Bitdefender®

ADVANCED VMI ON KVM: A PROGRESS REPORT

Mihai Donțu, Engineering Manager

1 November 2019

Agenda

- Why VMI
- What is VMI
- Advantages of VMI
- Typical use cases
- How we use it
- KVM work
- A word on AMD support
- QEMU work
- How we test it
- Performance numbers
- Related projects

Why VMI

Critical issues (kernel zero-days) and state sponsored attacks (via APTs) have created urgency for a different approach to software security

Why VMI

- Modern kernels are very complex
- Same for user software (e.g., browsers)
- Bugs can lead to total system compromise
- Hardening is also a complex process

What is VMI

A method of inspecting the state of a guest VM and determining:

- The type of OS is running (Linux, Windows etc.)
- What user applications are running
- Is there *potentially harmful* code being executed

All this without the use of in-guest tools

Advantages of VMI

For security applications, VMI:

- Offers better isolation
- Removes the reliance on the guest OS to function
- Minimizes (if any) interference with the guest OS

Typical use cases

- Inspect VM memory for violations (buffer overflows, code injection, etc.)
- Guarantee integrity of in-guest security
- Drivers, other kernel components
- Monitor both kernel and user-mode processes

How we use it

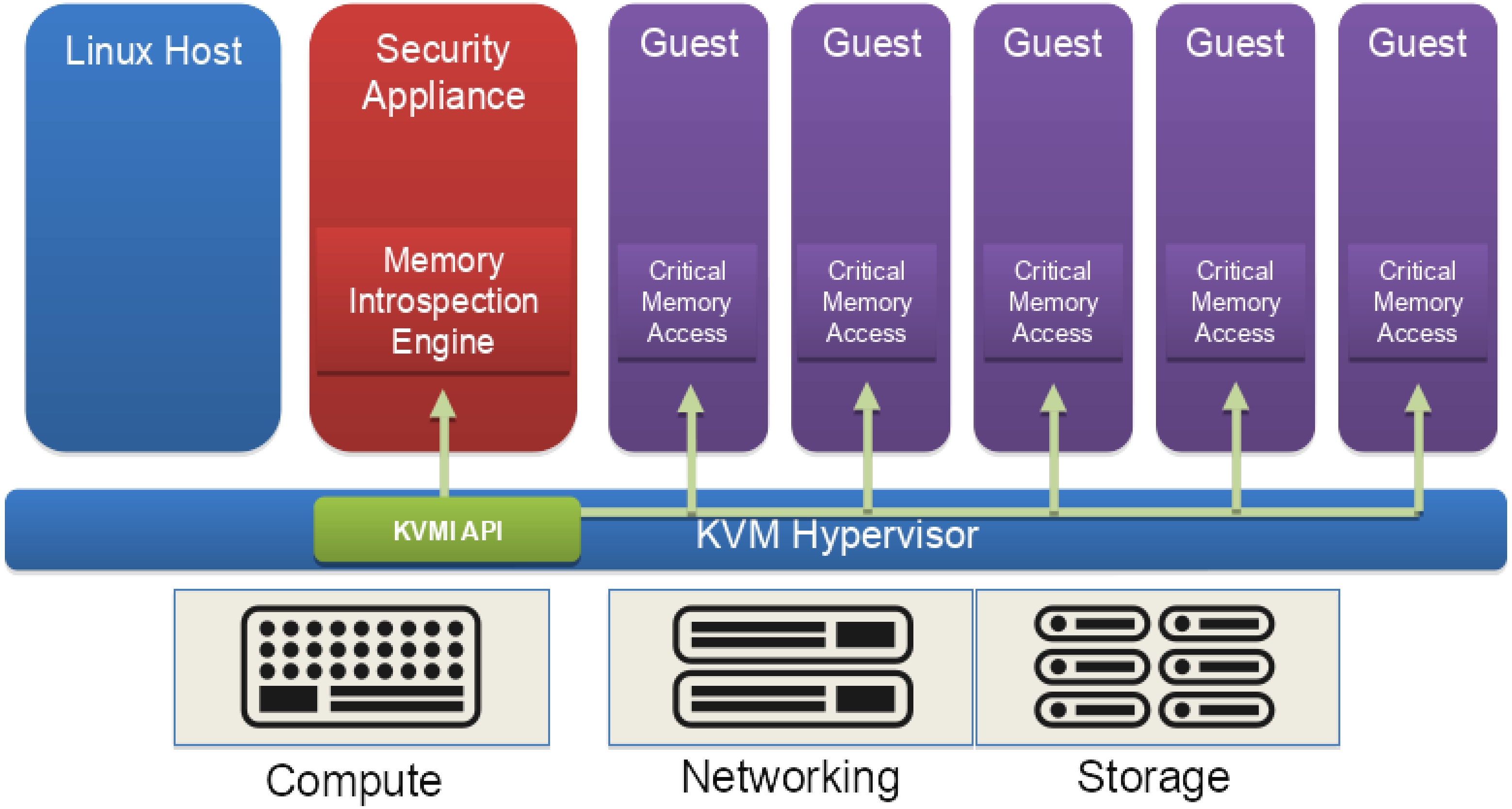
With the help of VMI and specifically EPT we:

- Secure the OS kernel
- Enforce the access restrictions to code, data, stack, heap etc.
- Secure IDT, GDT, SSDT, HDT, system CR3, tokens etc.
- Secure driver objects
- Enforce hardware features (CR4.SMEP and CR4.SMAP)
- Secure the kernel syscall entry point

Secure the user applications (e.g., browsers)

- Enforce access restrictions to code, data, stack, heap etc.
- Prevent code injections
- Prevent hooks (overriding DLL calls, eg. WinSock API)
- Immediately terminate applications in which an exploit has been launched (via ROP or other method evading the memory access restrictions)

Architecture overview



KVM work

Still RFC

Published 6 patch series:

- v1: June 16, 2017
- v2: July 7, 2017
- v3: September 11, 2017
- v4: December 18, 2017
- v5: December 20, 2018
- v6: August 9, 2019

KVM work (cont.)

Most difficult tasks:

- Control SPT (EPT / NPT) permissions → use page tracking
- Work around emulator limitations → use single stepping
- Inter-guest page sharing (aka remote mapping) → mimic KSM but without KSM and THP → still being worked on
- Control / event channel and protocol → BSD sockets on vhost-vsock
- Exception injection

KVM work (cont.)

Control SPT (EPT / NPT)

Extended Intel’s page tracking code with 4 new notifier callbacks:

- pre-read
- pre-write
- pre-exec
- create slot

KVM work (cont.)

Emulator limitations

Use Intel's MONITOR TRAP FLAG (MTF) to single-step instructions for which KVM's x86 emulator fails (SSE, AVXn etc.)

For SPT faults caused by page table walks:

```
gpa = kvm_mmu_gva_to_gpa_system(vcpu, gva, PFERR_WRITE_MASK, NULL);
if (gpa == UNMAPPED_GVA)
     gpa = kvm_mmu_gva_to_gpa_system(vcpu, gva, 0, &exception);
If (gpa != UNMAPPED_GVA)
     /* return to guest */
```

KVM work (cont.)

Page sharing / remote mapping

- Experimented with several approaches
- Could not reconcile KSM and THP

Working on another approach with ideas from Jerome Glisse.

KVM work (cont.)

Control / event channel

- QEMU connects to security application (introspector)
- After handshake, socket descriptor is passed to the host kernel
- Kernel handles events (eg. MSR write) and control requests (eg. get registers)
- Requests targeting a vCPU require access to a valid VMCS.
- Each vCPU thread handles requests

KVM work (cont.)

Exception injection

- Queue exceptions (e.g., page fault) after KVM's
- Abort if KVM has already programmed an exception or interrupt
- Notify security application; Try again at a later time

KVM work (cont.)

Additional work that will be published

- VMFUNC and #VE
- SPP (Sub-Page Protection)

Beta quality

Most KVMI work is done in-house. Will move to a public repository in 2020

A word on AMD support

There are several show-stoppers on AMD:

- NPT treats all guest page table walks as write faults (like EPT with A/D tracking on)
- NPT does not support execute-only
- No equivalent to VMX’ MTF (native debugging support *might* be a workaround)

QEMU work

Patches on github only: https://github.com/KVM-VMI/qemu/tree/kvmi-v6

Changes:

- Support for KVMI handshake and passing the socket to host kernel
- Hooks for:
  - VM reboot
  - VM shutdown
  - VM pause / resume
  - VM migration
- all hooks send an event to the security application -> remove any hooks from introspected guest

How we test it

Automated tests running on:

- Windows: 7, 8.1, 10 RS1-R5, Server 2008 R2, Server 2012 R2, Server 2016, Server 2019; x86 and x64
- Linux: CentOS 7+, Ubuntu Server 14.04+, SLES 12 SP2+, Oracle Linux 7.5+

Protected applications:

- Security software drivers
- Browsers: Chrome, Firefox, Internet Explorer, Edge, Opera
- Email clients: Thunderbird, Outlook
- Databases: MongoDB, PostgreSQL
- Other: Adobe Reader, Microsoft Office

Unit tests for all supported attack techniques; 100% coverage

Performance numbers

Xen:

- Event channels
- Lightweight implementation
- Host – guest shared ring buffer
- Guest has direct access to the ring buffer

KVM:

- VirtIO foundations
- vhost-vsock (derived from VMware’s vSock)
- BSD sockets layer (kernel and guest)

Performance numbers (cont.)

Open multiple links in multiple browser instances and wait for load to complete (maximum allowed time: 6s)

Times are: VMI on – baseline (lower is better)

OS	Browser	Xen (sec)	KVM (sec)
Windows 10 RS6 x64	Firefox	0.72	2.57
Windows 10 RS6 x64	Chrome	1.07	2.21
Windows 10 RS6 x86	Firefox	1.37	4.84
Windows 10 RS6 x86	Chrome	0.40	3.48
Windows 7 x64	Firefox	0.85	2.12
Windows 7 x64	Chrome	1.14	2.02

The vhost-sock optimization patches bring the times even lower!

XenServer 7.1 / Linux 5.2.9 + kvmi patches

Performance numbers (cont.)

UnixBench on Ubuntu 18.04 amd64

Scores are: VMI on – baseline (lower is better)

Benchmark	Xen	KVM
Dhrystone 2 using register variables	0.04	0.75
Execl Throughput	29.36	39.87
File Copy 1024 bufsize 2000 maxblocks	1.16	4.72
File Copy 256 bufsize 500 maxblocks	1.30	1.14
File Copy 4096 bufsize 8000 maxblocks	2.21	4.21
Pipe Throughput	1.30	0.58
Process Creation	55.86	65.39
Shell Scripts (1 concurrent)	34.80	34.60
Shell Scripts (8 concurrent)	36.60	49.99
System Call Overhead	0.33	3.31

Notable attacks

Using VMI the following can be detected and blocked:

- CVE-2016-5195 – Dirty COW
- CVE-2017-0144 – EternalBlue
- CVE-2019-0708 – BlueKeep
- CVE-2019-1125 – SWAPGS Attack
- DoubleAgent (Windows)
- Mimikatz (Windows)
- Doppelganging (Windows)

Related projects

Thin libVMI alternative: https://github.com/bitdefender/libbdvmi (no specific OS support; KVM driver to be published)

libkvmi: https://github.com/KVM-VMI/kvm/tree/kvmi-v6/tools/kvm/kvmi

Mathieu Tarral's work:

- libVMI's KVM driver port to KVMI: https://github.com/KVM-VMI/libvmi
- pyvmidbg: https://github.com/Wenzel/pyvmidbg
- libmicrovmi: https://github.com/Wenzel/libmicrovmi

There is on-going work to improve the performance of vhost-vsock (Red Hat)

THANK YOU