

# AMD-V for Hackers

SATOSHI TANDA (@STANDA\_T)

# Concept of Stealth Hooking 2

- ▶ Hooking: Modify code and execute arbitrary logic instead of, or on the top of original logic
  - ▶ Allows us monitoring and altering behavior of software
  - ▶ Very useful for reverse engineering software
  - ▶ Some detects that hooks are installed
    - ▶ PatchGuard, protectors
- ▶ Stealth: Make them invisible from the target, using SLAT

# Who I am

- ▶ Software Engineer at CrowdStrike
- ▶ Reverse Engineer
- ▶ Twitter: @standa\_t
- ▶ Speaker: Recon, BlueHat, Nullcon, CodeBlue
- ▶ Hiker, camper, runner, diver, cat lover, husband etc
  
- ▶ Creator of HyperPlatform – an open source hypervisor on Windows

# Motivation

# Hypervisors are...good

5

- ▶ Powerful
- ▶ Interesting
- ▶ Manageable

# Background

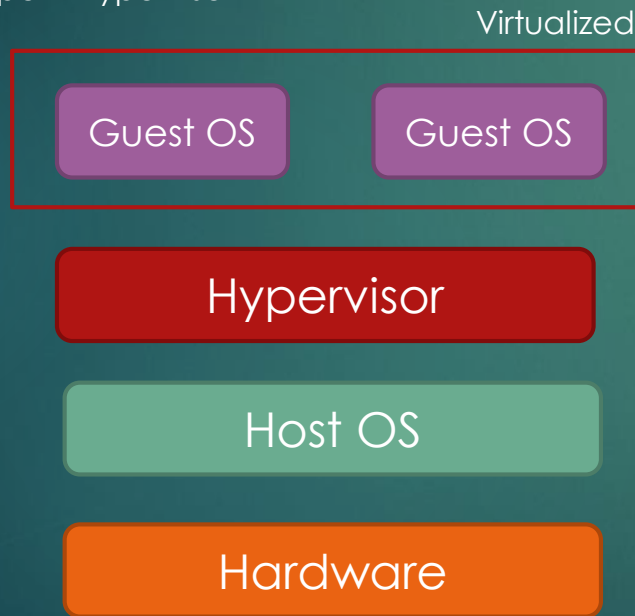
- ▶ HyperPlatform was created for the community
  - ▶ DdiMon, MemoryMon, GuardMon
- ▶ Lots of adoption and interests
  
- ▶ SimpleSvm was written for AMD users
- ▶ Lack of implementation of the useful ideas
- ▶ Can we have the feature parity with Intel VT-x based hypervisors?

# Basics of HW VT for Hackers

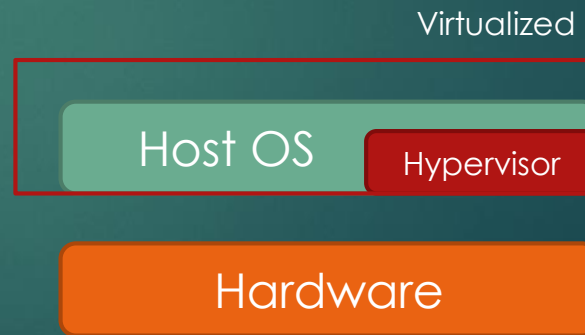
# Hypervisor as a tool

- ▶ Just a normal Windows driver (.sys)
- ▶ Can be developed with Visual Studio, tested in a VMware

Type-2 Hypervisor



Hyperjack-type Hypervisor





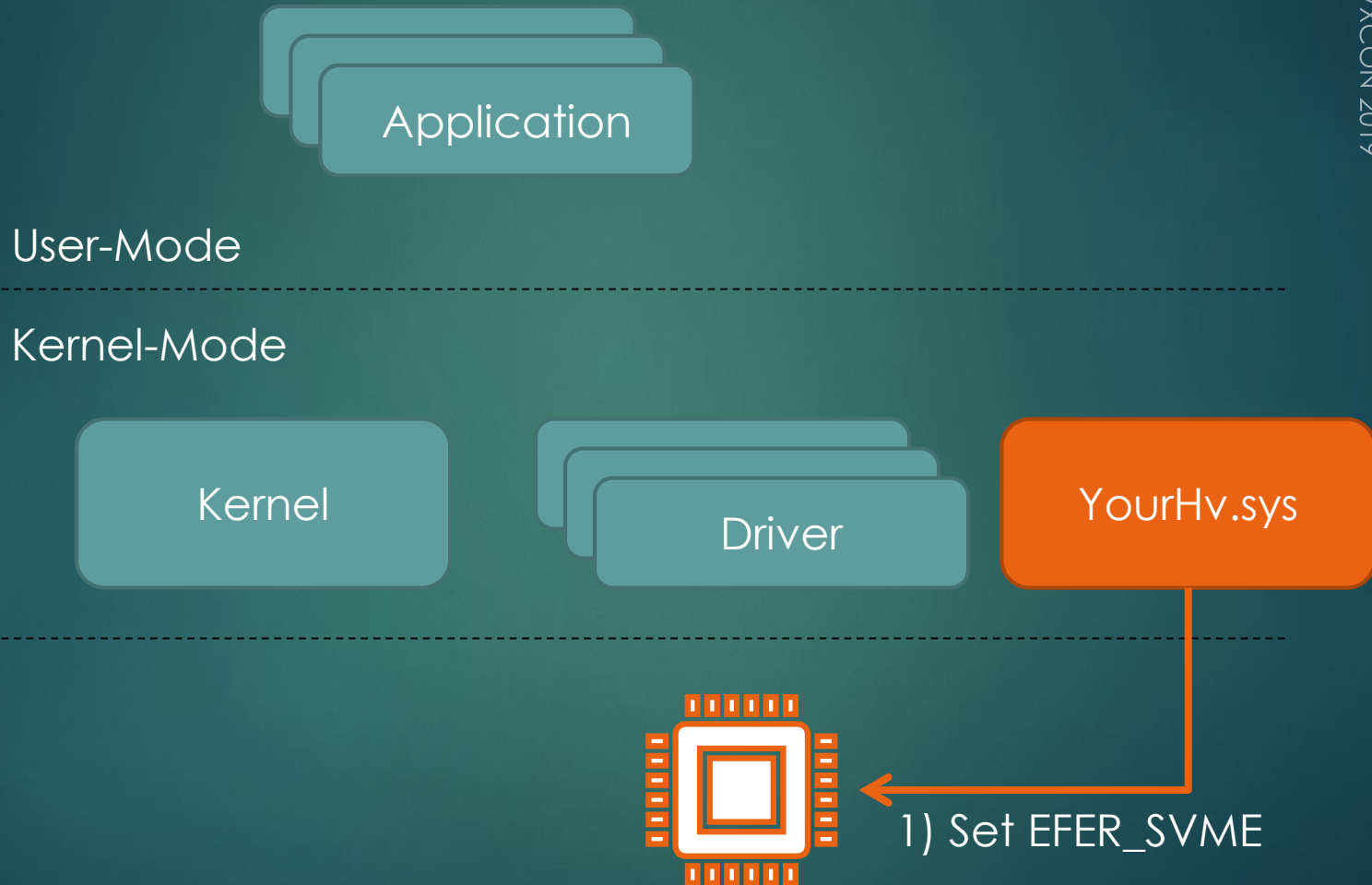
# Hypervisor as a tool

- ▶ Once loaded, the driver enables hardware-based virtualization technology (HW VT) of the processors;
  - ▶ Set `EFER_SVME` to `IA32_MSR_EFER` (Intel: VMXON)
- ▶ Sets up a context structure VMCB (Intel: VMCS); and
- ▶ Starts virtualization of the processors
  - ▶ `VMRUN` (Intel: VMLANCH)
- ▶ Processors trigger `#VMEXIT` (Intel: VM-exit) where your hypervisor handles

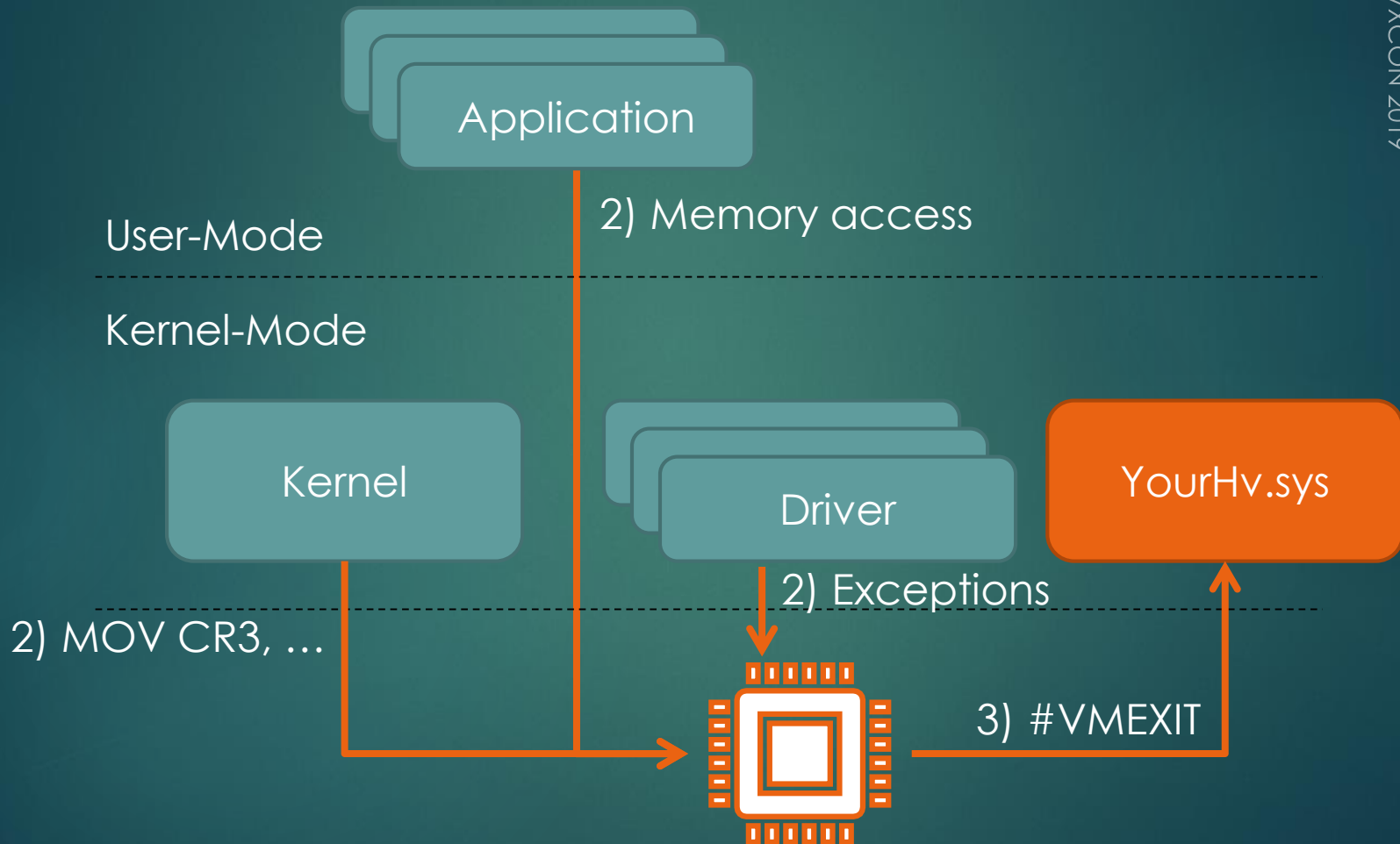
# Hypervisor as a tool

10

VXCON 2019



# Hypervisor as a tool



# Use cases

12

- ▶ System-wide debugger
  - ▶ Cheat Engine:DBVM
- ▶ PatchGuard, rootkit analysis
  - ▶ HyperBone
- ▶ Vulnerability hunting?

# SLAT: Second Level Address Translation

AKA, MEMORY VIRTUALIZATION

# What SLAT is

- ▶ SLAT allows a hypervisor to intercept and alter translation of virtual memory address (VA) to physical memory address (PA)
- ▶ An architecture agnostic term
- ▶ Called Rapid Virtualization Indexing (RVI), or Nested Page Tables (NPT)
  - ▶ Intel: Extended Page Tables (EPT)

# How SLAT works

15

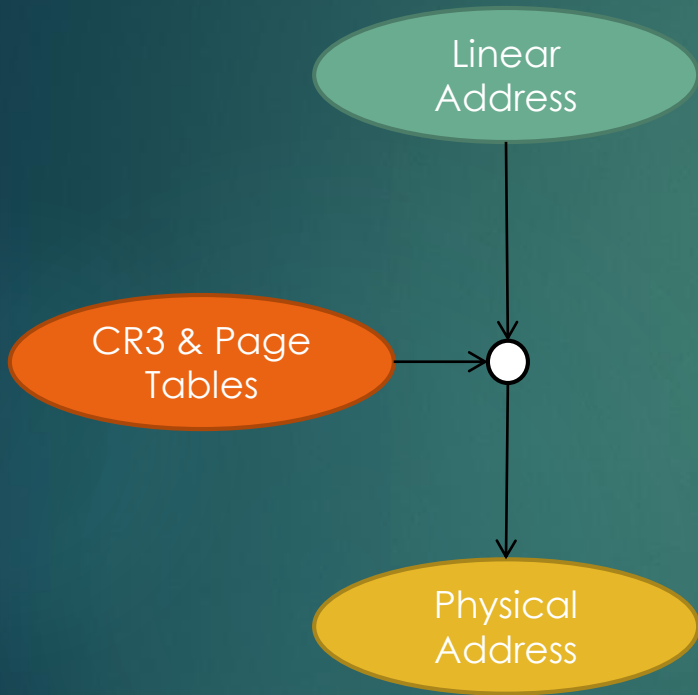
VXCON 2019

1. The processor performs the regular VA->PA translation with CR3 and page table structures
2. That PA is called Guest PA (GPA)
3. The processor performs GPA -> System PA (SPA) with nCR3 and NPTs
  - ▶ Intel: with the EPT pointer and EPTs

# Address Translation w/o VT

16

VXCON 2019

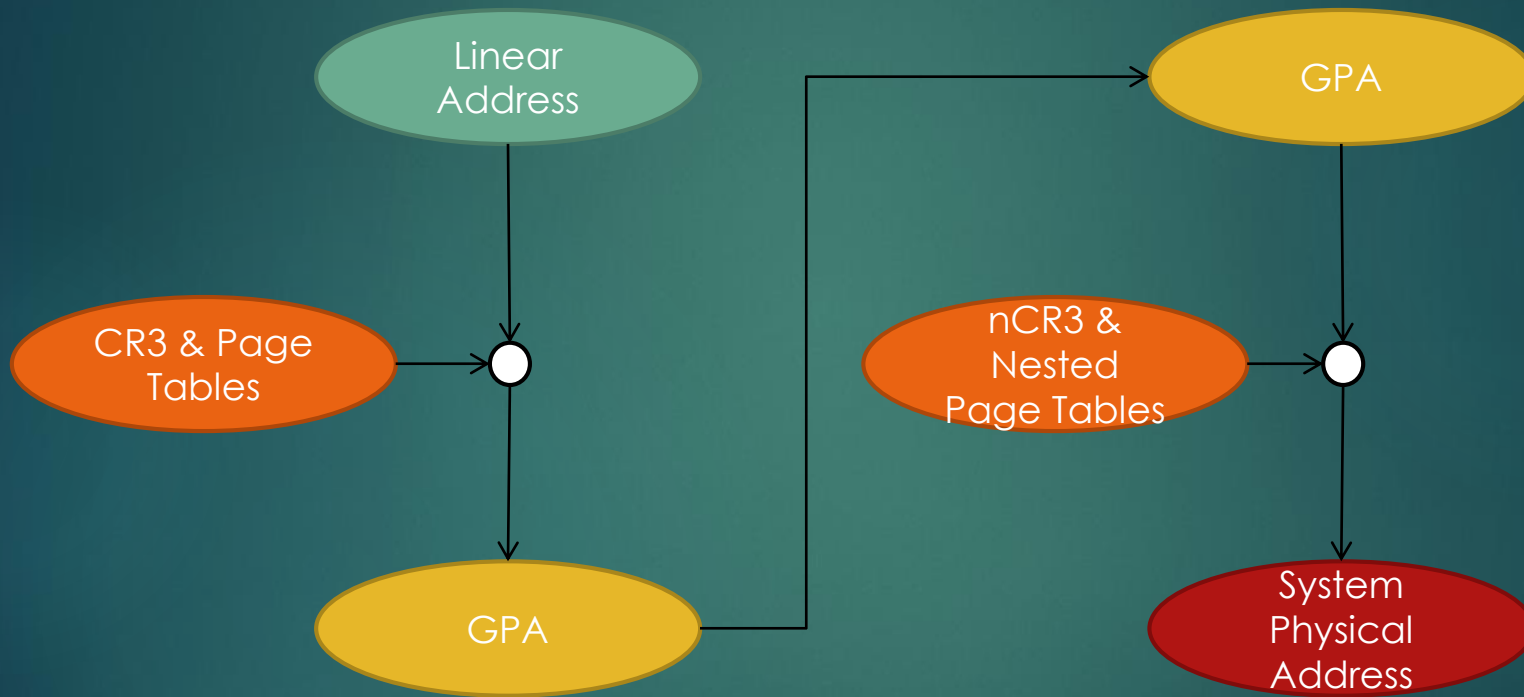




# Address Translation w/ VT

17

VXCON 2019



- ▶ Read/Write/Execute permissions can be configured like the regular PTEs

# Stealth Hooking with SLAT

# Concept

- ▶ Hooking: Modify code and execute arbitrary logic instead of, or on the top of original logic
  - ▶ Allows us monitoring and altering behavior of software
  - ▶ Very useful for reverse engineering software
  - ▶ Some detects that hooks are installed
    - ▶ PatchGuard, protectors
- ▶ Stealth: Make them invisible from the target, using SLAT

# Implementation


20

VXCON 2019

1. Set up two SLAT translations for the target GPA
  - ▶ Exec: translates to the SPA **with** hook
    - ▶ Not readable, not writable but executable (--X)
  - ▶ ReadWrite: translates to the SPA **without** hook
    - ▶ Readable, writable, but not executable (RW-)
2. At the runtime
  - ▶ On execute access, use Exec translation
    - ▶ The hook is executed
  - ▶ On read and write access, use ReadWrite translation
    - ▶ The hook is not visible and overwritable

# Implementation on Intel

- ▶ Assume ZwQuerySystemInformation is at 0xffff800`00000000 (VA) => 0xe000 (GPA)
- ▶ Hypervisor creates two translations:




GPA	SPA	Permission	Memory Contents
0xe000	0xe000	RW-	Without hook
0xe000	0xf000	--X	With hook

- ▶ Reading ZwQuerySystemInformation does not reveal hook

# Implementation on Intel

- ▶ On execution, VM-exit occurs due to access violation
  - ▶ Hypervisor switches translation and let the processor retry the same execution




GPA	SPA	Permission	Memory Contents
0xe000	0xe000	RW-	Without hook
0xe000	0xf000	--X	With hook

- ▶ Hook is executed

# Implementation on Intel

- ▶ On read, VM-exit occurs due to access violation
  - ▶ Hypervisor switches translation and let the processor retry the same execution



GPA	SPA	Permission	Memory Contents
0xe000	0xe000	RW-	Without hook
0xe000	0xf000	--X	With hook

- ▶ ZwQuerySystemInformation is read without hook

On AMD




# Challenges on AMD

25

VXCON 2019


- ▶ Not possible to set the execute-only permission
  - ▶ To be executable, the page must be readable too



GPA	SPA	Permission	Memory Contents
0xe000	0xe000	RW-	Without hook
0xe000	0xf000	<b>RWX</b>	With hook

# Challenges on AMD

- ▶ On execution, #VMEXIT occurs due to access violation
  - ▶ Hypervisor switches translation and let the processor retry the same execution




GPA	SPA	Permission	Memory Contents
0xe000	0xe000	RW-	Without hook
0xe000	0xf000	<b>RWX</b>	With hook

- ▶ Hook is executed
- ▶ The hook remains visible!

# Solution?

- ▶ Trigger #VMEXIT and switch translation to the non-executable version to hide the hook ASAP?
  - ▶ Eg, set eflags.TF, let the hypervisor handle it and switch translation?




GPA	SPA	Permission	Memory Contents
0xe000	0xe000	RW-	Without hook
0xe000	0xf000	<b>RWX</b>	With hook

- ▶ Performance impact is significant
  - ▶ Consider when the processor want to execute thousands of instructions on 0xe000
  - ▶ Every single instruction execution, access violation and eflags.TF trigger #VMEXIT

# Partial Solution

- ▶ The page must remain executable and readable for performance
  - ▶ ... while the page is being executed



GPA	SPA	Permission	Memory Contents
0xe000	0xe000	RW-	Without hook
0xe000	0xf000	<b>RWX</b>	With hook

- ▶ Switch translation when the processor goes outside the page

# Partial Solution


- ▶ Make all other pages non-executable while the hooked page is executed

GPA	SPA	Permission
0x0	0x0	RW-
0x1000	0x1000	RW-
0x2000	0x2000	RW-
...	...	...
0xe000	0xf000	RWX
...	...	...
0x8000`0000	0x8000`0000	RW-



# Partial Solution

- ▶ Make all other pages non-executable while the hooked page is executed




GPA	SPA	Permission
0x0	0x0	RW-
0x1000	0x1000	RW-
0x2000	0x2000	RW-
...	...	...
0xe000	0xf000	RWX
...	...	...
0x8000`0000	0x8000`0000	RW-

- ▶ When the processor go outside the page, #VMEXIT occurs
- ▶ Hypervisor restores permissions of the pages

# Partial Solution

- ▶ Make all other pages non-executable while the hooked page is executed




GPA	SPA	Permission
0x0	0x0	RWX
0x1000	0x1000	RWX
0x2000	0x2000	RWX
...	...	...
0xe000	0xf000	RWX
...	...	...
0x8000`0000	0x8000`0000	RWX

▶ When the processor go outside the page, #VMEXIT occurs

▶ Hypervisor restores permissions of the pages

# Partial Solution

- ▶ Make all other pages non-executable while the hooked page is executed



GPA	SPA	Permission
0x0	0x0	RWX
0x1000	0x1000	RWX
0x2000	0x2000	RWX
...	...	...
0xe000	0xe000	RW-
...	...	...
0x8000`0000	0x8000`0000	RWX

▶ When the processor go outside the page, #VMEXIT occurs


▶ Hypervisor restores permissions of the pages; and,

▶ Hide the hook



# Limitation

- ▶ Hook remains visible from code in the same page



GPA	SPA	Permission
0x0	0x0	RW-
0x1000	0x1000	RW-
0x2000	0x2000	RW-
...	...	...
0xe000	0xf000	RWX
...	...	...
0x8000`0000	0x8000`0000	RW-

- ▶ Further partial solution for the limitation: as soon as hook is executed, switch to the page **without** hook (ie, even before jumping out to the other page). Downside is that hook is not executed from code in the same page + little more perf cost due #VMEXIT for page switching.

# Performance Issue

# Naïve Implementation

35

- ▶ 524,288 -1 NTP entries must be updated to change permission of all pages on a system with 2GB RAM
  - ▶  $2\text{GB} = 0x8000`0000 = 0x8000`0000 / 0x1000 = 0x8`0000$
- ▶ Significant performance impact
- ▶ Protip: Measure performance. ALWAYS.

# Optimization 1

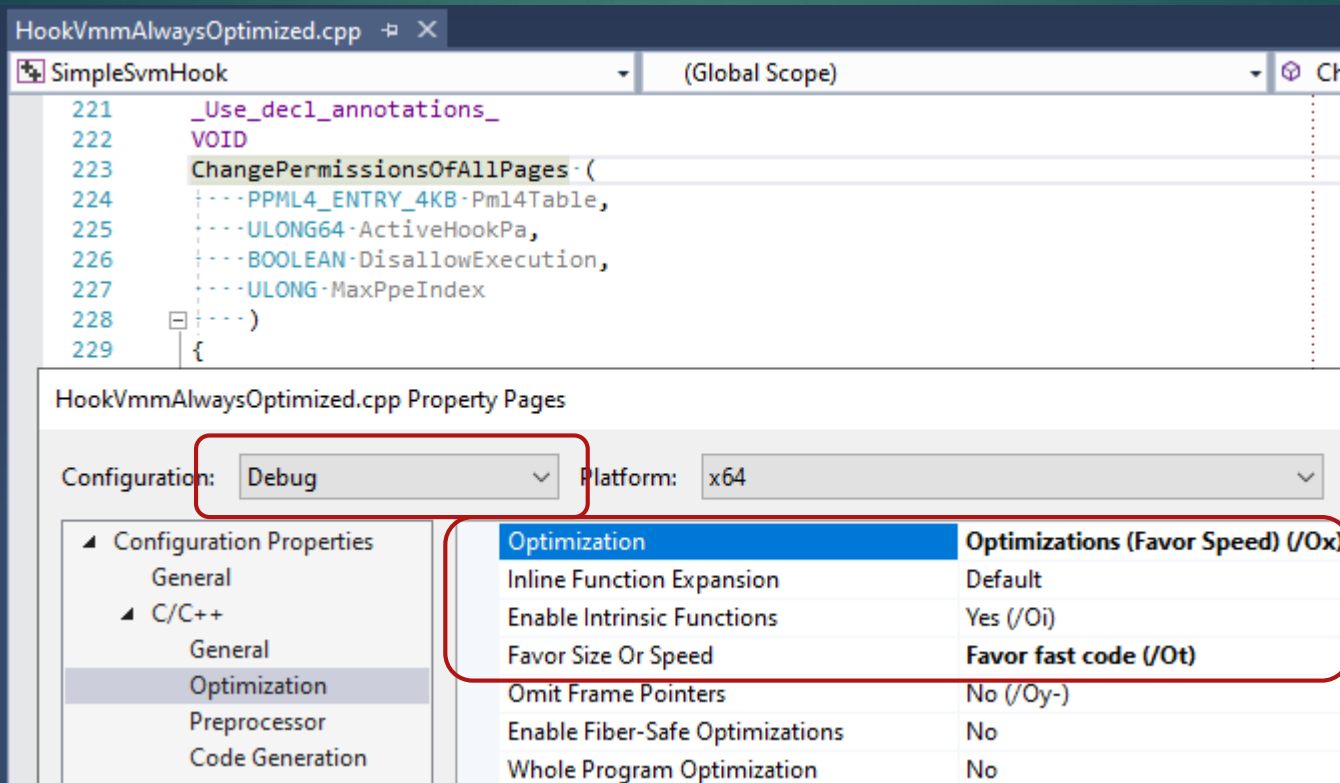
36

VXCON 2019

- ▶ Use the nesting NPT structures
  - ▶ Like the regular page structures, NPT is made up of PML4, PDP, PD, PT
  - ▶ Updating the permission on PML4 entry changes the permission of entire 512GB
    - ▶ PDP=1GB, PD=2MB, PT=4KB
- ▶ Update the upper level NPT entries where possible

# Optimization 2

- ▶ Enable optimization of NTP manipulation code even on Debug build



- ▶ Tip: such pure algorithmic code can be tested with an UM project separately

# Optimization 3

38

VXCON 2019

- ▶ Keep the number of hooks one on VMware
- ▶ NTP manipulation on VMware is very (VERY) slow
  - ▶ Performance measurement on VMware is pointless
- ▶ To measure performance and impact of optimization, use a bare metal machine

Demo

# Conclusion



# Takeaways

- ▶ Implementation of stealth hooking on AMD processors are possible
- ▶ But beware of the limitations:
  - ▶ Hooks remain visible from code in the same page
    - ▶ Or hooks are not called from code in the same page
  - ▶ Performance overhead is higher than similar implementation for Intel processors due to extensive manipulation of NTP entries
- ▶ Any performance measurement must be done on bare metal

# Resources

42

VXCON 2019

- ▶ Learn more about writing hypervisors on AMD
- ▶ <https://github.com/tandasat/SimpleSvm>
- ▶ And implementation of stealth hooking
- ▶ <https://github.com/tandasat/SimpleSvmHook>

# Thank you

FOR VXCON CREW, AND ALL OF YOU LISTENING <3