Transparent Intrusion Detection in Xen Virtual Machines using libVMI: Performance and Limitations

Terese Niels
Supervised by Lepers Baptiste

Introduction to Virtualization

Virtualization :

- Containers : process-level isolation, shared with host kernel
- VMs : OS isolation, run multiple OS on the same server

Advantages :

- efficiency -> multiple services on a single server
- isolation -> improved security
- easier maintenance : pause, clone, migrate systems
- snapshot and rollback

KrakOS

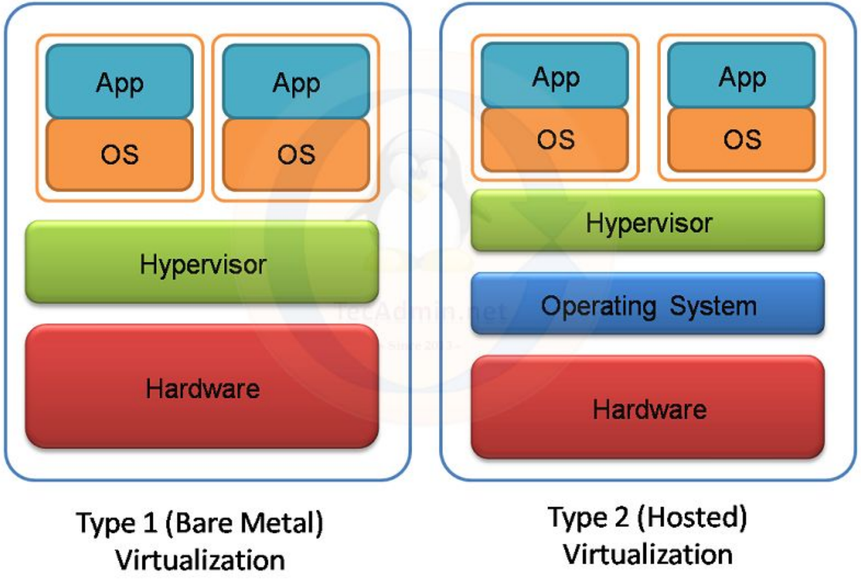
Managing VMs

Hypervisor :

- type1: KVM, Xen, Qemu, Proxmox…
- type2 : VirtualBox, VmWare Workstation

Hypercalls :

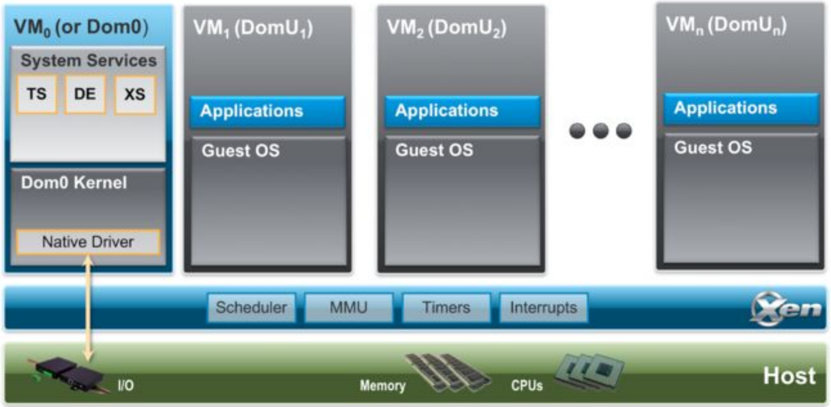
- like system calls, between VMs and hypervisor
- VM control (I/O, CPU sched, memory management)



source : https://tecadmin.net/type-1-vs-type-2-virtualization/

Xen Architecture

- Xen is a type-1 (bare-metal) hypervisor:
  - Runs directly on hardware
  - Manages CPU, memory, and I/O for virtual machines
- Components:
  - **Dom0**: Privileged domain with access to hardware and control tools
  - **DomU**: Unprivileged guest VMs

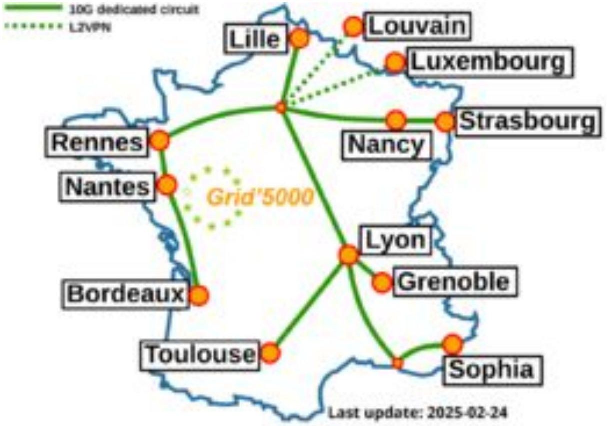


source : https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview

Grid5000 testbed

Experimental Testbed for Research

- Enables fast and reproducible environment creation
- Realistic setup: literally a cluster like in production



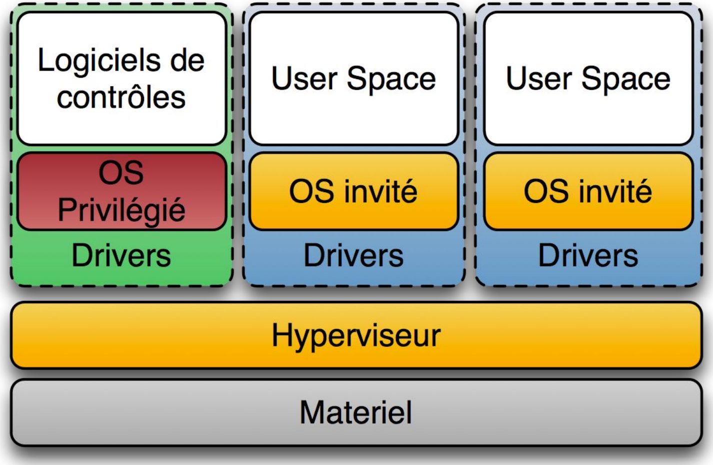
Introduction to VMI

Architecture : Server => Hypervisor + Vms

No agents in guest => External observation

Used for IDS / malware analysis

Trade-off : security ⇔ performance overhead



source : https://fr.wikipedia.org/wiki/Hyperviseur

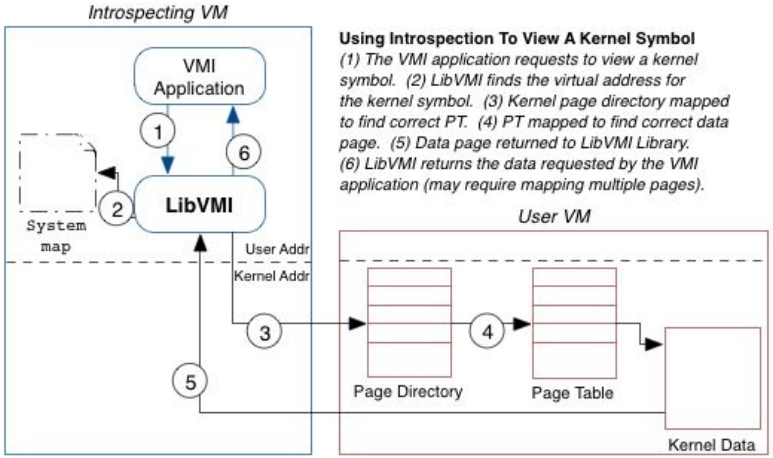
Memory introspection

libVMI - open source lib
Compatible with Xen, KVM, Qemu

VMs support : Windows / Linux
32/64 bits, ARM

2 snapshot modes : live snap / halt snap

Requires : /boot/System.map (from VM), and the offsets (kernel module => findoffets.so)



Using Introspection To View A Kernel Symbol
(1) The VMI application requests to view a kernel symbol. (2) LibVMI finds the virtual address for the kernel symbol. (3) Kernel page directory mapped to find correct PT. (4) PT mapped to find correct data page. (5) Data page returned to LibVMI Library. (6) LibVMI returns the data requested by the VMI application (may require mapping multiple pages).

source : https://libvmi.com/docs/gcode-intro.html

Benchmark tools & methodology

Tools :

- Phoronix Test Suite
- perf
- htop & xl top
- xentrace & xenalyze => custom parser + sync script

For every bench of Phoronix :

- with libVMI
- without libVMI

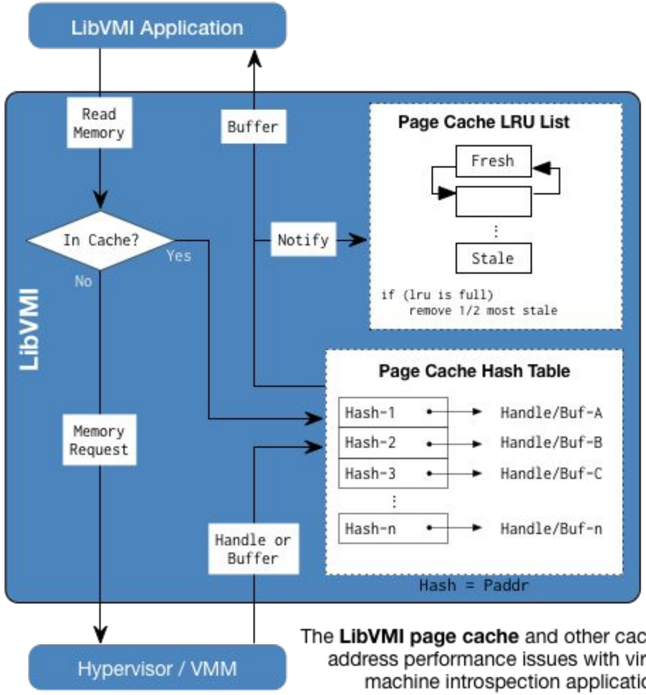
Init Setup & Cache

Init setup :

- vmi-process-list
- halt mode, external loop

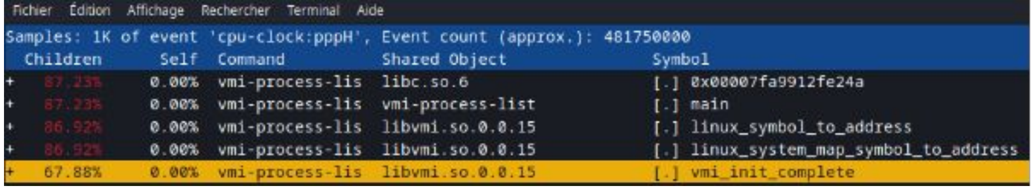
Cache :

- Least Recently Used
- Purpose: Speeds up memory access
- Method: Caches to avoid repeated physical reads
- Improves memory analysis efficiency
- Validity: Only valid when VM is running or paused



The **LibVMI page cache** and other caches address performance issues with virtual machine introspection applications.

Optimization



```
Samples: 1K of event 'cpu-clock:pppH', Event count (approx.): 481750000
  Children      Self  Command            Shared Object             Symbol
+   87.23%     0.00%  vmi-process-lis  libc.so.6                 [.] 0x00007fa9912fe24a
+   87.23%     0.00%  vmi-process-lis  vmi-process-list          [.] main
+   86.92%     0.00%  vmi-process-lis  libvmi.so.0.0.15          [.] linux_symbol_to_address
+   86.92%     0.00%  vmi-process-lis  libvmi.so.0.0.15          [.] linux_system_map_symbol_to_address
+   67.88%     0.00%  vmi-process-lis  libvmi.so.0.0.15          [.] vmi_init_complete
```

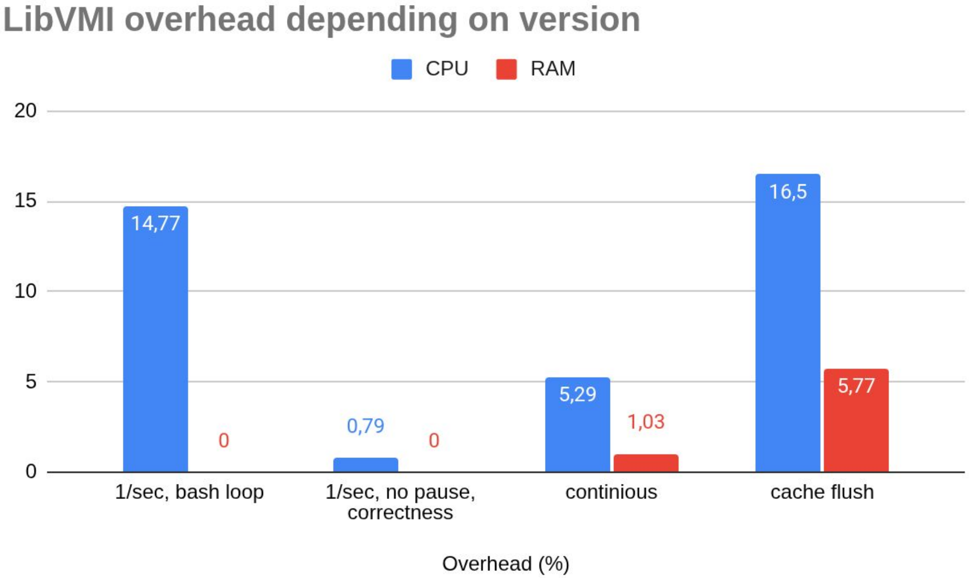
1. init overhead removed
   - overhead found with perf (dom0)
   - 0,30s in vmi_init_complete (clock time)
2. disabled VM pausing
3. improved robustness
   - max 5 iterations
4. enable continuous introspection
5. added page cache flushing
   - for reboot / VM delition
   - cache becomes obsolete, causing errors or inconsistencies

KrakOS

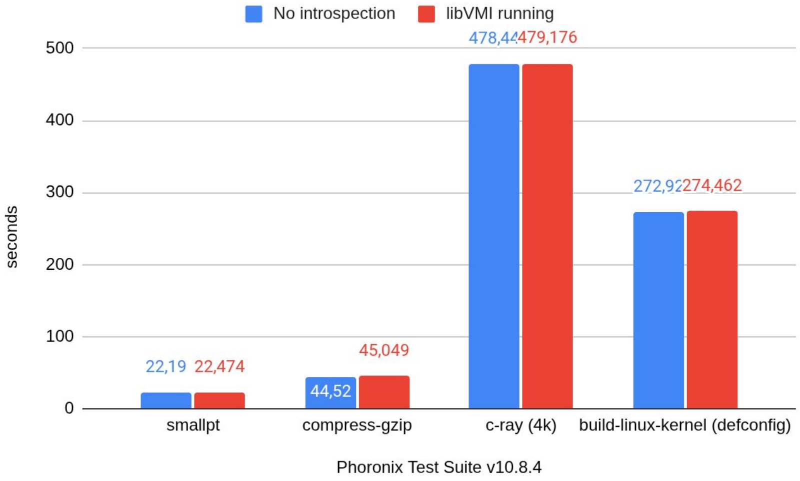
Performance impact in DomU

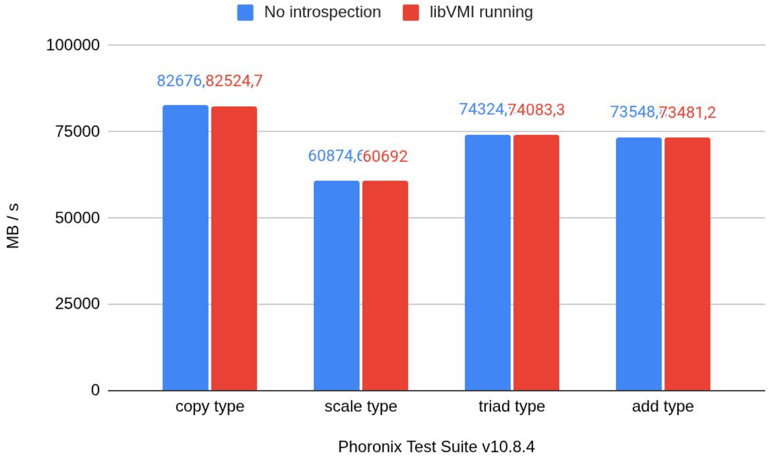
Benchmarks used :

- smallpt
- compress-gzip
- c-ray (4k)
- build-linux-kernel (defconfig)
- apache (200), pts/stream



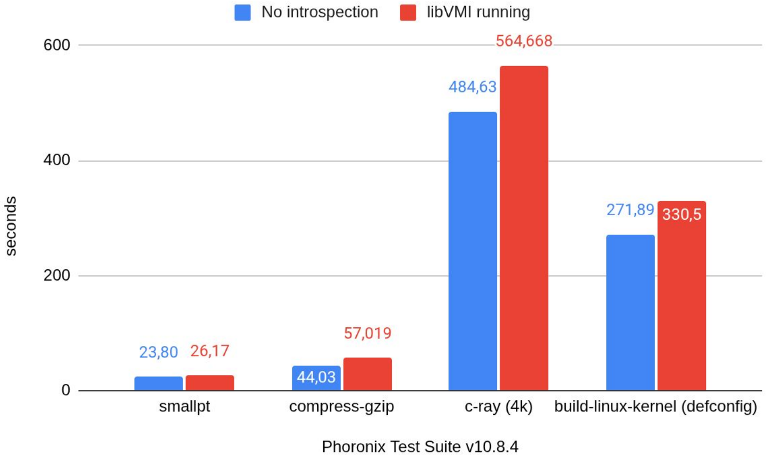
Most Optimized detailed results

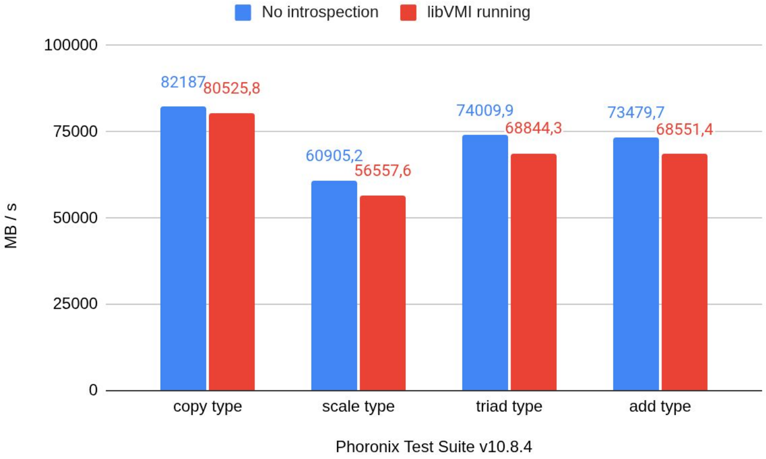




KrakOS

Most accurate detailed results





Profiling limitations

DomU profiling :

- limited (perf) => paused during introspection with “initial version”
- 2.40% overhead (with init version) => recovery cost after pause

Dom0 profiling :

- htop : no accurate data

Hypervisor profiling :

- xl top
- libVMI : dom0 39% usage, domU 740% CPU usage
- no libVMI : dom0 1.8%, domU 800%

Xentrace & Xenalyze

Tracing :

- xentrace -> raw binary file
- xenalyze : readable trace “summary”
- custom python parser : extract and generate stats

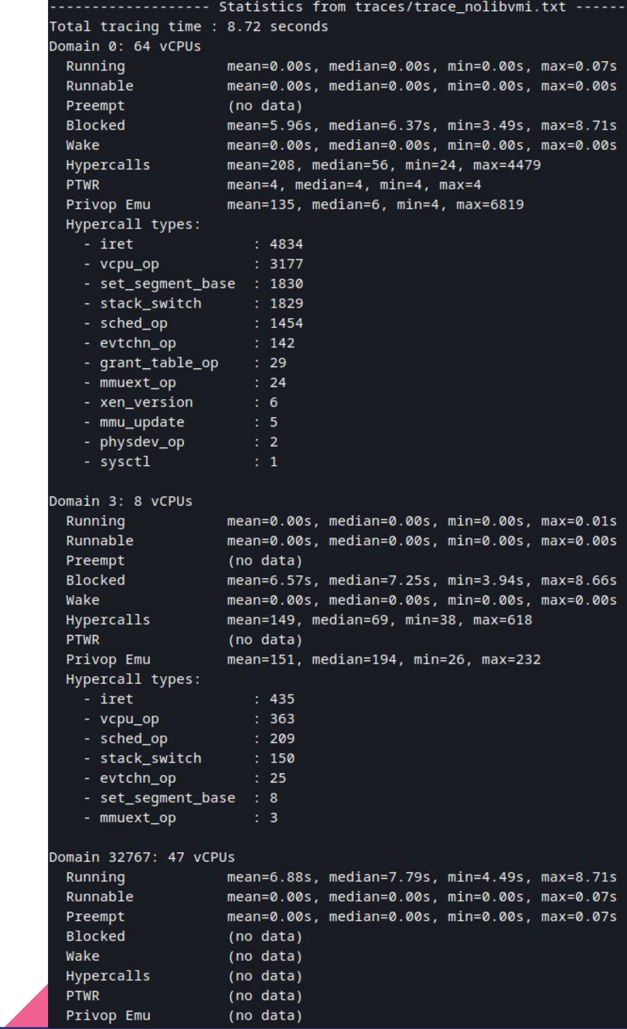
Methods :

- VM idle
- libVMI only
- Bench only
- libVMI + bench

Parser output

Findings :

- compress-gzip bench : 30% overhead
- libVMI with cache flush
- domU : hypercalls +15%
- dom0 : increased significantly
  (iret, stack_switch, mmu_update, vcpu_op)
- stable VM CPU time



```
------------------ Statistics from traces/trace_nolibvmi.txt ------
Total tracing time : 8.72 seconds
Domain 0: 64 vCPUs
  Running              mean=0.00s, median=0.00s, min=0.00s, max=0.07s
  Runnable             mean=0.00s, median=0.00s, min=0.00s, max=0.00s
  Preempt              (no data)
  Blocked              mean=5.96s, median=6.37s, min=3.49s, max=8.71s
  Wake                 mean=0.00s, median=0.00s, min=0.00s, max=0.00s
  Hypercalls           mean=208, median=56, min=24, max=4479
  PTWR                 mean=4, median=4, min=4, max=4
  Privop Emu           mean=135, median=6, min=4, max=6819
  Hypercall types:
    - iret               : 4834
    - vcpu_op            : 3177
    - set_segment_base   : 1830
    - stack_switch       : 1829
    - sched_op           : 1454
    - evtchn_op          : 142
    - grant_table_op     : 29
    - mmuext_op          : 24
    - xen_version        : 6
    - mmu_update         : 5
    - physdev_op         : 2
    - sysctl             : 1

Domain 3: 8 vCPUs
  Running              mean=0.00s, median=0.00s, min=0.00s, max=0.01s
  Runnable             mean=0.00s, median=0.00s, min=0.00s, max=0.00s
  Preempt              (no data)
  Blocked              mean=6.57s, median=7.25s, min=3.94s, max=8.66s
  Wake                 mean=0.00s, median=0.00s, min=0.00s, max=0.00s
  Hypercalls           mean=149, median=69, min=38, max=618
  PTWR                 (no data)
  Privop Emu           mean=151, median=194, min=26, max=232
  Hypercall types:
    - iret               : 435
    - vcpu_op            : 363
    - sched_op           : 209
    - stack_switch       : 150
    - evtchn_op          : 25
    - set_segment_base   : 8
    - mmuext_op          : 3

Domain 32767: 47 vCPUs
  Running              mean=6.88s, median=7.79s, min=4.49s, max=8.71s
  Runnable             mean=0.00s, median=0.00s, min=0.00s, max=0.07s
  Preempt              mean=0.00s, median=0.00s, min=0.00s, max=0.07s
  Blocked              (no data)
  Wake                 (no data)
  Hypercalls           (no data)
  PTWR                 (no data)
  Privop Emu           (no data)
```

Conclusion

- libVMI enables monitoring, but at a cost
- removing VM pause reduces performance overhead
- cache flushing ensures accuracy, but reintroduces overhead
- traditional tools miss indirect effects of introspection : hypervisor-level tools required

Future work :

- still a “black box”
- hard to get useful data from Xen
- adapt introspection frequency
- adapt cache flush frequency

KrakOS