Subverting Vista™ Kernel For Fun And Profit

Joanna Rutkowska
Advanced Malware Labs
COSEINC

SyScan'06
July 21st, 2006, Singapore
&
Black Hat Briefings 2006
August 3rd, 2006, Las Vegas

About this presentation

- This presentation is based on the research done exclusively for COSEINC Research
- This presentation has been first presented at SyScan conference in Singapore, on July 21st, 2006

Content

Part I

- loading unsigned code into Vista Beta 2 kernel (x64) without reboot

Part II

- Blue Pill – creating undetectable malware on x64 using Pacifica technology

Part I – getting into the kernel

Signed Drivers in Vista x64

- All kernel mode drivers must be signed
- Vista allows to load only signed code into kernel
- Even administrator can not load unsigned module!
- This is to prevent kernel malware and anti-DRM
- Mechanism can be deactivated by:
  - attaching Kernel Debugger (reboot required)
  - Using F8 during boot (reboot required)
  - using BCDEdit (reboot required, will not be available in later Vista versions)
- This protection has been for the first time implemented in Vista Beta 2 build 5384.

How to bypass?

- Vista allows usermode app to get raw access to disk
  - `CreateFile(\\.\C:)`
  - `CreateFile(\\.\PHYSICALDRIVE0)`)
- This allows us to read and write disk sectors which are occupied by the *pagefile*
- So, we can modify the contents of the pagefile, which may contain the code and data of the paged kernel drivers!
- No undocumented functionality required – all documented in SDK :)

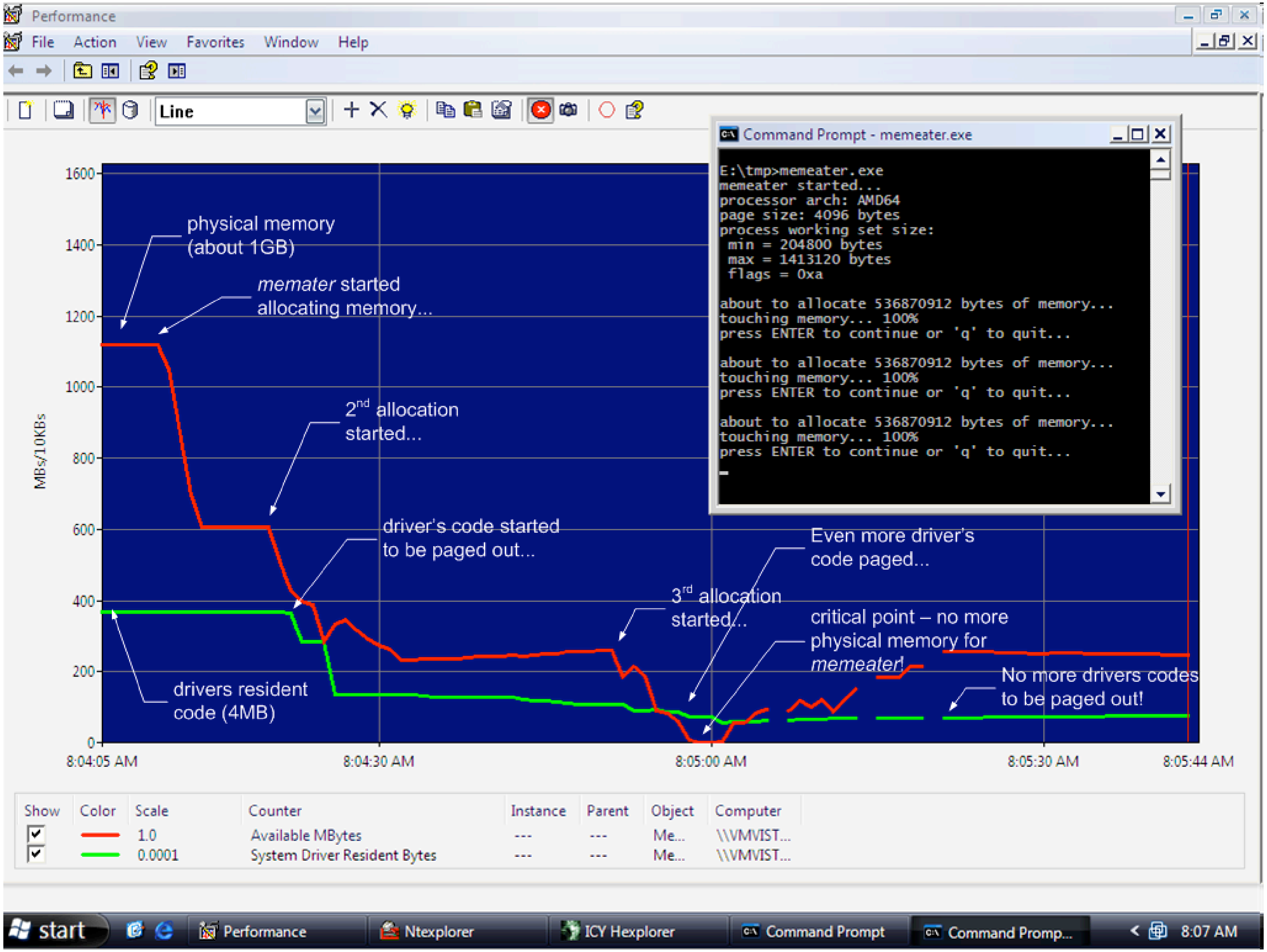
Challenges

- How to make sure that the specific kernel code is paged out to the pagefile?
- How to find that code inside pagefile?
- How to cause the code (now modified) to be loaded into kernel again?
- How to make sure this new code is executed by kernel?

How to force drivers to be paged?

- Allocate *lots of* memory for a process (e.g. using `VirtualAlloc()`)
- The system will try to do its best to back up this memory with the actual physical pages
- At some point there will be no more physical pages available, so the system will try to page out some unused code…
- Guess what is going to paged now… some unused drivers :)

Eating memory...



What could be paged?

- Pageable sections of kernel drivers (recognized by the section name starting with 'PAGE' string)
- Driver's data allocated from Non-Paged pool (e.g. `ExAllocatePool()`)

Finding a target

- We need to find some rarely used driver, which has some of its code sections marked as pageable…
- How about NULL.SYS?
- After quick look at the code we see that its *dispatch* routine is located inside a PAGE section – one could not ask for more :)
- It should be noted that there are more drivers which could be used instead of NULL – finding them all is left as an exercise to the audience ;)

Locating paged code inside pagefile

- This is easy – we just do a pattern search
  - if we take a sufficiently long binary string (a few tens of bytes) its very unlikely that it will appear more then once in a page file
- Once we find a patter we just replace the first bytes of the dispatch function with our shellcode
- The next slide demonstrates how to use disk editor to do that

How to make sure our shellcode gets executed?

- We need to ask kernel to be kind enough and execute our driver's routine (whose code we have just replaced in pagefile)
- In case of replacing driver's dispatch routine it's just enough to call `CreateFile()` specifying the target driver's object to be opened
- This will cause the driver's paged section to be loaded into memory and then executed!

Putting it all together

- Allocate lots of memory to cause unused drivers code to be paged
- Replace the paged out code (inside pagefile) with some shellcode
- Ask kernel to call the driver code which was just replaced

DEMO

- The above attack has been implemented in a form of a ‘1-click tool’
- Special heuristics has been used to automatically find out how much memory should be allocated, before ‘knocking the driver’
- The shellcode used in the demo disables signature checking, thus allowing any unsigned driver to be subsequently loaded

Creating useful shellcodes

- We can create a shellcode which would disable signature checking...
- … or we can create a small shellcode which would allocate some memory (via `ExAllocatePool`) and then “download” the rest of the malware from ring 3…

DEMO

Possible solutions (1/3)

- Solution #1: Forbid raw disk access from usermode.
- This would probably break lots of programs:
  - diskeditors/undeleters
  - some AV programs?
  - some data bases?
- Besides, access would still be possible from kernel mode
- So we can expect that lots of legal apps would provide their own drivers for raw disk access
- Those drivers would be signed of course, but could be used by attacker as well (no bug is required!).

Possible solutions (2/3)

- Solution #2: Encrypt pagefile!
- Generate encryption key while system starts and keep it in kernel non-paged memory. Do not write it to disk nor to the registry!
- Big (?) performance impact
- Encrypt only those pages which were paged from ring0, keep ring3 pages unencrypted
- Sounds better, still introduces some performance impact (not sure how much though)

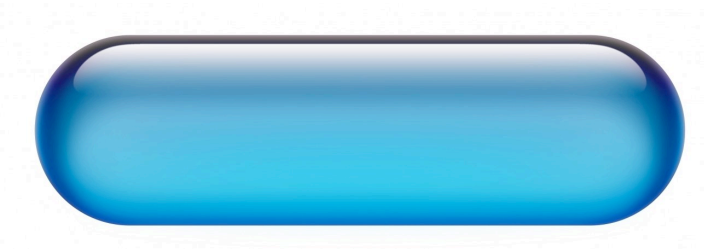
Possible solutions (3/3)

- Solution #3: Disable kernel memory paging!
- Disadvantage: wasting precious physical memory…
- On the other hand:
  - is RAM really so precious these days?
  - BTW, you can manually disable kernel memory paging in registry!
  - But it can be enabled again (reboot required), so it’s not a good solution.

Bottom line

- The presented attack does not rely on any implementation bug nor on any undocumented functionality
- MS did a good thing towards securing kernel by implementing signature check mechanism
- The fact that this mechanism was bypassed does not mean that Vista is completely insecure (it’s just not that secure as it’s advertised)
- It’s very difficult to implement a 100% efficient kernel protection in a general purpose operating system

Part II – Blue Pill



Invisibility by Obscurity

- Current malware is based on a concept...
- e.g. *FU* unlinks EPROCESS from the list of active processes in the system
- e.g. *deepdoor* modifies some function pointers inside NDIS data structures
- … etc…
- Once you know the *concept* you can write a detector!
- This is boring!

Imagine a malware...

- …which does not rely on a concept to remain undetected…
- …which can not be detected, even though its algorithm (concept) is publicly known!
- …which can not be detected, even though it's code is publicly known!

- Does this reminds you a modern crypto?

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

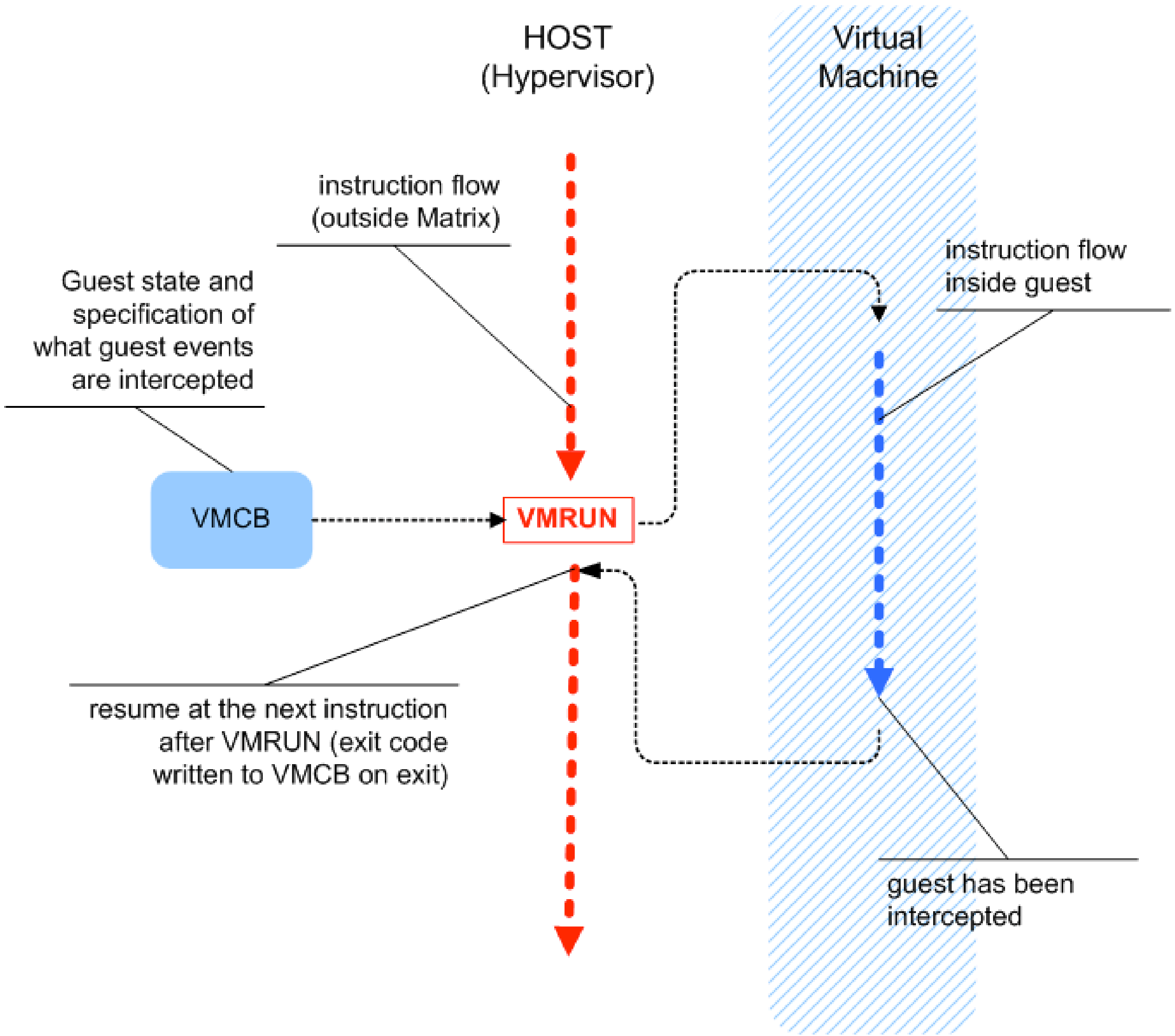
AMD64 & SVM

- Secure Virtual Machine (AMD SVM) Extensions (AKA Pacifica)
- May 23rd, 2006 – AMD releases Athlon 64 processors based on socket AM2 (revision F)
- AM2 based processors are the first to support SVM extensions
- AM2 based hardware is available in shops for end users as of June 2006

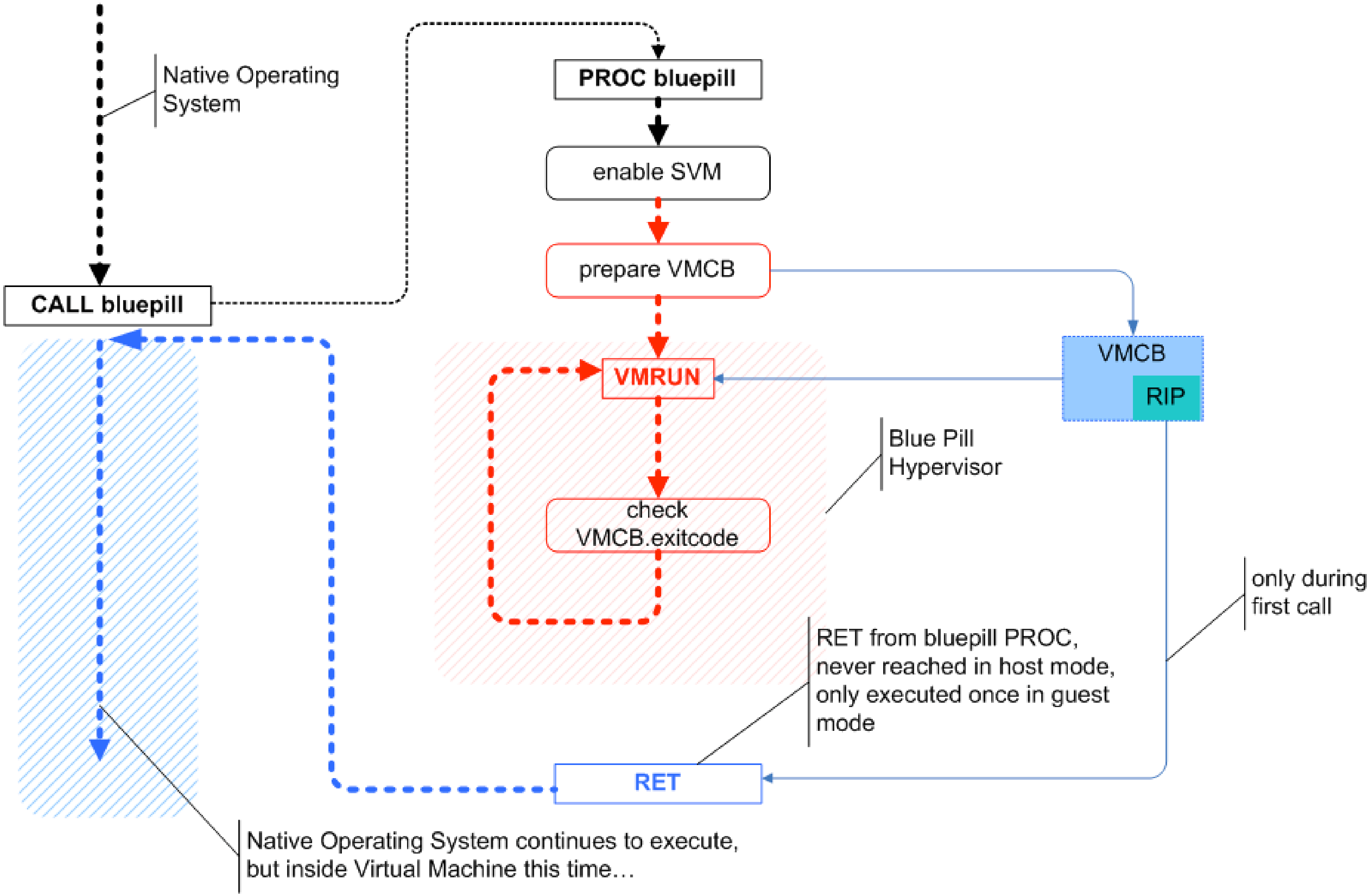
SVM

- SVM is a set of instructions which can be used to implement Secure Virtual Machines on AMD64
- MSR EFER register: bit 12 (SVME) controls weather 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

The heart of SVM: VMRUN instruction



Blue Pill Idea (simplified)



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
- But this is not a problem:
  - servers are rarely restarted
  - In Vista the 'Power Off' button does not shut down the system – it only puts it into stand by mode!
- And also we can intercept (this has not been yet implemented):
  - restart events (hypervisor survives the reboot)
  - shutdown events (emulated shutdown)

SubVirt Rootkit

- SubVirt has been created a few months ago by researches at MS Research and University of Michigan
- SubVirt uses commercial VMM (Virtual PC or VMWare) to run the original OS inside a VM

SubVirt vs. Blue Pill

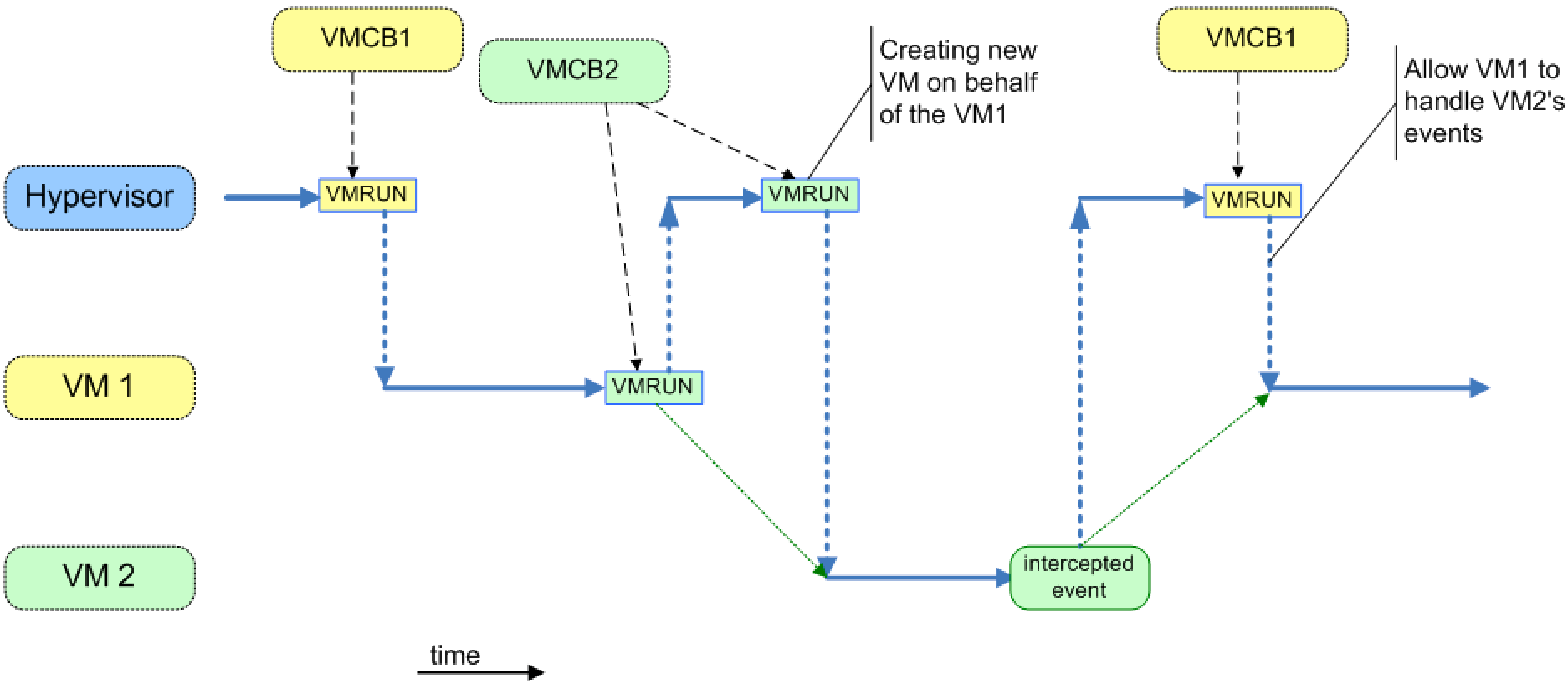
- SV is permanent! SV has to take control before the original OS during the boot phase. SV can be detected off line.
- SV runs on x86, which does not allow for full virtualization (e.g. SxDT attack)
- SV is based on a commercial VMM, which creates and emulates virtual hardware. This allows for easy detection

- Blue Pill can be installed on the fly – no reboot nor any modifications in BIOS or boot sectors are necessary. BP can not be detected off line.
- BP relies on AMD SVM technology which promises full virtualization
- BP uses ultra thin hypervisor and all the hardware is natively accessible without performance penalty

Matrix inside another Matrix

- What happens when you install Blue Pill inside a system which is already bluepilled?
- If nested virtualization is not handled correctly this will allow for trivial detection – all the detector would have to do was to try creating a test VM using a VMRUN instruction
- Of course we can cheat the guest OS that the processor does not support SVM (because we control MSR registers from hypervisor), but this wouldn't cheat more inquisitive users ;)
- So, we need to handle nested VMs…

Nested VMs


VMCB1
VMCB2
Creating new VM on behalf of the VM1
VMCB1
Allow VM1 to handle VM2's events
Hypervisor
VMRUN
VMRUN
VMRUN
VM 1
VMRUN
VM 2
intercepted event
time

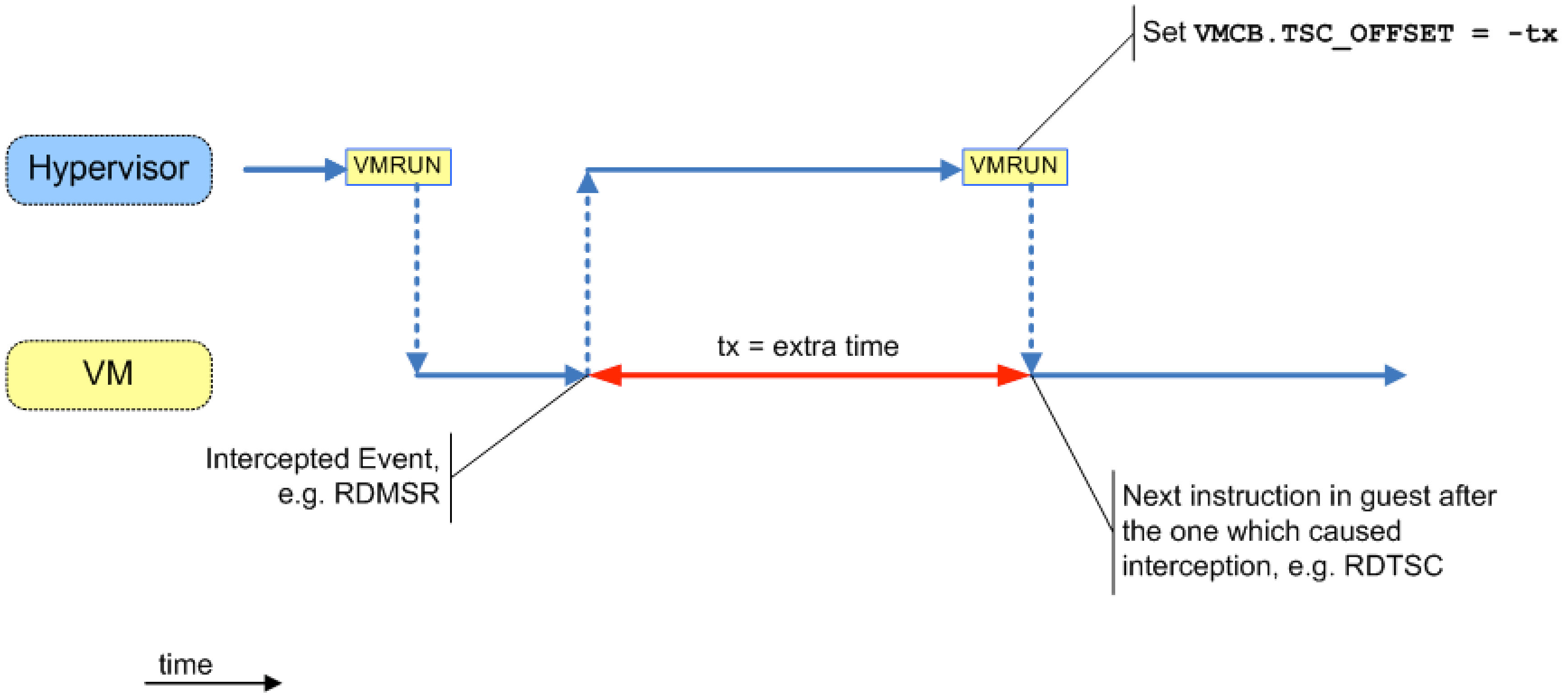
Detection via timing analysis

- We can assume that some of the instructions are always intercepted by the hypervisor
  - `VMMCALL`
  - `RDMSR` – to cheat about the value of `EFER.SVME` bit
- So, not surprisingly, the time needed to execute `RDMSR` to read the value of EFER would be different (longer) when running from guest
- Detector can execute such instructions a few millions of times and measure the time.

Cheating timing analysis

- The first problem is that detector (usually) does not have a base line to compare the results with…
- But even if it had (e.g. when testing access to physical pages and while one of them being backed up by disk) still we can cheat it!
- SVM offers a way to fool the guest's time stamp counter (obtained using RDTSC).
- All we have to do is to adjust VMCB.TSC_OFFSET accordingly before executing VMRUN (which resumes the guest)

Time dilatation for guest



Getting the real time…

Time profiling in practice

- Now imagine that you need to check 1000 computers in your company using the “external” stopwatch…
- Now imagine that you need to do this a couple of time every day…
- Time dilatation should make it impossible to write a self sufficient detector based on timing analysis!

- The challenge: we need a good ‘calibrating’ mechanism so that we know how much time to subtract.

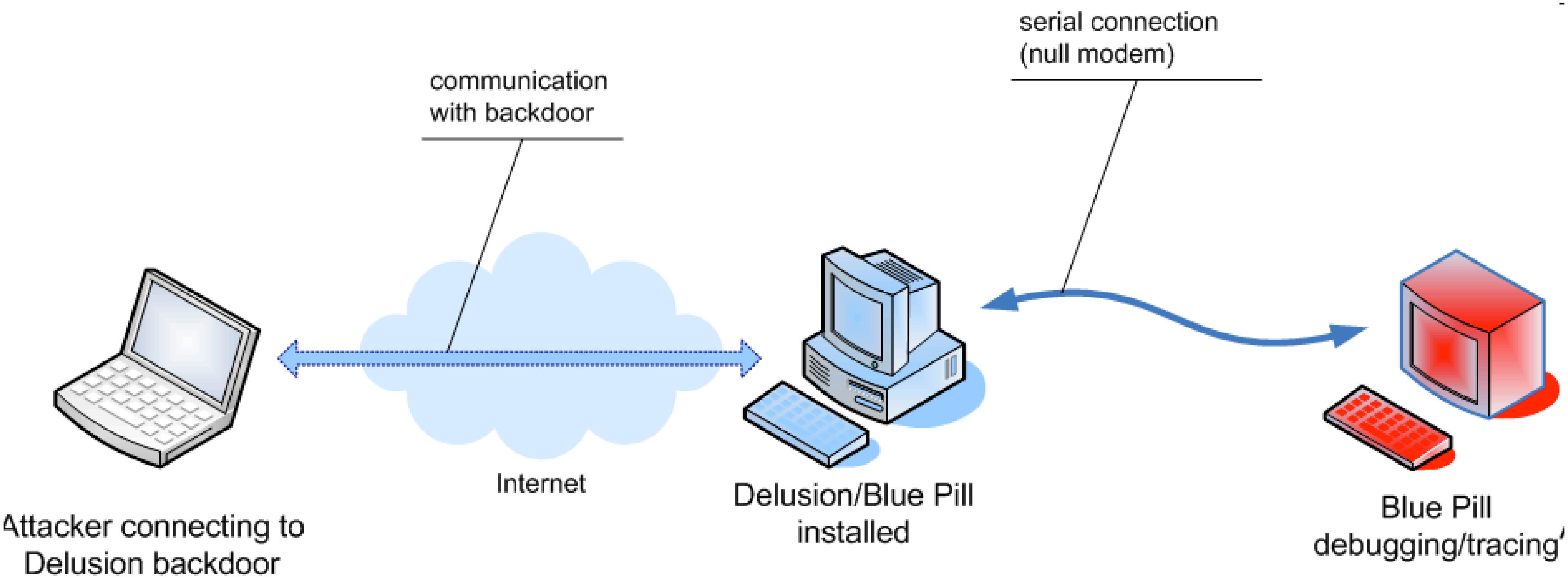
Blue Pill based malware

- Blue Pill is just a way of silently moving the running OS into Matrix on the fly
- BP technology can be exploited in many various ways in order to create stealth malware
- Basically ‘sky is the limit’ here :)
- On the next slides we present some simple example:

Delusion Backdoor

- Simple Blue Pill based network backdoor
- Uses two DB registers to hook:
  - `ReceiveNetBufferListsHandler`
  - `SendNetBufferListsComplete`
- Blue Pill takes care about:
  - handling #DB exception (no need for IDT[1] hooking inside guest) and protecting
  - protecting debug registers, so that guest can not realize they are used for hooking
- Not even a single byte is modified in the NDIS data structures nor code!
- Delusion comes with its own TCP/IP stack based on lwIP

Delusion Demo (Blue Pill powered)



Blue Pill detection

- Two level of stealth:
  - level 1: can not be detected even though the <u>concept</u> is publicly known (BPL1)
  - level 2: can not be detected even if the <u>code</u> is publicly known (BPL2)
- Level 1 does not requite BP's pages protection
- Level 2 is about avoiding signature based detection
- Level 2 is not needed in targeted attacks
- BPL2 has not been implemented yet!

Generic BP detection

- If we could come up with a generic program which would detect SVM virtual mode then…
- it would mean that SVM/Pacifica design/implementation does not support full virtualization!
- To be fair: AMD does not claim full virtualization in SVM documentation – it only says it is ‘Secure VM’… However it’s commonly believed that SVM == full virtualization…

Blue Pill detection

- We currently research some theoretical generic attacks against BPL1
- It seems that the attack would only allow for crashing the system if its bluepilled
- It seems that the only attack against BPL2 would be based on timing analysis (or crashing when some special conditions will be met, like e.g. user removing SATA disk in a specific moment during tests)

Pacifica vs. Vanderpool

- Pacifica (SVM) and Vanderpool (VT-x) are not binary compatible
- However they seem to be very similar
- XEN even implements a common abstraction layer for both technologies
- It *seems* possible to port BP to Intel VT-x

Blue Pill Prevention

- Disable it in BIOS
  - Its better not to buy SVM capable processor at all!
- Hypervisor built into OS
  - What would be the criteria to allow 3rd party VMM (e.g. VMWare or some AV product) to load or not?
  - Or should we stuck with "The Only Justifiable VMM", provided by our OS vendor? ;)
- Not allowing to move underlying OS *on the fly* into virtual machine
  - would not solve the problem of permanent, "classic" VM based malware
- or maybe another hardware solution…

Hardware Red Pill?

- How about creating a new instruction – **`SVMCHECK`:**

```
mov rax, <password>
svmcheck
cmp rax, 0
jnz inside_vm
```

- Password should be different for every processor
- Password is necessary so that it would be impossible to write a *generic* program which would behave differently inside VM and on a native machine.
- Users would get the passwords on certificates when they buy a new processor or computer
- Password would have to be entered to the AV program during its installation.

Future work

- Implement nested VMs
- Intercept restart and shutdown events (controlled restart, emulated shutdown)
- Support for multi-core processors
- Implement BPL1 using Intel VT-x
- Implement Blue Pill Level 2 (BPL2)
- Implement time dilatation for guest

Bottom line

- Arbitrary code can be injected into Vista x64 kernel
- This could be abused to create Blue Pill based malware on processors supporting virtualization
- BP installs itself on the fly and does not introduce any modifications to BIOS nor hard disk
- BP can be used in many different ways to create the actual malware – Delusion was just one example
- BP should be undetectable in any practical way (when fully implemented)
- Blocking BP based attacks on software level would also prevent ISVs from providing their own VMMs and security products based on SVM technology
- Changes in hardware (processor) could allow for easy BP detection

Credits

- Neil Clift for interesting discussions about Windows kernel
- Edgar Barbosa for preparing shellcode for the kernel strike attack
  - Edgar joined COSEINC AML at the end of June!
- Alexander Tereshkin AKA 90210 for thrilling discussions about Blue Pill detection
  - Alex is going to join COSEINC AML in August!

Thank you!

joanna@research.coseinc.com

check out http://coseinc.com/
for information about available trainings!