Digital Investigation 26 (2018) S98–S106

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2018 USA — Proceedings of the Eighteenth Annual DFRWS USA

Who watches the watcher? Detecting hypervisor introspection from unprivileged guests

Tomasz Tuzel[*], Mark Bridgman, Joshua Zepf, Tamas K. Lengyel, K.J. Temkin

Assured Information Security, Greenwood Village, CO, USA

Keywords:
Virtualization
Hypervisors
Virtual machine monitors
Cloud computing
Wall timing
Caches
Side-channel attacks
Non-temporal instructions

A B S T R A C T

We present research on the limitations of detecting atypical activity by a hypervisor from the perspective of a guest domain. Individual instructions which have virtual machine exiting capability were evaluated, using wall timing and kernel thread racing as metrics. Cache-based memory access timing is performed with the Flush + Reload technique. Analysis of the potential methods for detecting non-temporal memory accesses are also discussed. It is found that a guest domain can use these techniques to reliably determine whether instructions or memory regions are being accessed in manner that deviates from normal hypervisor behavior.

1. Introduction

Cloud computing offers many benefits to organizations of all sizes: economies-of-scale net cost savings, elasticity provides seamless scalability, and consolidation of Information Technology (IT) resources improves service quality and security. To facilitate cloud migration, modern hypervisors aim to minimize the differences between executing in a virtualized environment and on bare-metal by using hardware extensions to multiplex virtual machines (VMs) seamlessly and with minimal performance impact. The hypervisor's position of privilege on the system can come with a negative: a compromised hypervisor is able to introspect and corrupt its VMs, bypassing data protections and giving the adversary control over processing.

The ability of a cloud tenant to detect if and when a host is behaving in an unorthodox or outright intrusive fashion can be valuable in determining whether the platform is to be trusted. As numerous organizations continue migrating services to the cloud, it is essential that software be able to determine the trustworthiness of the environment in which it is executing as well as optimally respond to possible threats.

In this paper, we present our findings regarding the utilization of hardware side-channels to gain insight into computing environments, the limitations of this technique, and the potential for developing a framework to determine optimal responses. Using hardware side-channel information, we have evaluated the feasibility of using shared CPU resources to characterize privileged software. Herein, we provide a body of research regarding the limitations of environmental characterization of virtualized platforms.

Our tool, *Environmental Characterization and Response (ECR)*, analyzes instructions and memory accesses on a guest system which has been deployed on a hypervisor. *ECR* leverages a variety of metrics to determine the potential presence - or lack - of introspection, and serves to establish the limits of attack and limits of detection touched upon earlier. The *ECR* effort developed a novel technology capable of characterizing a cloud platform's privileged architectural software from within an unprivileged environment, providing the foundation for development of autonomous, self-protecting cloud applications.

Our contributions are as follows:

- Provide an in-depth overview of the effects of virtualization on shared hardware resources from a micro-architectural perspective
- Evaluate the efficacy of several timing techniques to supply a robust baseline to build detection systems
- Perform extensive experiments on the capability and limitations of detecting a variety of introspection techniques, including hypervisor accesses to particular in-guest memory ranges, instruction trapping and memory access tracing

* Corresponding author.
E-mail addresses: tuzelt@ainfosec.com (T. Tuzel), bridgmanm@ainfosec.com (M. Bridgman), zepfj@ainfosec.com (J. Zepf), lengyelt@ainfosec.com (T.K. Lengyel), k@ktemkin.com (K.J. Temkin).

https://doi.org/10.1016/j.diin.2018.04.015

2. Related work

The topic of malicious hypervisors has been widely discussed and has produced a significant body of work over the years. From the outset, there have been concerns that adding layers underneath the operating system (OS) will result in systems that may undermine, or outright compromise, the security and privacy of the OS (Rutkowska, 2006; Zovi, 2006).

For the detection of such hypervisors, many techniques have relied heavily on finding implementation-specific artifacts (Ferrie, 2007). Widely available open-source tools today showcase this approach by detecting hardware or software artifacts exposed to the guests by specific hypervisors (Paranoid Fish, 2018). It has also been proposed to utilize even lower layers for detection, such as the System Management Mode (Rutkowska and Wojtczuk, 2008). There has also been research which evaluated the notion of looking for hardware side-effects that a hypervisor would inadvertently introduce to the system (Thompson; Brengel et al., 2016; Fritsch, 2008).

In today's computing environment, however, the existence of a hypervisor is commonplace. Most of the research efforts thus far have not made a distinction between the detection of a hypervisor and the detection of an *introspecting* hypervisor. The research that is available is focused on the evaluation of the stealth attributes of malware analysis systems, such as Ether (Dinaburg et al., 2008) or DRAKVUF (Lengyel et al., 2014). Research into the limitations of these stealth approaches mainly involved looking for specific artifacts, such as discrepancies in the behavior of timing sources as these are being manipulated by the sandbox (Pék et al., 2011).

3. Background

In the following section we provide a brief but in-depth background for the concepts that *ECR* is built upon.

3.1. Virtualization

The creation of a VM that behaves as a typical hardware-based machine running a standard OS, but is separated from the actual physical hardware resources of the hosting system is commonly referred to as virtualization. This technology has lent itself to the birth of the cloud, which offers immense cost-savings through data-center consolidation, centralization of IT, and purchasing power of providers, however, there are security concerns with moving to cloud infrastructure. The most serious of these concerns being the risk of malicious software compromising a hypervisor and utilizing this privileged position to interfere with the operation of VMs, or observe sensitive data in those VMs.

3.2. Hypervisors

A hypervisor is a piece of privileged, low-level software that supervises the execution of guest VMs, and is typically responsible for maintaining isolation between those VMs. To provide a guest experience consistent with running on real hardware, a hypervisor typically shares hardware resources between VMs, directly or indirectly multiplexing access to real hardware resources. There are two generally accepted classifications of hypervisors: type-1 and type-2. A type-1 hypervisor is a bare-metal hypervisor, in which the hypervisor runs directly on the hardware. A type-2 hypervisor is a hosted hypervisor, in which a hypervisor runs as a process on the base OS. In this effort, the Xen Project hypervisor, which is a type-1 hypervisor, was utilized.

Typically in the Xen architecture, the core hypervisor only directly arbitrates access to a few critical system resources, including the CPU and RAM. To mediate access to the remaining hardware, Xen creates a domain known as dom0, which is empowered with the ability to perform hardware access by mapping hardware resources directly into that domain, creating a domain that is uniquely privileged, but which still has significantly less privilege than the virtual machine monitor (VMM) itself. In most use cases, the hardware domain typically runs a standard Linux distribution, such as Red Hat Enterprise Linux (RHEL) or Debian, which provides the drivers used for hardware interfacing and multiplexing software used to route networking traffic to and from the guests.

3.3. Virtual machine exits

To provide an environment capable of executing user workloads that include unmodified system software, hypervisor platforms must be capable of interceding when a guest attempts to perform operations that can impact the state of the real hardware. To enable efficient intercession, processor virtualization technologies, such as Intel's VT-x, provide hardware features that allow hypervisors to assume control once a privileged operation is attempted. As such, the hypervisor software has an opportunity to replace the relevant operation with its own handlers, which often perform equivalent operations to real hardware while limiting scope only to the active VM.

In VT-x terminology, a virtual machine exit, or VM-exit, is a point at which guest execution is paused and execution is returned to the hypervisor, which can then opt to intercede on the guest's behalf. To allow convenient world switches, VT-x based hypervisors use a special-purpose region of memory known as a VM Control Structure (VMCS), which is a data structure consisting of six logical groups that handles hypervisor operations and state transitions between the hypervisor and the guest.

During an exit (Ott, 2018):

1. The cause of the exit is recorded in the VM-exit information fields.
2. The current processor state is saved in the guest-state area.
3. The model-specific registers (MSRs) are stored in the VM-exit MSR-store area.
4. The processor state is loaded from the host-state area and VM-exit controls.
5. The MSRs are loaded from the VM-exit MSR-load area.

Once the hypervisor completes its operations, a VM-entry will be performed to transition control back to the guest. Since the states are stored in main memory, the entire routine results in significant overhead, due to generally low access rates versus a processor cache. This is key to enabling detection of an introspective hypervisor.

3.4. Timers & timing methods

Modern x86-64 platforms contain a variety of timers and timing methods. These include the *x86 Timestamp Counter (TSC)*, the *High Precision Event Timer (HPET)*, and the *Advanced Configuration and Power Interface Power Management Timer (ACPI PMT)*. The first two timers are of interest to us in this paper, and as such, will be addressed here.

The TSC is a high precision timer on-board modern x86 systems which is precise enough to measure individual processor clock cycles. This precision makes it a typical timing source used for analysis of timing-based side-channels, but it is important to note that the TSC value is easily modified by hypervisors. Hypervisors can easily intercede in requests for TSC values, and often do so for legitimate purposes, such as the suspension and resumption of a guest, or malicious reasons such as thwarting attempts at