Zero-Footprint Guest Memory Introspection from Xen

Improving VM Introspection Using Hardware Virtualization Extensions
Zero-Footprint Guest Memory Introspection from Xen

Improving VM Introspection Using Hardware Virtualization Extensions

Mihai DONȚU – Technical Project Manager, Bitdefender
Ravi SAHITA – Principal Engineer, Intel

Xen Project DEVELOPER SUMMIT
August 18th, 2014
Outline

• Security issues
• Memory introspection
  o Operation
  o Evolution
• XenAccess and mem-events enhancements
• Sample usages
• Hardware Acceleration for memory introspection
• Conclusions
Security issues we are facing today

- Advanced malware types
  - Rootkits
  - Kernel exploits
  - Zero-days

- APTs, botnets, cyber-espionage and so on heavily rely on those…
Security issues we are facing today

New Linux Kernel Vulnerabilities
(CVEs, nvd.nist.gov)

- 2011: 140
- 2012: 70
- 2013: 180
Security issues we are facing today

Malware today execute in the same context and with the same privileges as anti-malware software ➔ lack of isolation problem

Common Malware

Advanced Malware

Isolation Bypassed & Malware Controlled
Conclusion: advanced attacks evade traditional security solutions
Envision the big picture

so... what's the big difference?...
Memory introspection

Envision the big picture

Hypervisor Controlled, Hardware Enforced
STRONG ISOLATION

dom0

SVA (domU₀)

Memory Introspection Engine

domU₁

domU₂

Introspection Engine Alternative 3

Protected area

Xen Hypervisor

Introspection Engine Alternative 2

domUₙ
What is memory introspection?

- Address a number of security issues from outside the guest OS without relying on functionality that can be rendered unreliable by advanced malware.

- Analyze raw memory image of guest OS, services and user mode applications, then identify:
  - kernel memory areas
    - driver objects, driver code, IDT, etc.
  - user memory areas
    - process code, process stack, process heap, etc.
How does it work?

- Use existent hardware virtualization extensions (Intel EPT / AMD RVI)
- Set hooks on guest OS memory
  - mark 4K pages as non-execute or non-writable
  - hooking & notification must be supported efficiently by HV & CPU
- Audit access of those areas by the code running in VM (OS or apps)
  - write attempts (driver objects, fast I/O tables, page tables)
  - execution attempts
- Allow or deny attempts – decision provided by security logic
How does it work?

EPT protected areas
provide detection for attempts & protection against integrity violation

EPT protected areas
provide detection for various OS level changes (ex. new module load, new process start, ...)

Guest VM Physical Memory Space
OS kernel code
user mode stacks & heaps
kernel driver code and data
user mode code
kernel data
SSDT, IDT, ...

OS
kernel
code
user
mode
stacks & heaps
kernel
driver code
and data
user
mode
code
kernel
data
SSDT, IDT, ...

www.bitdefender.com
How does it work?

• Building a **reliable image of the guest OS state**
  - what objects are inside a guest VM?
  - what operations are being performed inside a guest VM?
  - object and event identification and correlation is done by the introspection engine – to decide event and object maliciousness

• Using hooks we can detect numerous events, including
  - a driver / kernel module is loaded or unloaded
  - a new user process or thread is created
  - user stack / heap is allocated
  - memory is being paged in / out
How does it work?

**Traditional in-guest security solution**
- Enum processes, files, …
- Read mem by Virtual Addr, read files, registry, …
- Setup well-known callbacks & notifications
- New process PID 0x1234

**Out-of-guest memory introspection**
- Read mem by Physical Addr
- Read vCPU registers

**Typical Anti-Malware Kernel Module**
- Write / Execute attempt on PA 0x000A12345678

**XEN Hypervisor**
- Setup EPT hooks on mem pages (by Physical Addr)
- RAX = 0x1234
- RIP = 0x7890

**Memory Introspection Engine**
- 01010 10011
Two big challenges

• bridging the semantic gap – obtain rich semantics from only raw physical memory pages and virtual CPU registers
  o how do we correlate 4K memory pages with semantically rich and meaningful OS specific data structures?
  o to be solved by security solution vendors

• forward lots of mem-event notifications with low overhead to introspection engine
  o to be solved by hypervisor and CPU vendors
Memory introspection evolution

  - the starting point for a considerable amount of academic research
- 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
Memory introspection in Xen

- 2007 – XenAccess, XenProbes
- 2008 – Lares
- 2009 – first patches for the mem-events API
- 2010 – LibVMI – uses XenAccess and XenStore
  - targets Xen, but support for other HV-s can be added
  - insufficient flexibility in changing page permissions
  - no support for mapping guest memory RW
  - insufficient information about the guest CPUs
  - high overhead when reading the vCPU register state
  - a ‘complete’ initialization requires intimate knowledge about the guest OS
  - code for handling specific guest OS-s (Windows, with Linux in the works)
XenAccess and mem-events enhancements

- 2014 – Bitdefender published a set of patches (as RFC) to enhance the mem-events API
  - implements its own version of LibVMI
  - simpler API
  - nothing [guest] OS specific
  - support for other HV-s can be added
  - allows to map guest memory (via EPT)
  - uses a very simple page cache to optimize (un)maps
  - optimized access to specific resources
- Some patches went into mainline, others will follow shortly
XenAccess and mem-events enhancements

```c
uint32_t flags;
uint32_t vcpu_id;
uint64_t gfn;
...
mem_event_regs_t regs;
```
Example use of the extended API

- SVA (domU₀)
  - Memory Introspection Engine
  - Protected areas
  - Critical Kernel Module
  - Critical Kernel Module
  - Code, stacks, heaps, IAT, ...

Mem-events and VMCALLs facilitated by XEN

Xen Hypervisor
Example use of the extended API

- Bitdefender’s own introspection engine can
  - protect the kernel from known rootkit hooking techniques
  - protect user processes (e.g. browsers, MS Office, Adobe Reader, …) from
    - code injection
    - function detouring
    - code execution from stack / heap
    - unpacked malicious code
    - inject remediation tools into the guest on-the-fly (no help from ‘within’ needed)
- Runs in userspace in a user domain (SVA – Security Virtual Appliance)
- Introspection logic has very small overhead
  - bulk of the overhead is given by sending / receiving events and calls between protected guest VMs and SVA
Hardware Acceleration for Memory Introspection

Ravi Sahita
Intel
Hardware Acceleration for Memory Introspection

- Factors Limiting VM Memory Monitoring Performance
- Addressing Lack of Memory Isolation
- Addressing Performance gaps for execution and data access-control policies
- Xen Extensions
Factors Limiting VM Memory Monitoring Performance

• Round-trip time
  – Monitoring execution and data accesses
  – Dynamic data structures imply high frequency events

• Filtering events
  – Monitoring data accesses requires filtering non-interesting events due to 4K page sharing

• Further, round-trip time is amplified with VMMs nesting
Multiple EPTs as Protection Domains

Extended Page Table (EPT) Domains

VM₀

Intel® VT-x with EPT

Hypervisor

EPT Walker

CPU₀

host physical address

Execution crossing EPT domains or data
Accesses causing events

OS kernel Code/data (RX/RW)

Driver Code (RO)

Data (NP)

OS kernel Code/data (RO/NP)

Driver Code (RX)

Data (RW)
Addressing Lack of Isolation...

Diagram showing the relationship between applications and security solutions.

- App1 (Office)
- App2 (Browser)
- Security Solution
- Drivers
- OS Kernel
- Security Filter
- Hypervisor-derived isolation
- Events
- CPU
- EPT Domains
…Without Sacrificing Performance

- Must allow for legal execution of components isolated in permission domains
- Each execution transfer across domains leads to VM exits that the hypervisor must mediate
- As components isolated via domains, numerous execution transfers are induced
- High Frequency of such VM exits to the hypervisor causes overhead

Eliminate these induced VM exits on legal control transfers
VM Functions: Hypercalls Without VM Exits

- VM Functions: Intel® VT-x extensions for services configured by the hypervisor
  - Different VM Functions correspond to different services
- Hypervisor configures VM Functions via new fields in VMCS
- Guest software invokes VM Functions via new instruction (VMFUNC<leaf>)
  - Value in EAX specifies which VM Function leaf is invoked
- CPU provides services as configured with no VM exit
VMFUNC-based Domain Switching

- Paravirtualized software can request protection domain switch via VMFUNC (specifying domain index)
- Hypervisor pre-configuration domain index to EPTPs
- Hypervisor pre-configures domain boundaries
- View switching to speed up access control policies
Asynchronous Induced VMExits

- In VM-introspection scenarios critical data is monitored in place.
- Legacy approaches are to VMExit and emulate access.
- Alternatively, VMExit and switch views to single step the guest (MTF).
- High frequency of writes to monitored data cause high overhead.
- Requires custom logic in the VMM increasing complexity/state in the hypervisor.
Accelerating Induced Events

- **Via Virtualization Exception (#VE)**
  - Mutates EPT violations into a new IA exception – delivered through guest IDT

- VMM enables EPT violation conversion to #VE
- Data access monitoring view policies setup in EPT domains
- Data access causes #VE instead of VMexit
- Guest monitoring agent can emulate in guest OR use VMFUNC to single step access
VMFUNC Configuration

- Hypervisor checks IA32_VMX_VMFUNC MSR for allowed VM-Function controls
- Opts-in by setting “Enable VM functions” in the secondary processor-based VM-execution controls field
- Activates “EPTP switching” by setting bit-0 in the VM Function Control
- Configure alternate EPTP values in memory referenced via VMCS field
- Guest software uses VMFUNC opcode with leaf selector EAX=0 and ECX containing the index of EPTP (view) selected
- Errors reported to the hypervisor via VM Exits
Virtualization Exception (VE) Configuration

- Enumerated by the VMM via capability MSR
- Set VMCS “Enable VE” bit
- Negotiate “VE Info” page with Hypervisor
- #VE delivered through guest IDT
- Suppress VE EPTE Bit 63
  - Set on pages the VMM does not want to cause a #VE for
Xen Extensions

1. Efficiently creating and maintaining alternate EPT views/domains via extension of p2m
2. Hypercalls to edit EPT permissions without conflicting with Xen EPT management
3. Report guest-specific memory events via #VE in a Xen compatible manner (Suppress #VE EPTE bit)
4. Enabling CPU acceleration if VMFUNC and #VE CPU enumerated and opted-in
VM Introspection Performance Improvements

• Round-trip time
  – VMFUNC to allow safe, fast Memory View (EPT) switches
  – VMexits mutated to #VE for guest memory monitoring

• Filtering events
  – Reduced latency of #VE event handling reduces overhead of filtering events

• Round-trip time amplification due to VMM nesting
  – No VMexits to root VMM implies no amplification of VMexits due to EPT violations
Conclusions

• Today Xen can be the base for providing a much improved layer of security – serves as a model for other HV vendors
  o Truly agentless security (zero in guest footprint)
  o IT Admins can deploy introspection based solutions seamlessly, without changing a single line of config inside the guest VMs
• Hardware enforced isolation (against kernel exploits, zero days, …)
• Hardware extensions enable intra-VM isolation to enable protected agent based introspection for high frequency access monitoring and agent isolation
• Both models require straight-forward Xen infrastructure changes (multi-EPT views, hardware acceleration capabilities)
Thank you!