Guest-Based System Call Introspection with Extended Berkeley Packet Filter

by

Huzaifa Patel
Supervisor: Anil Somayaji

A thesis pre-proposal submitted to the School of Computer Science in partial fulfillment of the requirements for COMP 4106

Carleton University
Ottawa, Ontario
September 2022

Introduction

Virtualization is a technology that makes it possible for multiple operating systems (OSs) to run concurrently, and in an isolated environment on a single physical machine. There are many techniques to achieve full virtualization of a computer system, one of which is hardware-assisted virtualization. Hardware-assisted virtualization is the use of a extensions such as Intel Virtualization (VT-x) to support the software that creates and manages virtual machines (VMs). The software that creates and manages VMs is formally known as a hypervisor. The operating system that is capable of running a hypervisor is called the host, while the VM that uses the hypervisors resource is known as the guest.

Motivation

As our computer systems continue to grow increasingly complex, so too does it become more difficult to measure precisely what they are doing at any given moment [1]. As a result, users often have a limited self-awareness of what is happening on their computer systems internal states. An unfortunate consequence of the lack of self-awareness in this domain is that it decreases the likelihood of spotting and appropriately reacting to malicious anomalies. If users are unable to adequately monitor their computer systems, computers should be programmed to watch over themselves [2].

This thesis is a first step towards addressing and fixing this flaw. In the remaining portion of this section, We will attempt to demonstrate why system calls are a valuable mechanism for effectively monitoring computer systems.

As the Linux kernel operates in ring 0, and the user space operates in ring 3, the user space cannot interact or access the Linux kernel directly. When the Linux kernel is needed to perform services for a user space process, a system call must be performed. Given that processess are heavily dependent on system calls for interaction between user space and kernel space, they become interesting from a security perspective. For

example, system call traces can be used to classify the actions of a process as benign or malicious. For this reason, it is essential to trace and observe system calls in computer systems.

Main Objectives

Our goal is to create a framework flexible enough to support VM system call introspection from a host that is using Intel x86 architecture. Our framework will use virtual machine introspection (VMI), a technique of monitoring and analyzing the state of VMs from the hypervisor level. We will make use of the Kernel Virtual Machine (KVM) hypervisor, which utilizes hardware-based virtualization. Hardware-based virtualization support from Intel was not designed with VMI in mind. For this reason, implementing a hardware-supported VMI system makes it difficult to trace and observe VM activity from the host OS. With our proposed framework, we will leverage system calls to overcome this challenge.

The remaining portion of this pre-proposal will describe the implementation goals of our VM system call tracing and monitoring framework.

Proposed Implementations:

Implement Intel VT-x KVM VM System Call Tracing

Within KVM, guest instructions are safely run directly on the physical CPU. Guest instructions continue to run on the physical hardware until it encounters an instruction that the hypervisor needs to emulate. As KVM VM system call instructions are run directly on the physical CPU, it is not possible by default to monitor system calls from the host OS due to the isolation between the host and guest.

To observe system calls from the guest VM, we will force every system call instruction to result in a VM-exit. A VM-exit is a transition from the VM to the hypervisor. To achieve this, we will unset the system call enable (SCE) bit of the guest VMs Extended Feature Enable Register (EFER), which is a Model Specific Register

(MSR). Unsetting this bit results in system call instructions being unknown to the physical CPU. As a consequence, when system call instructions are executed in a VM, an invalid opcode exception (#UD) is generated that induces a VM-exit with exit reason 0. From this point, we will use a built-in Linux kernel framwork named Extended Berkeley Packet Filter (eBPF) to observe VM-exits from the host. The VMs RIP register can be used to verify that VM-exits with reason 0 was due to a system call instruction. For example, if the RIP register is pointing to a value of 0x0F05, then the VM-exit with reason 0 was due to a system call.

As unsetting the SCE bit results in system call instructions to be unknown to the CPU, we will need to explictly emulate every system call instruction in the hypervisor before making an entry back into the VM.

Proposed Algorithm for System Call Emulation in the Hypervisor:

---

1. Disable SCE bit from the EFER MSR.
2. Every System call instruction implicitly results in VM-exit with Reason 0.
3. Explictly emulate system call in KVM.
4. Perform VM-entry implicitly.

Implement a Linux Kernel Tracepoint for Guest System Call Tracing

A tracepoint is a piece of code within the Linux kernel that can be hooked to a function for runtime introspection. eBPF allows users to trace guest system calls by using predefined Linux kernel tracepoints that track VM-exits. However, aside from informing us of a guest system call, this predefined tracepoint does not give us adequate information about each system call. For this reason, our goal is to implement a custom Linux kernel tracepoint by hooking a function that can provide us with sufficient information about guest system call activity. We will then utilize eBPF in conjunction with our custom tracepoint to automatically trace and monitor KVM VM system calls from the host.

Implement Filtration of System Calls by Virtual Machine and Process

KVM has the ability to operate more than one virtual machine concurrently, in an isolated environment, and on a single operating system. Therefore, every system call from every virtual machine will cause a VM-exit to the same hypervisor. At the time of writing, eBPF does not have the ability to filter VM-exits by specific running virtual machines. eBPF also does not have the ability to filter VM-exits by the guest process that called it. In our framework, we intend to implement a filter that will (1) allow us to know which system call corresponds to which virtual machine, and (2) filter system calls by guest processess that invoked the system call.

Respond to Anomalous System Calls with pH

After KVM system calls are observable from the host OS via eBPF, our goal is to use pH (Process Homeostasis) within our framework. pH is software that monitors process states in the form of system call sequences [2]. pH detects changes in program behavior by observing changes in short sequences of system calls [2]. When pH determines that a process is behaving unusually, it responds by slowing down that process's system calls [2].

.

Bi-weekly Schedule

Fall 2022	**Proposed Commitement**
Week 1	Begin research on related work and find the best approach for KVM VM system call introspection from the host
Week 3	Implement Intel VT-x KVM VM system call Tracing
Week 5	Implement a custom Linux kernel tracepoint for guest system call tracing
Week 7	Implement filtration of system calls by virtual machine and process
Week 9	Respond to anomalous system calls with pH
Week 11	Write up thesis proposal report
Week 13	Write up thesis proposal report

References

[1] Findlay W. Host-Based Anomaly Detection With Extended BPF. https://www.cisl.carleton.ca/ will/written/coursework/undergrad-ebpH-thesis.pdf. April 17, 2020.

[2] Somayaji A. Operating System Stability and Security through Process Homeostasis. https://people.scs.carleton.ca/ soma/pubs/soma-diss.pdf. July, 2002.