KVM FORUM

Bringing Commercial Grade Virtual Machine Introspection to KVM

Mihai Donțu / Bitdefender

THE LINUX FOUNDATION

Outline

- What is VMI
- Why VMI
- Quick History
- How We Use VMI
- Current Status in KVM
- Ongoing Work on KVM

What is VMI - Virtual Machine Introspection

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

- what 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 guest tools.

Additionally, use hardware features (EPT/NPT) to enforce memory access restrictions.

Why VMI

- Modern kernels are very complex
- Same for user software (eg. browsers)
- Contain bugs that can lead to total system compromise
- Hardening them is a process (just as complex)

Why VMI (continued)

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

Why VMI (continued)

For security applications, VMI:

- Offers better isolation
- Removes the reliance on the guest OS in order to function
- Minimum (if at all) interference with the guest OS

Why VMI (continued)

Very good for building defences against zero days and APT-s

Example: Bitdefender’s VMI-based software stopped EternalBlue (CVE-2017-0144) in its tracks without prior knowledge

Why VMI (continued)

Coupled with existing virtual infrastructure management solutions, it can become a powerful tool for forensics and event correlation

Barely explored territory from a software security standpoint

Quick History

- **2003** – Garfinkel & Rosenblum: “A Virtual Machine Introspection Based Architecture for Intrusion Detection” - the starting point for a considerable amount of academic research
- **2006** – Jiang & Wang: “’Out-of-the-box’ Monitoring of VM-based High-Interaction Honeypots”
- **2008** – Dinaburg et al.: “Ether: Malware Analysis via Hardware Virtualization Extensions” - built on top of Xen 3.1
- **2008** – VMsafe API announced by VMware, which provides access to a guest’s: CPU, memory, disk, I/O devices etc. Supported memory introspection for vSphere / ESXi
- **2010** – VMware vShield Endpoint (as a replacement for VMsafe API) in-guest agent based file introspection only
- **2012** – VMware deprecates VMsafe

Quick History (continued)

- **2014** : open source effort to improve VMI in Xen
- **2017** : Bitdefender HVI - first commercial security application using open source VMI software (Citrix XenServer)
- **2017** : begin work with the KVM community on designing a VMI subsystem

How We Use VMI

With the help of VMI and specifically EPT/NPT 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 (eg. 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)

Current Status in KVM

It is possible to do VMI via qemu (QMP), but it is limited:

- no events (eg. for CRx changes)
- no control over EPT/NPT
- slow access to guest memory
- slow for in-line use
- qemu is a sensible component on its own

Ongoing Work on KVM

- proposed a separate VMI subsystem (KVMI)
- agreed upon an initial API
- currently focusing on x86, next ARM
- working on basic functionality
  - retrieve basic guest information
  - pause/unpause vCPU-s
  - get/set registers
  - get CPUID
  - get/set page access (plays with EPT/NPT)
  - inject exceptions
  - read/write guest memory
  - configure events for: CRx, MSR, breakpoints (INT3), hypercalls, EPT/NPT page faults, traps etc.

Q&A

Thank You

KVM FORUM

THE LINUX FOUNDATION