Transparent Intrusion Detection in Xen Virtual Machines

TERESE Niels
INRIA and KrakOS
Grenoble, France
niels.terese@etu.univ-grenoble-alpes.fr

Supervised by: Baptiste Lepers

I understand what plagiarism entails and I declare that this report is my own, original work.
Name, date and signature: TERESE Niels 24/03/2025

1 Introduction

Intrusion detection is an essential component of securing virtualized environments, particularly in Xen-based systems, where multiple virtual machines (VMs) share a single physical host. In such systems, using tools like libVMI to detect potential intrusions in real-time requires continuous monitoring of VM states and inspecting memory for malicious activities.

While this monitoring approach is effective in enhancing security, it can introduce significant overheads that impact the overall system performance.

I conducted a series of experiments on the Grid5000 cluster, evaluating the performance overheads of libVMI-based intrusion detection under various workloads.

2 Current Status

2.1 Used Infrastructure

All benchmarks and experiments were conducted on the Grid5000 Cluster; more specifically, on the Dahu Grenoble's Cluster.

This section details the experimental environment used, including the benchmark tools and configurations used throughout the study.

Dahu Hardware

The Dahu cluster is composed of 32 nodes. Each one of them have the following hardware :

- Model : Dell PowerEdge C6420
- CPU : Intel Xeon Gold 6130 (Skylake-SP), x86_64, 2.10GHz, 2 CPUs/node, 16 cores/CPU
- RAM : 192 GiB
- Network : 10 Gbps, model: Intel Ethernet Controller X710 for 10GbE SFP+, driver: i40e

Environment setup

To work with Xen on Grid5000, I used the node (host) reservation in deploy mode. This allowed me to deploy a preexisting Debian 12 NFS image. Afterward, I installed Xen on the image and configured it to boot into Xen. The image I generated avoids the need to recompile everything, saving a significant amount of time. Xen uses a modified Debian as the Dom0, with all the Xen toolstack installed.

VM Setup

I then created an Ubuntu VM inside Xen for my tests, and the VM is stored in my home directory in Grenoble to ensure that the installation is preserved. Ubuntu is installed using the default configuration, with a OpenSSH server.

VM Specifications

- System : Ubuntu
- 8 vCPUs
- RAM : 16 GiB
- Network MAC : 00:16:3E:84:00:02
- Disk : 30 Go

2.2 Benchmarks

To evaluate performance, I used the following tools :

- Phoronix Test Suite (v10.8.4) to benchmark CPU performance
- Perf to measure low-level performance metrics, including CPU cycles and system calls
- htop
- xl top

Phoronix benchmarks were always executed within the Ubuntu VM, while Perf was used both inside the VM and occasionally on Dom0.

Each of the following benchmarks was initially run without libVMI introspection and then rerun with libVMI introspection enabled. To perform the introspection, I used a script running in dom0 that executes the command ”vmi-process-list -n basevm” every second.

Phoronix Results

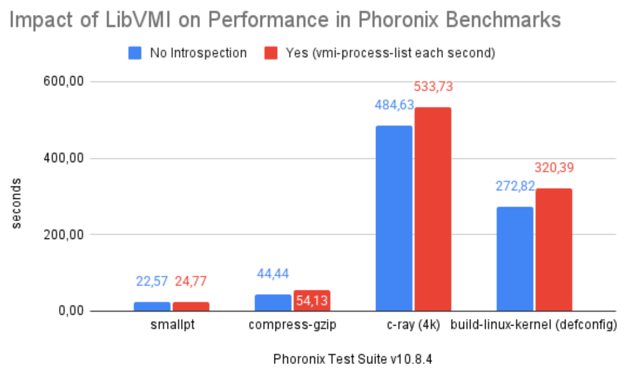

Figure 1: Enter Caption

I observed an average performance loss of 14,77% with introspection enabled.

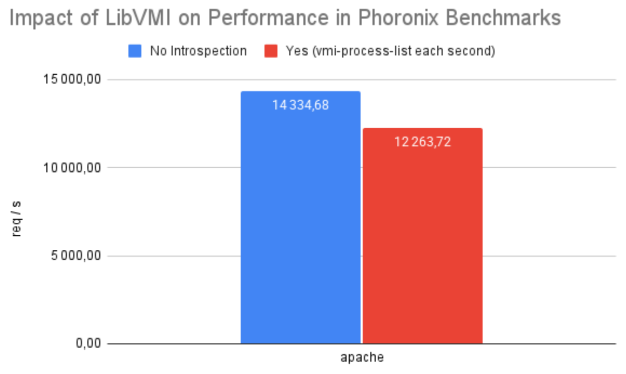

Figure 2: Enter Caption

For Apache, the performance loss was 14,44%.

Inside the VM
For more precise measurements, I used perf. However, the tricky part is that I could only measure a 2,40% performance loss. This can be explained by the fact that when libvmi lists the processes, it pauses the VM, and consequently, perf as well.

We can deduce that the 2,40% performance loss is due to the VM's overhead while "unpausing" it.

So, why does not Phoronix capture this pause? It is because Phoronix uses real-time measurements for benchmarking, which does not account for the paused state of the VM.

From dom0
From dom0, I ran htop to check if I could observe CPU activity during a benchmark running (smallpt from Phoronix) in the VM. The result was that there were no noticeable differences.

To detect CPU activity more accurately, I needed to use "xl top", which performs a hypercall to obtain real CPU data.

When the benchmark runs without libVMI, all vCPUs of the vm are running at full capacity (800% utilization due to 8 vCPUs), while dom0 is at 1,8%.

However, when running the same benchmark with the libVMI loop, the vCPUs are running at 740%, and the utilization of dom0 increases to around 39%.

3 Next Steps

3.1 Perf on dom0: Capturing Real Data ?

I discussed this issue with Andrew Cooper and @andriy, and they pointed out that Xen uses performance counters as a host watchdog by default. As a result, using perf counters inside the guest requires disabling this feature.

According to Andrew Cooper: "Xen uses perf counters as a host watchdog by default. In order to use perf counters in a guest, you need to turn this off."

To disable the watchdog, I set watchdog=0 in the GRUB configuration. However, I have not yet been able to get it to work. Further experimentation is needed to determine a functional setup.

3.2 Adapting Dufy Teguia's KVM Process Detection for Xen

The next step is to get Dufy's code, ensuring that I understand any dependencies or hacks required for it to work. Once I have it, I will adapt and test it on Xen, and then compare the results between KVM and Xen to analyze the differences.

4 Context : libVMI

Virtual Machine Introspection (VMI) is a technique that enables external monitoring and analysis of a virtual machine (VM) from the hypervisor level. Unlike traditional monitoring tools that run inside the guest system, VMI provides a way to inspect a VM's state without modifying or interfering with its internal execution.

LibVMI is an open-source library that facilitates VMI by providing access to VM memory, process lists, and other system-level observables. However, to correctly interpret kernel structures and symbols, libVMI requires additional setup. First, the /boot/System.map file from the VM must be copied to the host. This file contains symbol addresses for the guest kernel, but it does not provide direct memory offsets needed for introspection. To obtain the correct offsets, a dedicated kernel module (findoffsets.ko) must be compiled and loaded within the VM. This module extracts essential memory offsets and logs them in /var/log/syslog, allowing libVMI to correctly map kernel structures.

The main advantage of using VMI from outside the VM, rather than relying on in-guest monitoring tools, is enhanced security and reliability. Internal monitoring tools can be compromised by rootkits or advanced malware, whereas external introspection remains isolated from the guest. Additionally, since VMI runs from the hypervisor (dom0 in Xen), it avoids adding overhead inside the VM itself. However, this method introduces new challenges, such as measuring the performance impact of introspection and ensuring minimal disruption to the observed system.

5 Writing Guidelines

5.1 Headings and Sections

When necessary, headings should be used to separate major sections of your paper. (These instructions use many headings to demonstrate their appearance; your paper should have fewer headings.)

Section Headings

Print section headings in 12-point bold type in the style shown in these instructions. Leave a blank space of approximately 10 points above and 4 points below section headings. Number sections with arabic numerals.

Subsection Headings

Print subsection headings in 11-point bold type. Leave a blank space of approximately 8 points above and 3 points below subsection headings. Number subsections with the section number and the subsection number (in arabic numerals) separated by a period.

Subsubsection Headings

Print subsubsection headings in 10-point bold type. Leave a blank space of approximately 6 points above subsubsection headings. Do not number subsubsections.

Special Sections

You may include an unnumbered acknowledgments section, including acknowledgments of help from colleagues, financial support, and permission to publish.

Any appendices directly follow the text and look like sections, except that they are numbered with capital letters instead of arabic numerals.

The references section is headed "References," printed in the same style as a section heading but without a number. A sample list of references is given at the end of these instructions. Use a consistent format for references, such as that provided by BibTEX. The reference list should not include unpublished work.

5.2 Citations

Citations within the text should include the author's last name and the year of publication, for example [Gottlob, 1992]. Append lowercase letters to the year in cases of ambiguity. Treat multiple authors as in the following examples: [Abelson *et al.*, 1985] or [Baumgartner *et al.*, 2001] (for more than two authors) and [Brachman and Schmolze, 1985] (for two authors). If the author portion of a citation is obvious, omit it, e.g., Nebel [2000]. Collapse multiple citations as follows: [Gottlob *et al.*, 2002; Levesque, 1984a].

5.3 Footnotes

Place footnotes at the bottom of the page in a 9-point font. Refer to them with superscript numbers.[1] Separate them from the text by a short line.[2] Avoid footnotes as much as possible; they interrupt the flow of the text.

[1]This is how your footnotes should appear.

[2]Note the line separating these footnotes from the text.

6 Illustrations

Place all illustrations (figures, drawings, tables, and photographs) throughout the paper at the places where they are first discussed, rather than at the end of the paper. If placed at the bottom or top of a page, illustrations may run across both columns.

Illustrations must be rendered electronically or scanned and placed directly in your document. All illustrations should be in black and white, as color illustrations may cause problems. Line weights should be 1/2-point or thicker. Avoid screens and superimposing type on patterns as these effects may not reproduce well.

Number illustrations sequentially. Use references of the following form: Figure 1, Table 2, etc. Place illustration numbers and captions under illustrations. Leave a margin of 1/4-inch around the area covered by the illustration and caption. Use 9-point type for captions, labels, and other text in illustrations.

Acknowledgments

The preparation of these instructions and the LATEX and BibTEX files that implement them was supported by Schlumberger Palo Alto Research, AT&T Bell Laboratories, and Morgan Kaufmann Publishers. Preparation of the Microsoft Word file was supported by IJCAI. An early version of this document was created by Shirley Jowell and Peter F. Patel-Schneider. It was subsequently modified by Jennifer Ballentine and Thomas Dean, Bernhard Nebel, and Daniel Pagenstecher. These instructions are the same as the ones for IJCAI–05, prepared by Kurt Steinkraus, Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Lab.

A LATEX and Word Style Files

The LATEX and Word style files are available on the IJCAI–11 website, `http://www.ijcai-11.org/`. These style files implement the formatting instructions in this document.

The LATEX files are `ijcai11.sty` and `ijcai11.tex`, and the BibTEX files are `named.bst` and `ijcai11.bib`. The LATEX style file is for version 2e of LATEX, and the BibTEX style file is for version 0.99c of BibTEX (*not* version 0.98i). The `ijcai11.sty` file is the same as the `ijcai07.sty` file used for IJCAI–07.

The Microsoft Word style file consists of a single file, `ijcai11.doc`. This template is the same as the one used for IJCAI–07.

These Microsoft Word and LATEX files contain the source of the present document and may serve as a formatting sample.

Further information on using these styles for the preparation of papers for IJCAI–11 can be obtained by contacting `pcchair11@ijcai.org`.

References

[Abelson *et al.*, 1985] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Massachusetts, 1985.

[Baumgartner *et al.*, 2001] Robert Baumgartner, Georg Gottlob, and Sergio Flesca. Visual information extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Databases*, pages 119–128, Rome, Italy, September 2001. Morgan Kaufmann.

[Brachman and Schmolze, 1985] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April–June 1985.

[Gottlob *et al.*, 2002] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002.

[Gottlob, 1992] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, June 1992.

[Levesque, 1984a] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23(2):155–212, July 1984.

[Levesque, 1984b] Hector J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 198–202, Austin, Texas, August 1984. American Association for Artificial Intelligence.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.