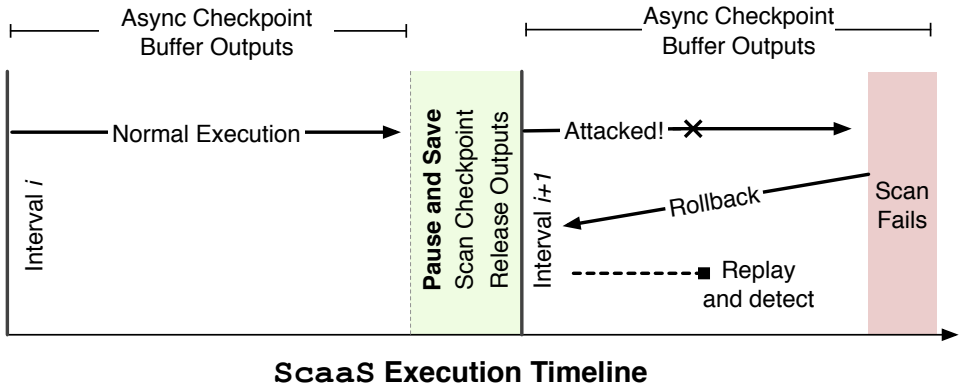

Figure 2: A VM executes normally during each checkpoint interval, only pausing briefly at the end so the checkpoint can be scanned for security vulnerabilities. If an attack is found, the VM can be rolled back and analyzed more carefully as it runs to precisely detect the point of attack.

ing outputs we can ensure that even if a VM is compromised between scans, it will not be able to adversely affect other servers. This is illustrated in Figure 2, where the VM's memory is asynchronously checkpointed to the Scanner host during two checkpoint intervals. At the end of an interval, the VM is paused so a consistent checkpoint can be made and the full security scan can be performed. If a scan ever fails, the VM can be paused for later forensic analysis, or rolled back to the last checkpoint and resumed under careful introspection, allowing the attack to be analyzed while it is in progress.

3.1 Making Secure Checkpoints

Checkpoints are the basis for security in ScaaS because they provide a stable snapshot of a virtual machine that can be analyzed or rolled back to. Creating checkpoints of a virtual machine entails copying its memory contents, CPU state, and possibly its underlying disk. Checkpointing is a well studied research area, and various approaches have been proposed to optimize both live VM checkpointing and migration, which typically rely on the same hypervisor functionality [6, 7].

In ScaaS we build on the Remus high availability system included in the Xen hypervisor [6]. Remus provides high availability by continuously creating snapshots of a virtual machine and replicating them to a secondary server. Through optimizations including dirty page tracking and compression, Remus is able to perform dozens of checkpoints per second with acceptable overhead. Remus also uses output buffering to delay the network packets and disk writes of a virtual machine until a checkpoint has been persisted to the backup machine, providing synchronous reliability guarantees at low cost. ScaaS extends Remus to record a history of checkpoints on the secondary server, scan them for vulnerabilities, and provide a framework for controlled replay and forensic analysis.

The Scanner host maintains multiple past stable checkpoints and a pending checkpoint holding the updates from the current scan interval. The checkpoint history can be useful for forensic analysis after an attack to understand how the system evolved prior to detection. However, maintaining a complete history of all checkpoints quickly becomes impractical since dozens may be created per second. We are exploring techniques to intelligently prune the checkpoint history. Rather than simply store the last N checkpoints, ScaaS will seek to keep checkpoints that correspond to important events; for example, ScaaS might detect that a checkpoint corresponds to when a new user logged into the server over SSH, and flag that as an important point to retain. Checkpoints are tagged with this kind of information by the Scanner's introspection engines described in the following section.

3.2 Asynchronous Security Scanning

ScaaS provides a framework for running security scans to detect evidence of recent exploits. While some security systems work by intercepting instructions or function calls that could trigger an exploit, this typically has high overhead and must be run on each application [10, 11]. For example, Google's Address Sanitizer can detect buffer overflow attacks, but it incurs a 30% performance overhead. In contrast, ScaaS uses an evidence based approach [12], meaning that it runs the virtual machine as normal but then periodically inspects key OS or application data structures for signs that an attack occurred. For example, ScaaS could monitor the kernel's memory pages storing the system call table to detect an exploit that attempts to overwrite entries, or ScaaS could fingerprint an application's memory to detect evidence of a stack smashing attack.

We are developing scans that can be performed syn-

chronously and asynchronously. A synchronous scan is one that requires a consistent image of the VM's memory, thus it can only be performed at the end of a checkpoint interval when the full memory state has been received. In contrast, scans such as memory fingerprinting do not need a fully consistent image and can be performed asynchronously as memory pages arrive. Performing scans asynchronously can significantly reduce overhead since it reduces the time during which the primary VM is paused.

While memory-based scans are done on the Scanner host, we are also investigating what type of scans can be efficiently run on the same host as the primary VM. For example, we are developing an intrusion detection engine that scans the packets in ScaaS's network buffer. This could, for example, fingerprint packets to detect worm traffic signatures, allowing them to be caught more easily than through memory analysis. These scans can be done without adding significant overhead on the primary, but allow a broader range of attacks to be detected.

3.3 Attack Detection and Response

An attack can be detected at any time by scans on the primary host (e.g., from network packet buffer analysis), at the end of a checkpoint interval by a synchronous scanner, or at any time by an asynchronous scanner. ScaaS can respond to an attack in one of several ways.

Forensic analysis: In the simplest case, when an attack is detected the primary VM is immediately terminated, and the most recent checkpoint as well as information about where the attack was detected is saved for future analysis. The checkpoint can be analyzed with detailed forensic tools which are not practical to run on live VMs in order to more precisely identify the source of the attack. ScaaS could also provide a history of checkpoints for analysis, allowing the investigator to see how the attack evolved over time.

Rollback and Replay: Alternatively, the VM can be rolled back to its most recent checkpoint and immediately resumed on the secondary host. More intrusive analysis tools could then be applied to precisely identify the attack. For example, if the detected attack caused a buffer overflow at a particular memory address, a hardware breakpoint could be enabled at that location. During re-execution, the breakpoint would be triggered, allowing the VM to be paused at the exact point of the attack (instead of a checkpoint shortly before or after the attack). This would give the forensic investigator precise information about the system state, including application call stacks, when the attack occurred.

Honeypot Mode: Since the VM will be resumed on a Scanner host, it is also easy to transition the VM to act as a honeypot after the attack is detected. In this mode, the VM is resumed on the Scanner, but it is kept isolated from sensitive resources (e.g., it may be connected to a sandboxed network or dummy disk). The VM can then continue executing after the attack occurs, but with all of its outputs carefully examined and sanitized to prevent infection of other active nodes. This would allow security investigators to observe how the attack evolves, which can be useful for building detection and prevention systems.

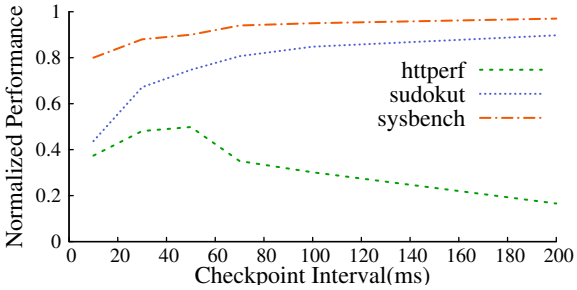

Figure 3: Scan Frequency Overhead

4 Prototype Evaluation

We are building a prototype of ScaaS using the Xen hypervisor version 4.5.2, the Remus checkpointing tool, and libVMI for introspection. Our ScaaS framework introduces scanner modules that can be hooked into the checkpoint creation code on the primary side or in the checkpoint receiver on the Scanner. Our current prototype supports two simple scans:

A *Memory Fingerprinter* that hashes the memory pages in each checkpoint so they can be compared against either malware signatures or "known good" state.

A *Process Black/White List Enforcer* that uses introspection to detect the processes currently running in the VM. It can then trigger an error depending on whether a target process is running or not.

We also test a scanner that performs a configurable amount of computation at the end of each checkpoint in order to emulate scanners of varying cost.

4.1 Checkpoint Overhead

We first measure the overhead of different checkpoint intervals on a variety of application benchmarks. In these experiments we disable all security scans to find the baseline cost of continuously replicating a virtual machine's memory state. We send checkpoints to the Scanner host over a 1Gbps link while buffering outgoing network packets between each interval.

Figure 3 shows how several benchmarks perform under different checkpoint intervals. For applications that are CPU intensive (sysbench and sodokut), a longer checkpoint interval has better performance since it means VMs are paused less frequently. On the other hand, using httperf to access an apache webserver is a network latency sensitive applications, giving a more com-

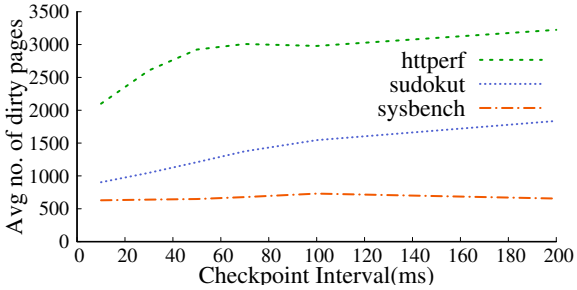

Figure 4: Memory Dirty Rate

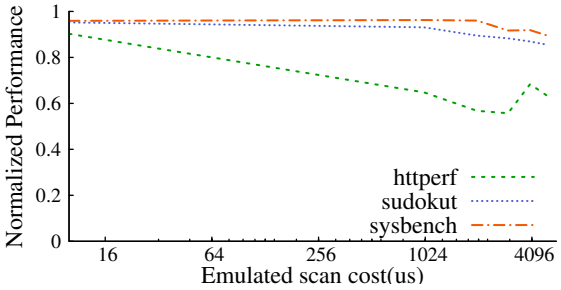

Figure 5: Scan Cost Overhead

plex trend—a very low checkpoint interval causes higher overhead due to pauses, but a longer interval can substantially delay network packets.

The overhead of the network benchmark httperf is unexpectedly high compared to numbers in prior work; we believe that this can be improved by using some of the optimizations to Remus proposed by SecondSite [7]. In our ongoing work we are looking at sources of overhead in the checkpointing process including page mapping and unmapping for each iteration in a checkpoint, shadow page table operations, and fast suspend/resume techniques at the end of each checkpoint.

4.2 Scan Cost and Scalability

To understand the cost of security scans we first measure the page dirty rate; this indicates the volume of data that needs to be transferred to and analyzed by the scanner. Figure 4 shows the memory dirty rate when adjusting the number of checkpoints created per second.

Figure 5 shows how application performance changes when we add an emulated scan of different costs during the pause period at the end of each checkpoint. We normalize performance compared to a zero-cost scan when making checkpoints every 50 milliseconds. The CPU-intensive benchmarks again see little overhead as the scan cost rises, while httperf worsens due to the added latency before packets are released from the network buffer.

Finally, we measure the CPU usage on the Scanner host to understand the scalability of our approach. Figure 6 reports the CPU usage required by remus to receive

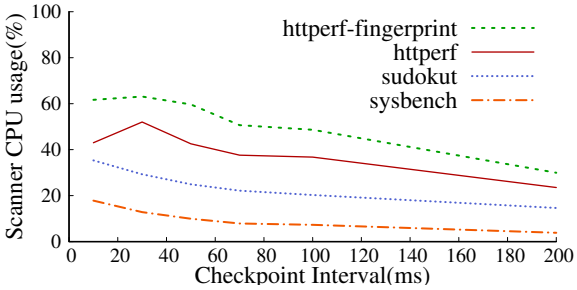

Figure 6: Scanner CPU Usage

checkpoints, as well as the CPU usage overhead when the VM hashes memory pages as they arrive. The CPU consumption changes depending on the workload and the checkpoint frequency. Running all received memory pages through our fingerprint scanner adds further overhead as shown in the *httperf-fingerprint* line, but that overhead becomes negligible with the increase in the checkpoint interval. We find that we can fingerprint about 91,000 pages per second with one CPU core. Our process black list scanner takes about 1 millisecond to search through the process list at the end of each checkpoint. These results show that even when performing 100 checkpoints per second, the CPU usage stays below 50% for all checkpoint intervals above 70ms, suggesting that a Scanner host with 10 or more CPU cores could easily handle a significant number of VMs.

5 Conclusions

Running applications within virtual machines provides new ways to deploy security systems. We are building ScaaS so that cloud platforms can provide scalable attack detection and forensic analysis for their customers. The key insight in ScaaS is that many attacks can be detected after the fact by examining memory checkpoints with VM introspection. By deferring external outputs (i.e., network packets and disk writes) until a checkpoint has been scanned, ScaaS will be able to offer strong guarantees about the damage an attack can cause.

A number of challenges remain to make ScaaS a reality: the checkpoint overhead must be decreased (especially for multi-core VMs), the history of checkpoints maintained at the Scanner host needs to be pruned in a way that keeps critical past checkpoints while limiting memory and storage use, and introspection-based security scanners need to be developed to detect a range of realistic attacks. We believe that the ScaaS framework holds promise to scalably protect critical VMs while incurring reasonable performance overhead.

Acknowledgments: This work was supported in part by NSF grants CNS-1525992 and CNS-1525888.

References

[1] *Top 5 cloud computing challenges | Trilogy*. June 2015.

[2] J. Chow, T. Garfinkel, and P. M. Chen. "Decoupling Dynamic Program Analysis from Execution in Virtual Environments". In: *USENIX 2008 Annual Technical Conference*. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14.

[3] X. Jiang, X. Wang, and D. Xu. "Stealthy Malware Detection Through Vmm-based "Out-of-the-box" Semantic View Reconstruction". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS '07. New York, NY, USA: ACM, 2007, pp. 128–138.

[4] R. Riley, X. Jiang, and D. Xu. "Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing". In: *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*. RAID '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–20.

[5] T. Garfinkel, M. Rosenblum, and others. "A Virtual Machine Introspection Based Architecture for Intrusion Detection." In: *Networked and Distributed System Security Symposium*. Vol. 3. NDSS. 2003, pp. 191–206.

[6] B. Cully et al. "Remus: High Availability via Asynchronous Virtual Machine Replication". In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 161–174.

[7] S. Rajagopalan et al. "SecondSite: Disaster Tolerance As a Service". In: *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*. VEE '12. New York, NY, USA: ACM, 2012, pp. 97–108.

[8] B. Hay and K. Nance. "Forensics Examination of Volatile System Data Using Virtual Introspection". In: *SIGOPS Oper. Syst. Rev.* 42.3 (Apr. 2008), pp. 74–82.

[9] B. D. Payne. "Simplifying virtual machine introspection using libvmi". In: *Sandia report* (2012).

[10] K. Serebryany et al. "AddressSanitizer: A Fast Address Sanity Checker". In: *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*. USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 28–28.

[11] C. Cowan et al. "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-overflow Attacks". In: *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*. SSYM'98. Berkeley, CA, USA: USENIX Association, 1998, pp. 5–5.

[12] T. Liu, C. Curtsinger, and E. D. Berger. "DoubleTake: Fast and Precise Error Detection via Evidence-Based Dynamic Analysis". In: *International Conference on Software Engineering*. May 2016.