CHIPSET BASED DETECTION AND REMOVAL OF VIRTUALIZATION MALWARE a.k.a. DeepWatch

Yuriy Bulygin
Joint work with David Samyde
Security Center of Excellence / PSIRT, Intel Corporation

Black Hat USA 2008
To my wife, Natalia
AGENDA

• Introduction
• Chipset based detection of virtualization malware
• DeepWatch: proof of concept detector
• Removing virtualization rootkits
• Limitations
• Demo
• Detecting SMM rootkits
• Bypassing detection
• Comparing with other detection approaches
• Summary
INTRODUCTION
(G)MCH-BASED PLATFORM OVERVIEW
(G)HMM.. PROBLEM
EMBEDDED μCONTROLLER
uC SUMMARY

• (G)MCH a.k.a. NorthBridge has embedded microcontroller(s)
• Embedded uController in NorthBridge appears as a separate integrated device on PCI bus with one or more PCI functions
• Assigned with its B/D/F for device enumeration by BIOS
• Embedded uController may integrate different hardware engines such as various bus controllers, PIC, crypto accelerators, DMA engine(s) etc.
uC Firmware

- Embedded uController runs firmware
  - Real-time operating system (RTOS)
  - Firmware drivers operating hardware engines
  - Firmware applications

- Embedded firmware can operate hardware engines in chipset such as crypto hardware, internal DMA..

- Internal SRAM memory or memory stolen from DRAM is used for firmware code, stack/heap..

- External non-volatile memory is used for storing firmware binaries, e.g.:
  - Serial Peripheral Interface (SPI) Flash memory
uC EXAMPLE

• Intel® Active Management Technology (iAMT) [13]
• New Intel chipsets have embedded uController which executes Intel firmware digitally signed and stored in non-volatile SPI Flash
SO CAN WE USE uC TO DETECT VIRTUALIZATION MALWARE ??
BUT THIS IS INSANE

• Embedded firmware runs on embedded core in the chipset
• .. “underneath” any hypervisor executing on the host CPU
• uC’s internal DMA hardware can be programmed by this firmware to access OS/hypervisor memory in DRAM
• .. and scan for code/structures of known HW virtualization rootkits and remove them
• .. or verify integrity of running hypervisor
HERE GOES A ROOTKIT

```
C:\work\deepwatch>bpknock.exe 0x7c70
knock answer: 0x7280202

C:\work\deepwatch>net start vtrootkit
The vtrootkit service was started successfully.

C:\work\deepwatch>bpknock.exe 0xc70
knock answer: 0xbadd00d
```
Programming internal DMA hardware in JTAG debugger to copy 64 bytes from 0x73000 host phys addr to internal memory.
INTRODUCING DeepWatch: CHIPSET BASED DETECTOR PoC

DeepWatch

- DeepWatch is implemented as a Proof of Concept in (G)MCH firmware on Intel® Q35 chipset codename “Bearlake”
- Detects Intel® VT-x based rootkits on Intel® Core 2 Duo CPU
- Implemented in debug Intel firmware on debug Silicon
CURRENT IMPLEMENTATION

• DeepWatch firmware is stored with the rest of Intel firmware in SPI Flash
• DeepWatch firmware thread starts with BIOS POST code after BIOS initialized memory controller
• DeepWatch periodically programs internal DMA hardware in uC to access physical memory
• DMA transfers 32/64 kB chunks of physical memory into internal memory
• DeepWatch then scans DMA-ed contents for code/data of known VT-x rootkits
VM EXIT HANDLER

• Hypervisor may have a lot of various code !! So what exactly to detect ?

• Hypervisor traps on certain instructions (e.g. CPUID) or events (e.g. exceptions)
  – Traps may be unconditional or conditional

• Trapping by the hypervisor is called “VM Exit”

• VM Exits are handled by VM Exit handler code
  – VM Exit handler reads VM Exit reason and lots of other information about VM Exit and guest state from VMCS
  – May be located anywhere in physical memory

• DeepWatch uses opcodes specific to malicious VM Exit handler as a “signature”
KICKING OFF DEEP WATCH SCAN

```
g_pill_sig points to signature buffer
```

```
g_detected_at will be set to physical address of detected VM Exit handler
```

```
g_local_buffer points DMA destination buffer
```

```
arc> v g_pill_sig
(UINT8[]) g_pill_sig = 0x1076660

arc> dump 0x1076660 32
01076660: 000c703d 0f077400 0223e9a2 0db80000 | =p....t.....#......
01076664: 000c703d 0f077400 0223e9a2 0db80000 | .....=p....t.....#

arc> v g_detected_at
(UINT32) g_detected_at = 0x0

arc> v g_local_buffer
(UINT8[]) g_local_buffer = 0x124e5c8

arc> dump 0x124e5c8 32
124e5c8: 00000000 00000000 00000000 00000000 | ............
124e5d8: 00000000 00000000 00000000 00000000 | ............
```

```
arc: halted at 0x1058640 in _hl_blockedPeek() at hl_bios.c:108

arc> arc:dbgprint()

000 [ deep watch ]: # START DMA AND SCAN OF HOST PHYSICAL MEMORY..
001 [ deep watch ]: # base physical address 0x00000000
002 [ deep watch ]: # size to scan 0x100000 bytes
003 [ deep watch ]: # signature length 0x13 bytes
004 [ deep watch ]: 0x00000000: DMA and scan block of physical memory..
005 [ deep watch ]: 0x00007fed: DMA and scan block of physical memory..
006 [ deep watch ]: 0x0000ffda: DMA and scan block of physical memory..
007 [ deep watch ]: 0x00017fc7: DMA and scan block of physical memory..
008 [ deep watch ]: 0x0001ffb4: DMA and scan block of physical memory..
```
DETECTING ROOTKIT SIGNATURE

VM Exit handler of rootkit detected at 73000h phys addr.

“Signature” of DMA-ed malicious VM Exit handler
WHY IS IT INTERESTING ??

• Detects virtualization rootkits rather than virtualization

• Detects rootkits in memory regions inaccessible to host OS and anti-viruses

• Detection can be unnoticeable by the user:
  - doesn’t consume main CPU time !!
  - fast DMA access to DRAM (hundreds MBytes per second)
  - could scan for rootkit in memory/SSD while host is in sleep

• Can provide capability to anti-virus engines to detect virtualization rootkits

• Can provide hardware based verification of Patch Guard or other OS kernel-mode rootkit detectors
DETECTION.. DETECTION.. HOW ABOUT REMOVING VIRTUALIZATION ROOTKITS ??
REMOVING VIRTUALIZATION ROOTKIT

• Once detected VM Exit handler code we can program internal DMA hardware in the opposite direction

• And replace VM Exit handler with any contents

Option 0:

• Corrupt or wipe off malicious VM exit handler/VMCS

• Quick and dirty .. but unaesthetic
OPTION 1 - “OWN THE ROOTKIT”

• Detect VM Exit handler code of the rootkit
• Replace VM Exit handler code with your own VM exit handler
• New VM Exit handler works as a legitimate hypervisor - traps on events/instructions and resumes the guest OS
• OS is still virtualized but by new hypervisor – harmless ;)

• Wait for the demo !!
OPTION 1 (cont’d)

- VM Exit handler of malicious hypervisor responds with 0xbadd00d to magic CPUID request
- DeepWatch replaced it with the handler that responds with 0xdeadd00d
OPTION 2 - “ROOTKIT SUICIDE”

• Detect VM Exit handler code of the rootkit
• Replace VM Exit handler code with a new code
• .. that restores guest state and does VMXOFF
• Upon the next VM Exit, rootkit restores guest OS state with “trampoline” code and leaves VMX mode of operation
• .. effectively turning itself off ;)
• It’s like Blue Chicken but forever
LIMITATIONS

• Detection is OS-independent but chipset-specific
• Embedded uController and internal DMA hardware are chipset specific
• DeepWatch detects only VT-x based rootkits
SIGNATURE SCAN or INTEGRITY CHECKS ??

• Current DeepWatch uses simple and effective signature matching ;)

• But it’s not about signature scan; it can combine multiple techniques

• Heuristics:
  – E.g. search for VMCS structures by revision id to get an address of VM Exit handler
  – Then hash VM Exit handler and compare against white-list

• Integrity checks of host OS/hypervisor pages:
  – How does it know what to hash ??
  – A lot of information can be learned about OS and VMM from bare memory dump [20-24]
  – Find hypervisor CR3, page tables, VMCS structures etc.

• DeepWatch can cooperate with the host code (SMI handler like HyperGuard) that has this information about OS/VMM
For information on HyperGuard please see “Xen Owning Thrillogy” by Joanna Rutkowska/Alex Tereshkin/Rafał Wojtczuk (Invisible Things Lab)
DEMO

REMOVING VT-x BASED ROOTKIT
DETECTING SMM ROOTKITS

• SMM malware compromises SMM memory (SMRAM) protections to run in System Management Mode [17-19]
  – Also today’s talk by Shawn Embleton & Sherri Sparks

• Chipset/CPU don’t allow non-SMM CPU access to SMRAM after it’s locked

• Chipset doesn’t allow DMA access to SMRAM by I/O devices

• Anti-viruses or external DMA devices cannot verify SMRAM contents after it’s locked by malware

• But chipset could allow its own embedded uController to access SMRAM so that

• DeepWatch could verify SMI handler code and static data in SMRAM
FORGOT A SMALL DETAIL: DMA REMAPPING
VT-d conformant chipset should remap DMA cycles from all I/O devices including from its own internal uControllers.
DMA REMAPPING

• VT-d capable chipsets have one or more DMA-remapping engines virtualizing Directed I/O access [12]
• Internal devices are also a subject to DMA-remapping
• Chipset has dedicated register-set for each DMA-remap unit accessible by software as MMIO range which software can use to protect certain memory regions from certain I/O devices
• Rootkit can create DMA-remapping page tables to translate addresses of DMA requests issued by embedded uC (identified by its PCI B/D/F) to different host physical addresses
  – or read/write protect entries in DMAr pages tables
  – or mark context-entry as not Present to cause translation fault
  – or enable PLMR/PHMR DMA-protected regions to prevent any DMA
• And relocate code/data (VMExit handler, VMCS ..) to memory protected by DMAr page tables or to PMR regions
SO WHAT CAN WE DO ABOUT THIS ??

• DMA-remapping unit can distinguish DMA requests issued by DeepWatch internal device function inside embedded uC

• by its requester id from DMA requests issued by other internal functions

• and not translate them

• Or disable and lock DMA-remapping of DeepWatch device function if DeepWatch is used

• And allow only trusted software like SMX authenticated code modules (Intel® TXT) to enable and program DMA-remap engine for DeepWatch
COMPARING WITH OTHER DETECTION APPROACHES
ANOMALY BASED DETECTION

• Timing measurements of instructions causing VM Exit
  - Local timing: RDTSC, ACPI timer, Local APIC, RTC [8, by bugcheck]
  - Remote timing: NTP [8]
  - Using another thread on SMT CPU to measure VM Exit latency [7]
  - Using timers of integrated devices in the chipset

• TLB profiling of instructions causing VM Exit events
  - Measure timing of address translation due to TLB evictions by a hypervisor [5,6,8,9]
  - TLB coloring: observe if TLB VA-2-PA mappings changed due to VM Exit [9]
  - Measure # of TLB misses due to flushing TLB’s upon VM Entry/Exit

• Using μArchitectural side-channel attacks
  - RSB based side-channel: corruption of RSB state by VM Exit handler [25]

• Other: CPU errata, causing faults or exhausting resources, Last Branch Record, different CPU behavior in VMX root vs. in non-root modes
COMMON PROBLEMS

• Do not distinguish good hypervisors from bad
  - *Detected SSDT hook, it may be XXX anti-virus or a kernel rootkit. Remove ??*

• Probabilistic: need to run lots of tests to reduce probability of a false positive

• How about removing rootkits from the system ??

• Conceptual contradiction: any VMX non-root detector is less privileged than any VMX root(kit)
  - Detector shouldn’t have any legitimate way to affect more privileged VMX root(kit) by design of virtualization

• Agents win ;)

- **Note:** The word “XXX” is likely a placeholder for a specific anti-virus or rootkit software.
HARDWARE MEMORY ACQUISITION

• Uses DMA capable device to acquire physical memory dump and perform forensic analysis [15]

• DMA remapping (VT-d) hardware in chipsets protects VMM pages from external I/O devices. Hypervisor can always avoid detection by programming DMA remap

• Method requires discrete DMA capable card. Can IT security folks scan memory of every host physically with external device ??
SUMMARY

• DeepWatch approach = using embedded μController(s) in chipset to reliably detect and remove virtualization malware

• DeepWatch can use various techniques: integrity checking, signature scan or combine them

• This approach can bring benefits to OS anti-rootkit solutions or to traditional anti-malware engines

• Can enable detection of other rootkits inaccessible to OS/anti-viruses such as SMM rootkits

• Single solutions fail. DeepWatch should coexist with preventative and other detection solutions (like HyperGuard)
FINAL REMARKS

• This work is now a joint research with Intel Corporate Technology Group, Networking Technology Lab

• Acknowledgements:

  Dean Krekos, Sagar Dalvi, Jason Fung, Toby Kohlenberg, Howard Herbert

  All authors of research in virtualization rootkits and their detection techniques

• Updated materials: http://www.c7zero.info
Thank you !!

yuriy.bulygin@intel.com
Intel Security Center of Excellence (SeCoE)

- Evaluates Intel products, platforms and technologies for security vulnerabilities
- Staffs Intel Product Security Incident Response Team (iPSIIRT) to respond to security incidents in Intel products
- Drives Secure Development Lifecycle process across Intel products
- Doesn’t do all that alone: works with all security architects, technology architects, development and validation teams

secure@intel.com
http://www.intel.com/security
REFERENCES

2. Joanna Rutkowska, Subverting Vista Kernel For Fun And Profit, Black Hat USA 2006, SyScan 2006
   http://bluepillproject.org/stuff/IsGameOver.ppt
6. Peter Ferrie. Attacks on Virtual Machines. Symantec Corporation
7. Edgar Barbosa, Blue Pill Detection, COSEINC Advanced Malware Labs, SyScan'07
   http://rapidshare.com/files/42452008/detection.rar.html
   http://www.cs.cmu.edu/~jfrankli/hotos07/vmm_detection_hotos07.pdf
    http://crucialsecurity.com/
    http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf
    http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf
REFERENCES


19. BSDaemon, coideloko, D0nand0n. System Management Mode Hacks http://www.phrack.org/issues.html?issue=65&id=7#article


23. int for(ensis){blog;} -- PTFinder -- KntTools and KntList http://computer.foresnikblog.de/en/topics/windows/memory_analysis/


25. Yuriy Bulygin. CPU side-channels vs. virtualization rootkits: the good, the bad, or the ugly. ToorCon Seattle 2008. http://www.c7zero.info/home.html#hyper-channel