

Advanced x86:
Virtualization with VT-x
Part 3

David Weinstein

dweinst@insitusec.com

All materials are licensed under a Creative Commons “Share Alike” license.

- <http://creativecommons.org/licenses/by-sa/3.0/>

You are free:



to Share — to copy, distribute and transmit the work



to Remix — to adapt the work

Under the following conditions:



Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



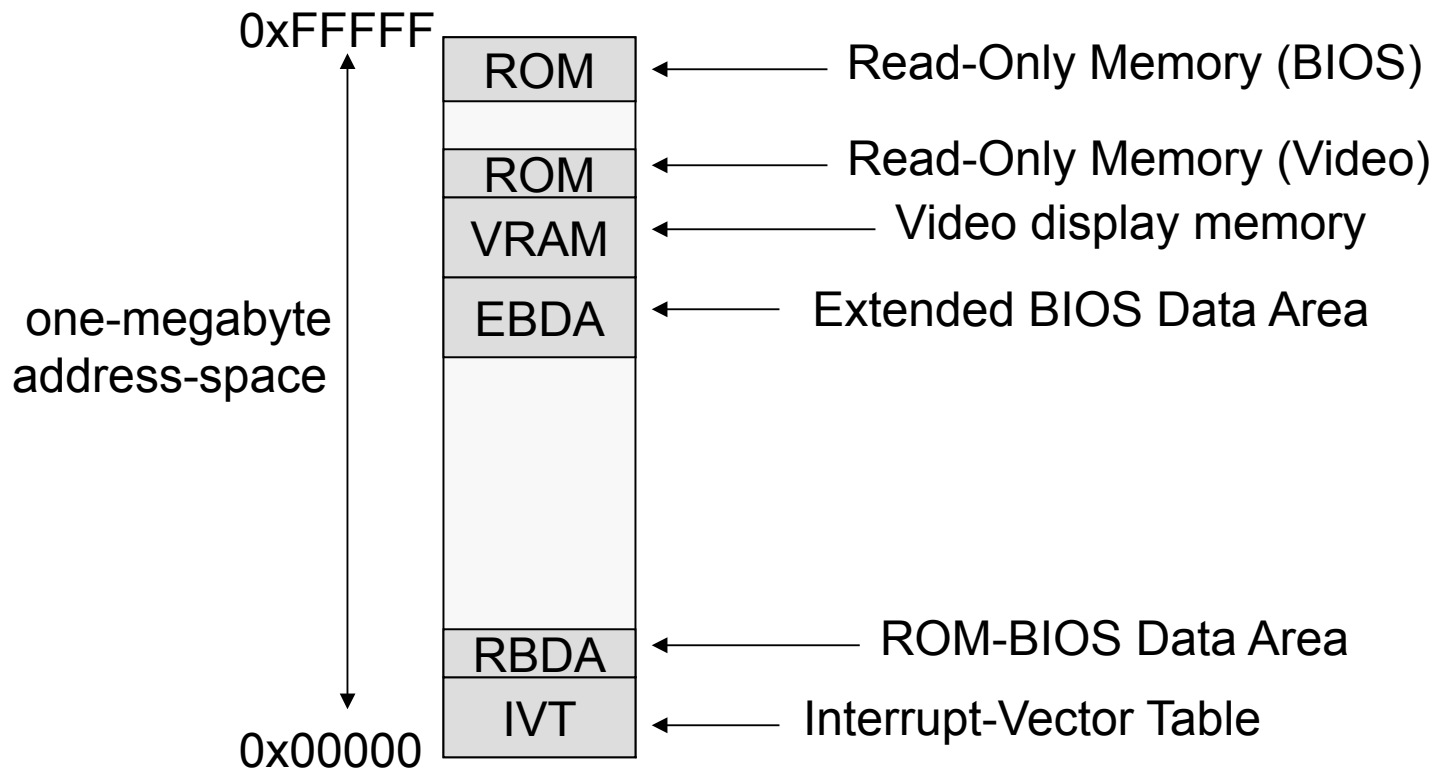
Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Real mode guest VM

- What if we wanted to run some real mode code as a guest VM.
 - Maybe because support for Virtual-8086 emulation is unsupported by the CPU's compatibility mode in 64-bit mode
- Allan Cruse (Prof. Emeritus @ University of San Francisco) shows us how to do this with a guest VM
 - <http://www.cs.usfca.edu/~cruse/cs686s07/lesson24.ppt>
- So I fixed the code to work with recent Linux 3.* kernels
- We'll get to experience the fun of calling a BIOS interrupt in a guest VM container 😊
 - In the comfort of our Linux environment

The Real Mode Address Space

- Code that uses real-mode addresses is limited to the bottom megabyte of memory:



Ref: <http://www.cs.usfca.edu/~cruse/cs686s07/lesson24.ppt>

Real mode guests... for reals

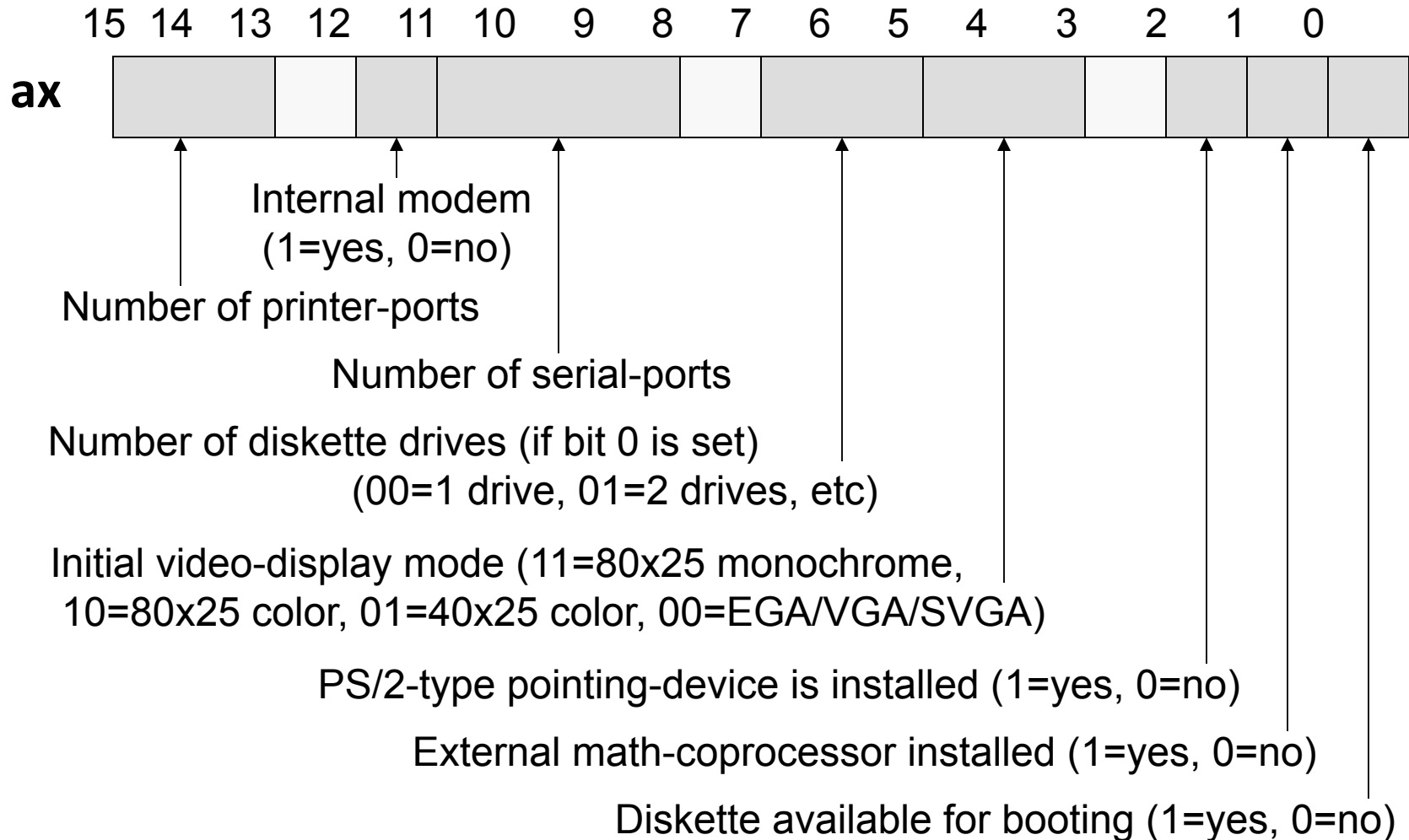
- To support guest real-mode execution, the VMM may establish a simple flat page table for guest linear to host physical address mapping.

BIOS Services

- int 0x10: video display services
- **int 0x11: equipment-list service**
- int 0x12: memory-size service
- int 0x13: disk input/output services
- int 0x14: serial communications services
- int 0x15: system software services
- More on BIOS stuff

<http://wiki.osdev.org/BIOS>

Int 0x11: Equipment List result



linuxvmm.c

- Character-mode Linux device-driver
 - Loaded as a kernel module at runtime
- Accessed via a `/dev/vmm`
- Can use standard functions like `fopen`, `mmap`, and `ioctl` to interact with the device
- Compile with included `mmake.cpp`
 - `g++ -o mmake mmake.cpp`
- Read the README file in the linuxvmm directory

tryoutpc.cpp

- Uses an ioctl on the /dev/vmm
 - Ask the host VMM to continue to execute the kernel's code in 64-bit mode, and to launch a Guest VM that will execute the real-mode procedure in Virtual-8086 mode
 - We'll supply the register values to be placed in the guest VM as part of the ioctl.
 - And we'll see the result when the guest exits and returns the resulting register values.

dram.c

- Another kernel module to allow us to browse the physical memory of the system
 - Exposed via `/dev/dram`
 - Best viewed with `fileview` (in `linuxvmm` dir)
 - `g++ -o fileview fileview.cpp`
 - `sudo ./fileview /dev/dram`

Lab: 'real-mode' guest VM container

- Purpose
 - Demonstrate execution of a real-mode guest
 - Execute BIOS interrupt 0x11 to obtain the available system device hardware
- Steps
 - README in directory

VM “Introspection” (1)

- By registering with the VMCS events that cause VMExits, the transition from VMX non-root to root mode allows the inspection of guest state and memory, thus allowing the external inspection of the guest.
- We’ve discussed a number of events that can trigger a VM exit and allow for inspection of the guest system’s state
- We’ve also shown how we can essentially create callbacks that get hit when the VM exit conditions match an event of interest

VM “Introspection” (2)

- Since the CR3 register contains the page directory pointer during a context switch (and thus VM Exit), this can be used to identify the upcoming process before it executes.
- At the end of the day, this is a tool that can be used for malware analysis, system integrity checking, code isolation, etc.

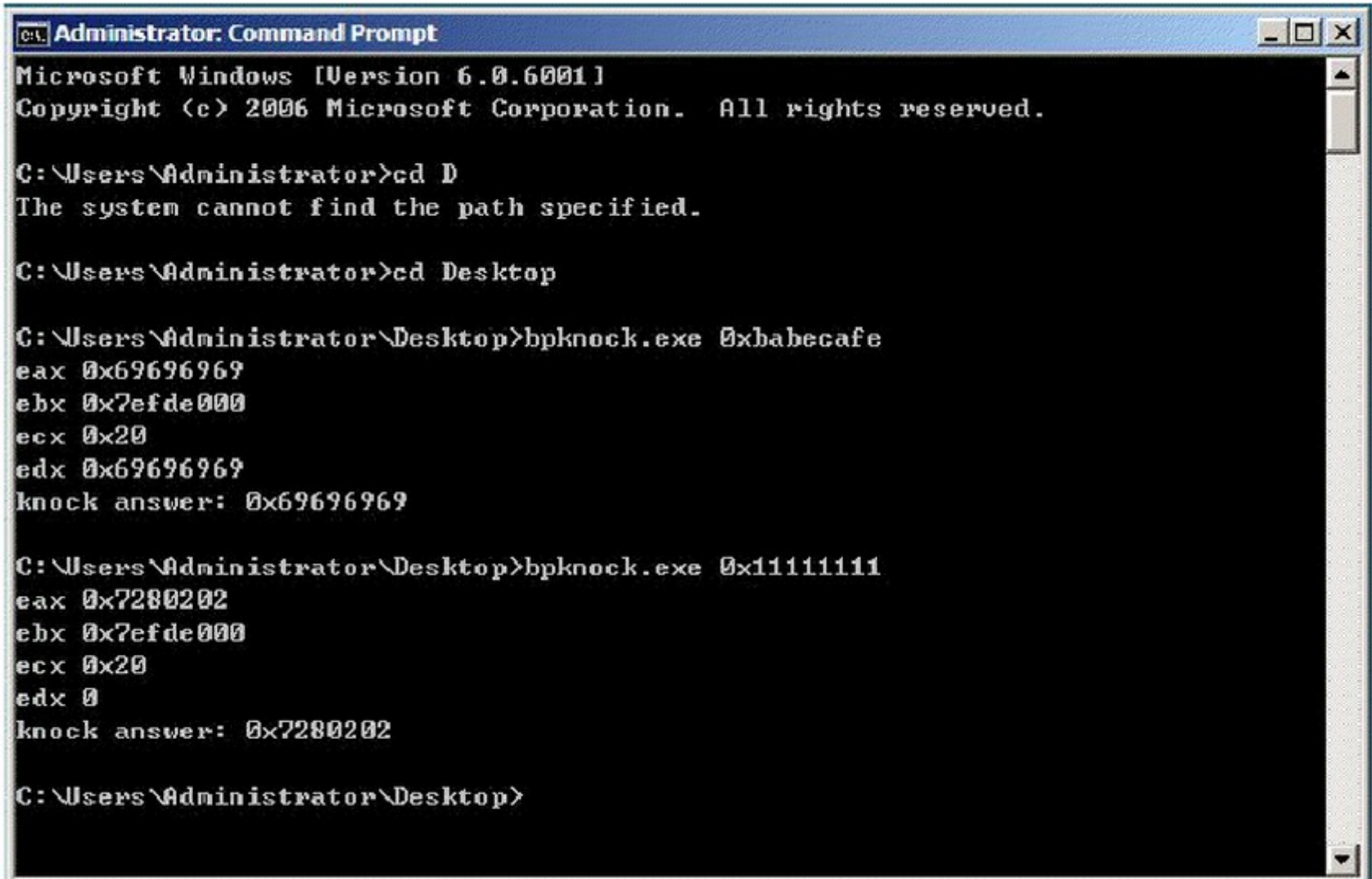
General Hardware VM Based Rootkit

- Virtual Machine Based Rootkit (VMBR)
- Start with CPL=0
- Allocate some unpagged physical memory
 - Ensure no linear mappings to VMM after guest entry
- Move running OS into VMCS
- Intercept access to hardware (IO ports, ...)
- **Communicate to hardware VM rootkit via sentinel instructions**

Keylogging in VMBR

- Setup VMCS appropriately
 - Determine the keyboard's IO ports
 - Intercept IO port access and handle/reinject the event to the guest VM.
- Look up the lab Xen0 made for talking to the keyboard controller
 - <http://opensecuritytraining.info/IntermediateX86.html>
- Another example, see Hyperdbg
 - <https://code.google.com/p/hyperdbg/source/browse/trunk/hyperdbg/keyboard.c>

bpknock



```
Administrator: Command Prompt
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd D
The system cannot find the path specified.

C:\Users\Administrator>cd Desktop

C:\Users\Administrator\Desktop>bpknock.exe 0xbabecafe
eax 0x69696969
ebx 0x7efde000
ecx 0x20
edx 0x69696969
knock answer: 0x69696969

C:\Users\Administrator\Desktop>bpknock.exe 0x11111111
eax 0x7280202
ebx 0x7efde000
ecx 0x20
edx 0
knock answer: 0x7280202

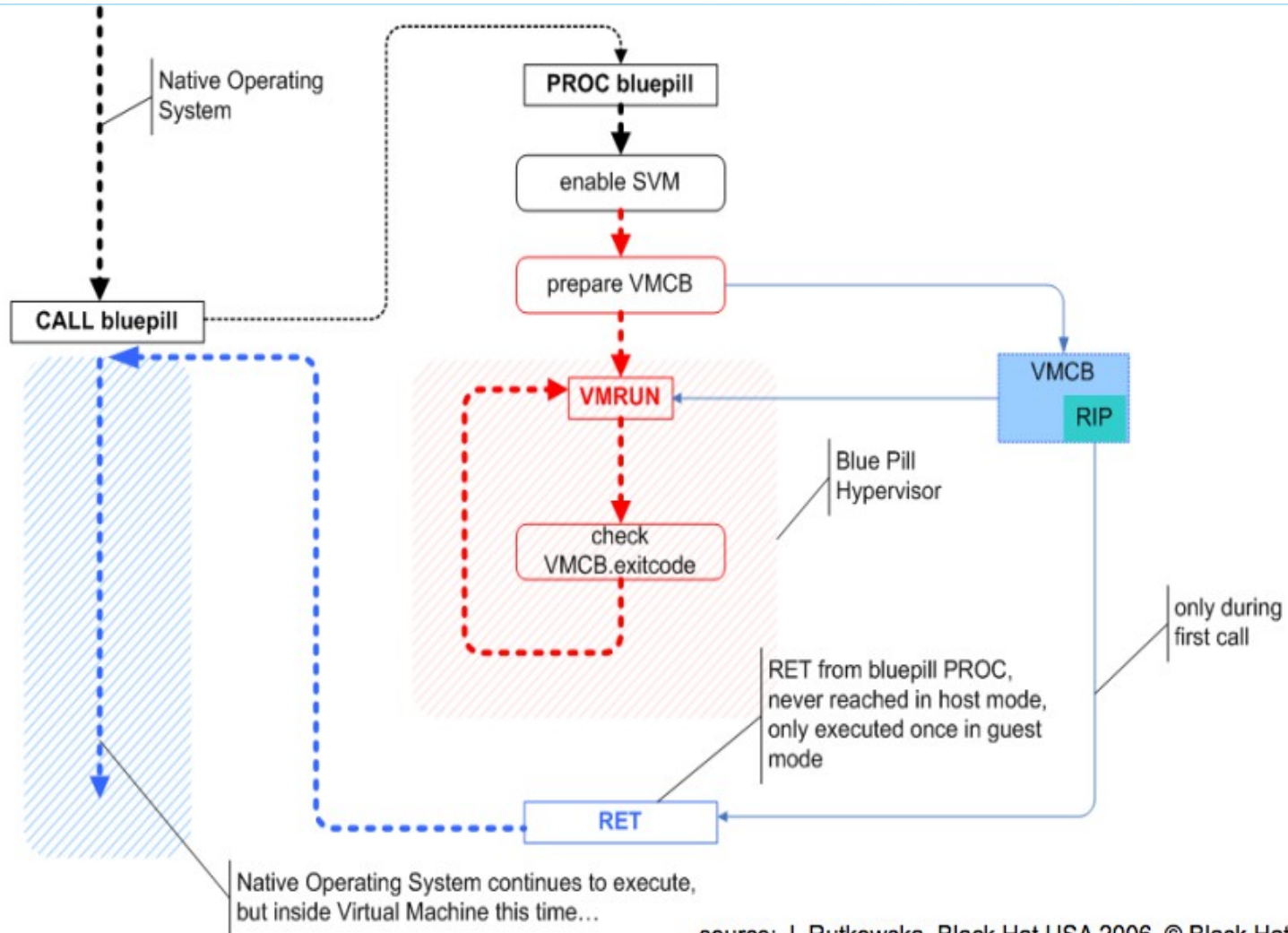
C:\Users\Administrator\Desktop>
```


bpknock

```
30 int __cdecl main(int argc, char **argv) {
31     ULONG32 knock;
32     if (argc != 2) {
33         printf ("bpknock <magic knock>\n");
34         return 0;
35     }
36     knock = strtoul (argv[1], 0, 0);
37
38     __try {
39         printf ("knock answer: %#x\n", NBPCall (knock));
40     } __except (EXCEPTION_EXECUTE_HANDLER) {
41         printf ("CPUDID caused exception");
42         return 0;
43     }
44
45     return 0;
46
47 }
```

```
1  /* bpknock.cpp
2  based on joanna rutkowska's bpknock (bluepill knock)
3  this utility communicates to the VMM via calls to CPUID
4  (which can be run in ring 3)
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <windows.h>
9
10 //typedef unsigned int ULONG32;
11
12 ULONG32
13     __declspec(naked) /*<- indicates no compiler generated prolog/epilog */
14     NBPCall (ULONG32 knock) {
15         __asm {
16             push    ebp
17             mov     ebp, esp
18             push    ebx
19             push    ecx
20             push    edx
21             cpuid
22             pop     edx
23             pop     ecx
24             pop     ebx
25             mov     esp, ebp
26             pop     ebp
27             ret
28         }
29 }
```

Blue Pill Idea (Simplified)



Vitriol

BH USA 2007. Goldsmith and Lawson

- ★ (1) get to cpl0
- ★ (2) check cpuid, feature msr for VMX
- ★ (3) allocate vmx and vmcs from IOMalloc
- ★ (4) initialize vmcs, call vmclear
- ★ (5) copy segments, stack, cr3 to vmcs host and guest
- ★ (6) set host(/root/hypervisor) eip to trap handler
- ★ (7) set exec controls to pick events we want

Virtualization projects

- Lguest, Xen, QEMU/KVM
- Vitriol (Matasano), BluePill/NewBluePill (ITL)
- Debugging
 - Hyperdbg, virtdbg
- Academic
 - SubVirt (Microsoft Research), V3vee Palacios (NWU), SecVisor (CMU), BitVisor (University of Tsukuba)

NewBluePill

- Created for a Black Hat training session
- Copyright terms are limiting (reproduced below)
- <https://bluepillstudy.googlecode.com/svn/trunk/nbp-0.32-public/>

```
; Copyright holder: Invisible Things Lab  
;  
; This software is protected by domestic and International  
; copyright laws. Any use (including publishing and  
; distribution) of this software requires a valid license  
; from the copyright holder.  
;  
; This software is provided for the educational use only  
; during the Black Hat training. This software should not  
; be used on production systems.
```

Lguest

- Simple x86 hypervisor for hosting other Linux kernels
- Load kernel module which you will load into running kernel
- Simple I/O for communication

Detecting Virtualization/VMBRs

- Godsmith, Lawson proposed detection heuristics [1]
 - Functional (behavior or state changes)
 - Side-channel (timing variations)
- Point methods
 - Processor errata
 - VMCall functions/CPUID results
 - Look for artifacts in processes, file system, and/or registry, memory.
 - Look for specific virtual hardware
 - Look for specific processor instructions and capabilities
- See RedPill, NoPill, and ScoopyNG
 - ScoopyNG = Scoopy Doo + Jerry

[1] http://www.matasano.com/research/bh-usa-07-ptacek_goldsmith_and_lawson.pdf

Instructions That Cause VM Exits Unconditionally

- CPUID, GETSEC, INVD, and XSETBV. This is also true of instructions
- introduced with VMX, which include: INVEPT, INVVPID, VMCALL,5 VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.
- Meaning there should be a timing difference caused by a VM exit

Instructions That Cause VM Exits Conditionally

- If you can figure out whether the software trying to escape detection will be forced to exit to support a particular pre-existing feature
- Section 25.1.3

RedPill

- Joanna Rutkowska, 2004 - “Red Pill... or how to detect VMM using (almost) one CPU instruction”
 - <http://www.invisiblethings.org/papers/redpill.html>
- Using SIDT (Store Interrupt Descriptor Table Register) instruction to profile the current value in the IDTR
- She had found that the most significant byte of the IDTR had a predictable value in VMWare 4 and VirtualPC, which was different from what it was in a non-virtualized system

Measuring time (1)

- CPU Tick Counter
 - RDTSC instruction
 - resolution: number of processor cycles (super high!)
 - very accurate, but trivial to cheat!
 - TSC offsetting/RDTSC VMExit
- High Precision Event Timer (HPET) and other local timers
 - Might have a high resolution
 - But we can cheat them
 - e.g. interrupt interception

<http://invisiblethingslab.com/resources/bh07/IsGameOver.pdf> (Slide 54)

Measuring time (2)

- Real Time Clock (RTC)
 - I/O with RTC device
 - resolution: milliseconds (poor)
 - relatively easy to cheat (I/O interceptions)
- External clock
 - e.g. NTP protocol
 - resolution: 10 milliseconds (very poor)
 - can not be cheated using generic approach – only attacks against specific implementation

TLB Profiling

- Based on the belief that a VMM puts the hardware TLB entries to 0 if it is intercepting an instruction.
- Technique
 - Detector can watch timing access of a page, calling a possibly intercepted instruction, and then once again timing access to the same page
 - Comparing both results should indicate a longer access time (from an external reference) if there has been an interception.

Volatile Memory Capture over DMA

- Access all of physical memory over external peripheral interface (e.g. Firewire)
- It should then be possible to detect a VMBR by searching for its signature.
 - What signatures would you look for having learned what we know about the various data structures?

BlueChicken/TOCTOU

- “It's a funny feature that allows Blue Pill to defeat timing-based virtualization detectors, so they can't find out that they're inside a VM. Obviously we do *not* need Blue Chicken in case there is Virtual PC in the system or any other application that makes use of hardware virtualization already.” - Joanna Rutkowska
- Blue Chicken puts Blue Pill into a mode where it doesn't interfere or cause VM exits. This sleep mode means it doesn't hook anything and could remain in memory encrypted to avoid detection.
- This is an example of a Time of Check Time of Use attack designed to protect Blue Pill from detection.

Nested Virtualization

- ***Ben-Yehuda et al. The Turtles Project: Design and Implementation of Nested Virtualization***
 - Talk: <http://www.youtube.com/watch?v=FbH63kVGTek>
 - “... our approach multiplexes multiple levels of virtualization ... on the single level of architectural support available”
- *Alexander Tereshkin (ITL), Bluepillling the Xen Hypervisor*
- Other uses
 - IaaS providers
 - Live Migration
 - Debugging hypervisors
- “Nested virtualization is needed in case we have some other applications in the target system that also want to make use of virtualization (e.g. Virtual PC 2007) or we have a system with built-in hypervisor. In both cases Blue Pill must run those applications and/or OS' own hypervisor as nested ones.” - Rutkowska

Cheat Engine

- “Cheat Engine is an open source tool designed to help you with modifying single player games running under window so you can make them harder or easier depending on your preference(e.g: Find that 100hp is too easy, try playing a game with a max of 1 HP), but also contains other usefull tools to help debugging games and even normal applications.”
- Implements a VMM along the way ☺ (DBVM)
 - SC2/D3 hax?
- <http://cheatengine.org/aboutce.php>
- <https://code.google.com/p/cheat-engine/>

SubVirt Rootkit

- *Wang et al.* SubVirt: Implementing malware with virtual machines
 - Microsoft research
 - Proof of concept against Windows XP and Gentoo Linux
 - On Windows it implants itself during system shutdown event (using LastChanceShutdownNotification event handler) so that it will load on next boot.
 - On Linux they modify init.d (rc.d?) scripts to load their VMBR on next boot.

Azure

- “Named after the rootkit that relies on similar principles for its operation, Azure is a proof-of-concept malware analysis tool for Windows XP-based guests that functions externally through the use of Intel VT. It was implemented using KVM (a Linux-based virtualization solution) as a base.”
- “Azure uses virtual machine introspection to identify a target process and fine-grained tracing to monitor its behavior; coarse-grained tracing is left as future work.”
 - <https://code.google.com/p/azurema/>

Some old and new VMM Bugs

- VMMs are non-trivial to write
 - Microsoft Virtual Server 2005 R2, CVE-2007-0948
 - CVE-2006-5379, Nvidia vulnerability
 - Webpage visit -> Guest to Host Ring0
 - VMWare ESX 3.0.1, CVE-2007-4496
 - Xen 3.0.3, CVE-2007-4993
 - CVE-2012-1516, VMWare ESXi 4.1 RPC events, arbitrary code execution.
 - Intel SYSRET privilege escalation, CVE-2012-0217
 - <http://blog.xen.org/index.php/2012/06/13/the-intel-sysret-privilege-escalation/>

End