RootView

a KVM-based eBPF malware detection engine

Luis Abraham - labrahamesco2024@my.fit.edu
Dylin Irons - dirons2024@my.fit.edu
Dominick Morales - dmorales2024@my.fit.edu
Braiden Ames - bames2024@my.fit.edu

Dr. Ribeiro - eribeiro@fit.edu

Client: Amy Alvarez, NASA Scientist

Meeting with the advisor once a month during his office hours TR 2-3:15pm. Meeting with the client once a month when available.

Our goal in developing RootView is to develop a platform for observing and eventually detecting malicious activity within the eBPF subsystem of the Linux kernel. RootView will use KVM-based virtualization and Virtual Machine Introspection (VMI) to inspect a Linux guest from outside of the guest operating system. Our motivation is that the existing tooling for eBPF malware detection would be running on an infected kernel, so we wanted to make something that provides a new perspective from outside of the system that allows the user to see what truly is occurring within a VM.

To begin, eBPF is a technology that “can run sandboxed programs in a privileged context such as the operating system kernel”. BPF is widely used for networking, monitoring, performance analysis, and security, but its ability to execute code within the kernel also makes it an attractive target for attackers. Recent research has demonstrated eBPF-based rootkits, but there is limited tooling specifically designed to analyze and detect this type of malware.

RootView will use the hypervisor as an external observation point. This allows users to inspect guest memory and kernel state without relying on software running inside the potentially compromised guest. In the first semester of our project, we will focus on building the VMI and eBPF observation infrastructure, which will serve as the foundation for the malware detection engine in the following semester.

We will also provide tools for creating and managing Linux virtual machines using QEMU and KVM. Once a VM is running, LibVMI will allow RootView to inspect it externally. The initial capabilities will include reading physical and virtual memory, address translation, processor state, and basic Linux processes and kernel structures.

RootView will provide higher-level information about the Linux kernel and eBPF subsystem instead of only displaying raw memory. The goal is that RootView will investigate kernel profiles, BTF/DWARF information, kernel symbols, and do page-table walking to interpret kernel structures across different Linux versions.

For eBPF, RootView will inspect objects such as BPF programs, maps, and attachments. This will give users an external view of eBPF activity within the guest. We will develop an eBPF event collector that converts relevant kernel activity into a standardized format. The initial goal is to establish telemetry rather than implement every possible rootkit detection technique.

The eventual detection engine will combine eBPF telemetry with VMI observations. For example, eBPF can report that a program was loaded while VMI independently examines the corresponding kernel state. Comparing these sources may help identify suspicious behavior or manipulation.

Our final milestone is to provide a Python API over the C/C++ backend. Users will be able to manage VMs and perform memory, Linux, and eBPF introspection without directly interacting with LibVMI or KVM. This API will also provide the interface for the detection engine developed next semester.

The primary novel aspect of RootView is its focus on eBPF-based malware and rootkit detection using external VMI. eBPF rootkits are a somewhat recent development, and current tooling operates inside the system being monitored, while RootView provides an outside view through the hypervisor. This means that while other tools inside a running VM might not detect malware as a result of it going to great lengths to hide itself from outside programs, RootView can tell that something is wrong.

Through combining VMI and eBPF telemetry, we provide two different perspectives of the same system. Comparing these perspectives may reveal activity that would be difficult to identify using only traditional in-guest monitoring.

Algorithms and tools:

- KVM/QEMU for virtualization
- LibVMI for virtual machine introspection
- Page-table walking for address translation
- BTF/DWARF and kernel profiles for Linux structures
- eBPF for kernel telemetry
- C/C++ for the VMI backend
- Python for the research API and future detection engine
- Web framework for the user interface

Technical Challenges:

1. KVM and VMI: We have limited experience with KVM and VMI, and must learn how to access guest memory, processor state, and page tables from outside the guest.
2. Linux Kernel Introspection: Kernel structures change between versions, so RootView must investigate ways to identify and interpret structures without relying entirely on hardcoded offsets.
3. eBPF Malware: We must learn how eBPF works internally, how it can be abused by rootkits, and what characteristics of malicious activity can be observed.

The first milestone will establish the basic VMI infrastructure.
Tasks:

- Select KVM/QEMU, LibVMI, web framework, and initial eBPF tools
- Create simple tooling for setting up an Ubuntu VM
- Connect to a running guest through LibVMI
- Read physical and virtual guest memory
- Investigate address translation
- Retrieve basic register state
- Begin process introspection
- Serve a basic test page
- Complete the Requirements Document
- Complete the Design Document
- Complete the Test Plan

The second milestone will expand RootView into a Linux and eBPF introspection platform.
Tasks:

- Linux kernel introspection
- BTF/DWARF and kernel symbol investigation
- Process and kernel object enumeration
- eBPF introspection
- Integration of VMI and eBPF observations
- Web interface improvements
- Initial kernel-version testing

The third milestone will turn the backend into a reusable research platform.
Tasks:

- C/C++ API for VM, OS, memory, and eBPF introspection
- Python bindings
- High-level Python interface

- Error handling and testing
- Linux kernel version compatibility
- Kernel profile improvements
- API documentation
- Stable interface for the future detection engine

Task	**Luis**	**Braiden**	**Dominick**	**Dylin**
Develop tooling for easy VM setup (CLI, JSON configs, integration with introspection tooling, and more)	Write 10%	Write 45%	Write 45%	Write 0%
Develop web server for users	Write 0%	Write 0%	Write 0%	Write 100%
Write LibVMI introspection back end	Write 100%	Write 0%	Write 0%	Write 0%
Requirement Document	Write 25%	Write 25%	Write 25%	Write 25%
Design Document	Write 25%	Write 25%	Write 25%	Write 25%
Test Plan	Write 25%	Write 25%	Write 25%	Write 25%