

29<sup>th</sup> June 2024 – REcon 2024

# Hypervisor-enforced Paging Translation

The end of non data-driven Kernel Exploits?

Andrea Allievi (@aall86)  
Satoshi Tanda (@standa\_t)

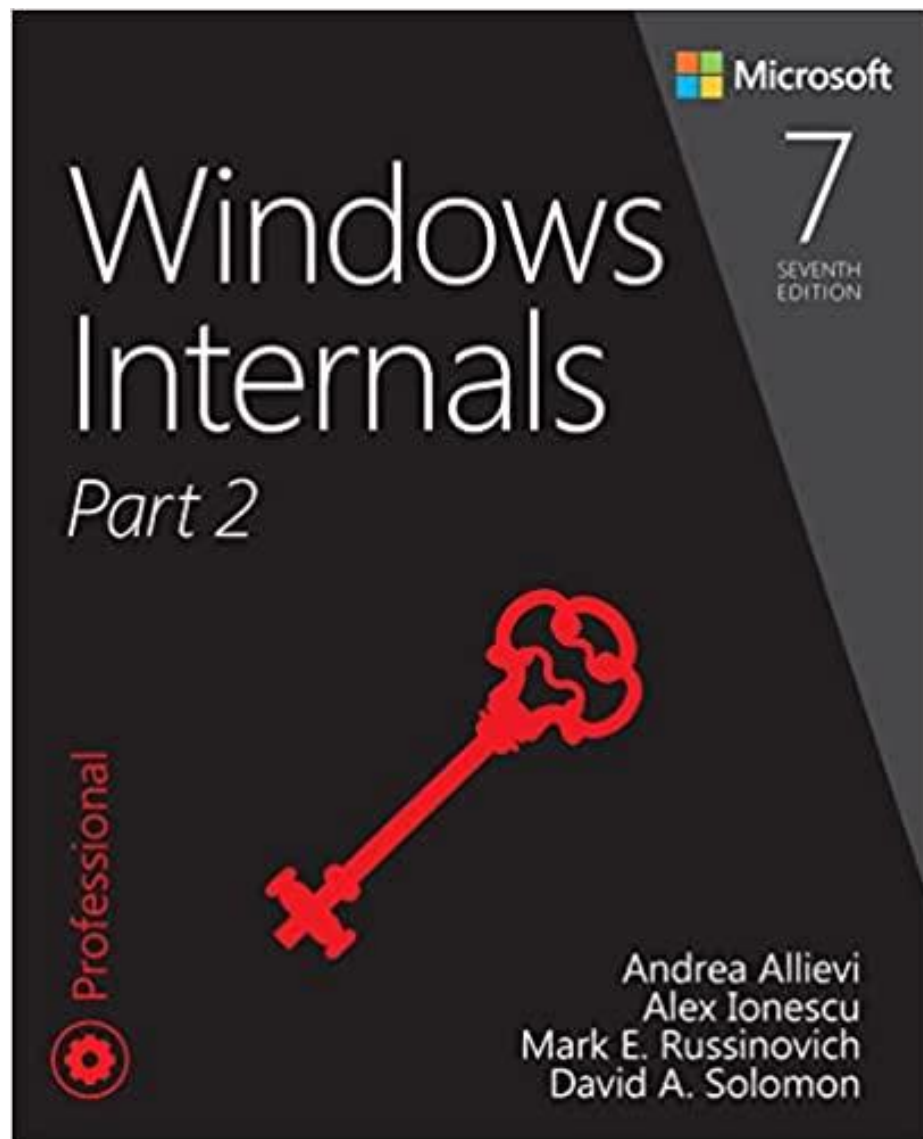
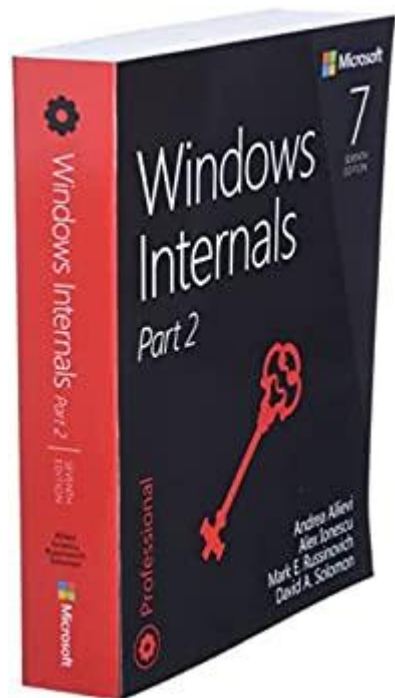


# Who I am



- Italian CORE OS Architect, mainly focused on Windows Kernel developing. Trainer at Pearson / O'Reilly.
- Microsoft OSs Internals enthusiast / Security Researcher
- Work for the Security Core OS Team of Microsoft Ltd
- Previously worked in the Threat Intelligence Center of Microsoft (MSTIC), in the TALOS Security Group of Cisco Systems, Webroot and Saferbytes
- Original designer of the first UEFI Bootkit, Patchguard 8.1 bypass in 2014, Windows Intel PT Driver, and many other research projects...
- Since 2016 I am responsible of the Secure Kernel research and development
- Main developer and contributor of Retpoline and Import Optimization, KDP, Function Overrides, HLAT and many others Kernel features and tools.
- AND ...

One of the  
main authors ☺



# Who I am

- Satoshi Tanda

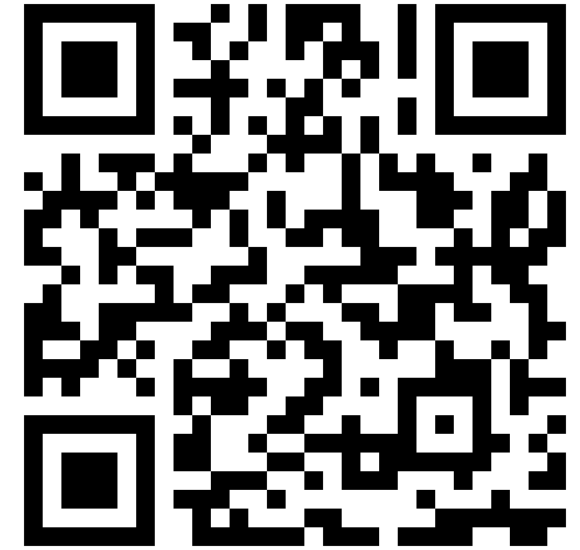
 [in/satoshitanda](https://www.linkedin.com/in/satoshitanda)

 [@standa\\_t](https://twitter.com/standa_t)

 [@tandasat](https://github.com/tandasat)

 [@satoshi\\_tanda@infosec.exchange](mailto:@satoshi_tanda@infosec.exchange)

- System software dev – virtualization for game console
- Security researcher - specializes in platform security
- Trainer – teaches a [hypervisor development course](#)



BIO and training course

# Outline – What we are going to talk...

## Part 1 - Introduction

- State of arts of the latest Exploit mitigations technologies in Windows
- Primarily focused on KDP and Shadow stack (CET)

## Part 2 – Attacks on the page tables and HLAT

- Limitation of KDP and HVPT as a solution

## Part 3 – The NT Implementation

- Private vs Shared pages - How the Secure Kernel tracks NT Memory
- NTEs – Normal Table Address
- How the black magic work – the born of HVPT
- **[BONUS]** Session space – Why a heck MS took so long to implement HVPT

# Part 1 – Introduction

- During last years, multiple mitigations were introduced in Windows, with the idea to “limit the damage” or make exploitation of bugs hard or impossible.
- In Windows, the majority of them are supported thanks to the Secure Kernel.
- Kernel CFG, Hardware MBEC are good examples, KDP and Shadow Stack (ARM64 Pointer authentication) are the latest...
- Due to time constraints, we will not discuss all of them, but we will just introduce the Kernel mitigations that have led to the design of HVPT
- Our talk is about a new Kernel mitigation, the User-mode world is **not** discussed.
- Multiple talks and documentation already available in the wild.. (Windows Internals book included 😊)

... Let's start with a recap ...

# ROP (Return-Oriented Programming) & CET

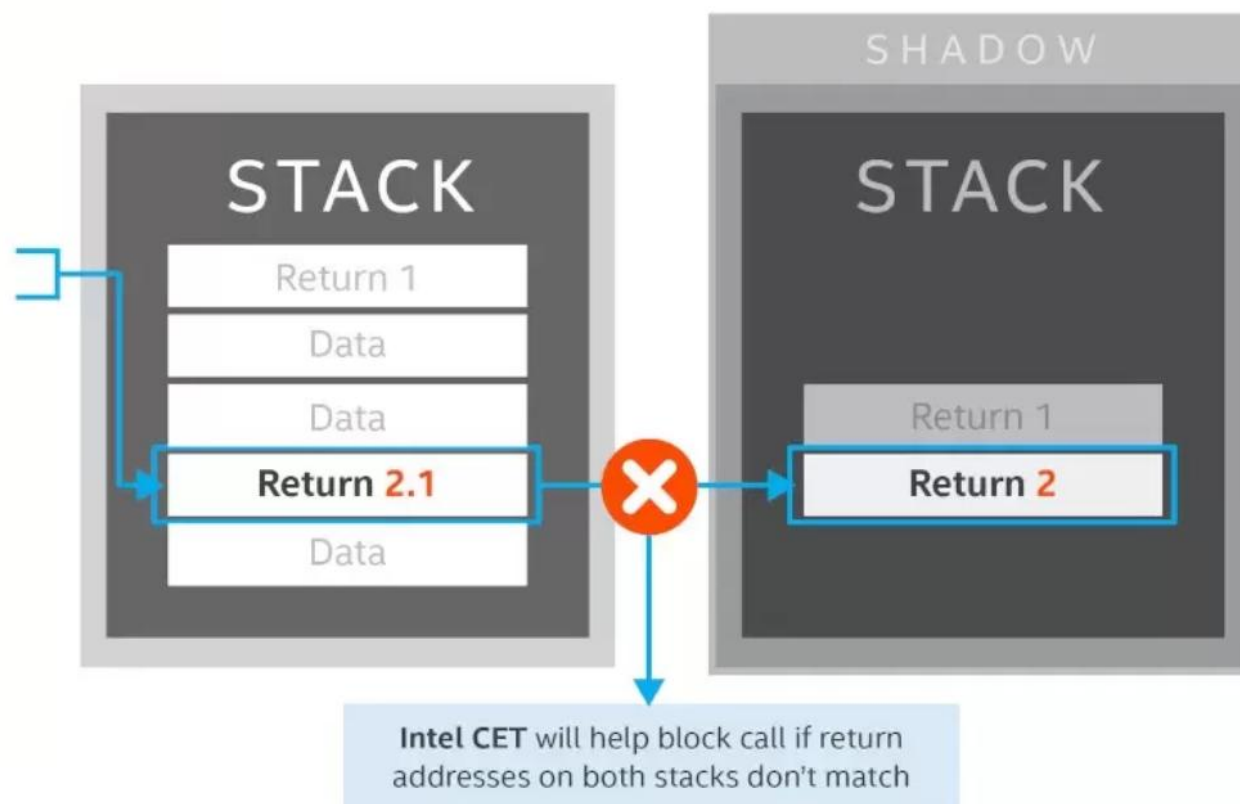
- Since HVCI, code section of kernel modules can not be forged / touched anymore
- Attackers started to target stacks, with the goal to find gadgets for a ROP chain
- How to **stop** ROP attacks?
- In 2019 Intel introduced Control Enforcement Technology (**CET**), an-hardware assisted technology which creates another “shadow” stack, containing only “branch” addresses (and a token).
- The shadow stack is not accessible by software executing at the same privilege level\*
- When returning from a function call, the processor compares the return address extracted from the regular stack with the value popped from the Shadow stack. If the two values do not match a Control Protection Exception (**#CP**) is raised.
- In Windows, CET is enabled in **both** user and kernel, which means the **end** of ROP attacks.



# ROP (Return-Oriented Programming) & CET

## SHADOW STACK (SS)

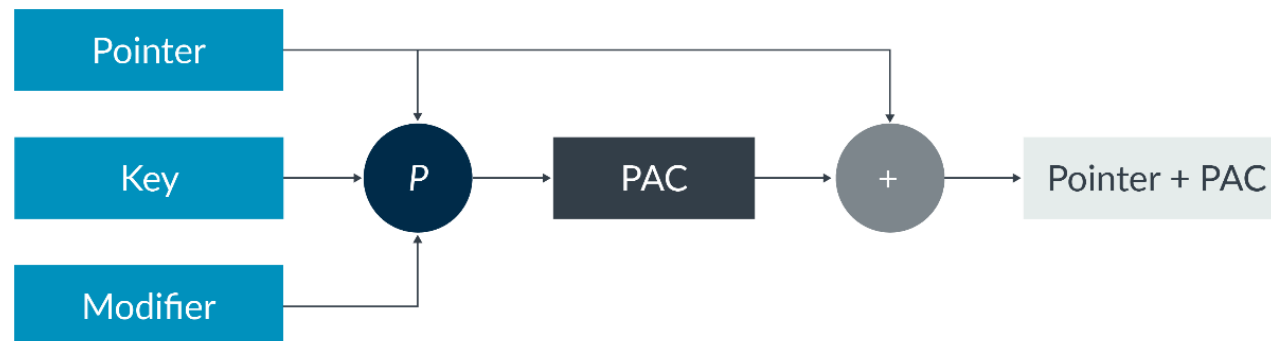
SS delivers return address protection to defend against return-oriented programming (ROP) attack methods.



\*Image courtesy of Phoronix  
(I am too lazy to draw it myself)

# Pointer Authentication

- What about ARM64? Another hardware technology exists and can be used to protect against ROP attacks: Pointer authentication.
- Specs were written before Intel CET in ARMv8.3, but its first adaption came later.
- Pointer authentication exploits the unused bits of a VA to store a signature referred as Pointer authentication code (PAC).
- PAC is calculated via a cryptographic algorithm, based on a **key** and a **modifier**.
  - Keys are usually generated by the OS and are secrets
  - Modifiers are regularly another register value, which ties the PAC to a particular value
  - Using SP as modifier, gives you PAC that is only valid at the entrance and exit of a function.



# Pointer Authentication

- On function **entry**, the compiler always emit the PACIBSP opcode, which stands for PACIB LR, SP  
Encode the pointer contained in the Link Register with the secret key and the modifier value contained in SP (stack pointer)
- On function **exit**, the compiler emits a AUTIBSP instruction, which decodes the pointer contained in LR (popped from the regular stack) with the secret key and the SP modifier.
- If the authentication fails, an invalid pointer is returned, leading to an **instant crash**.

```
nt!MiCreateNewSection:
fffff802`007b1b18 7f2303d5 pacibsp
fffff802`007b1b1c fd7bbaa9 stp    fp, lr, [sp, #-0x60]!
fffff802`007b1b20 f35301a9 stp    x19, x20, [sp, #0x10]
fffff802`007b1b24 f55b02a9 stp    x21, x22, [sp, Segment(!) (sp+20h)]
fffff802`007b1b28 f76303a9 stp    x23, x24, [sp, ResidentImagePages(!) (sp+30h)]
fffff802`007b1b2c f96b04a9 stp    x25, x26, [sp, OptionalHeader{.ImageBase(!)} (sp+40h)]
fffff802`007b1b30 fb2b00f9 str    x27, [sp, OptionalHeader.SizeOfImage(!) (sp+50h)]
fffff802`007b1b34 fd030091 mov    fp, sp
fffff802`007b1b38 1a46f197 bl     ntkrnImp!_security_push_cookie (fffff802004033a0)
fffff802`007b1b3c ffc302d1 sub    sp, sp, #0x80

Breakpoint 0 hit
nt!MiCreateNewSection+0x4:
fffff802`007b1b1c a9ba7bfd stp    fp, lr, [sp, #-0x60]!
1: kd> r lr
lr=70c17802007b1754

nt!MiCreateNewSection+0x580:
fffff802`007b2098 f55b42a9 ldp    x21, x22, [sp, #0x20]
fffff802`007b209c f35341a9 ldp    x19, x20, [sp, #0x10]
fffff802`007b20a0 fd7bc6a8 ldp    fp, lr, [sp], #0x60
fffff802`007b20a4 ff2303d5 autibsp
fffff802`007b20a8 c0035fd6 ret
fffff802`007b20ac 1f2003d5 nop

Breakpoint 1 hit
nt!MiCreateNewSection+0x590:
fffff802`007b20a8 d65f03c0 ret
1: kd> r lr
lr=fffff802007b1754
```

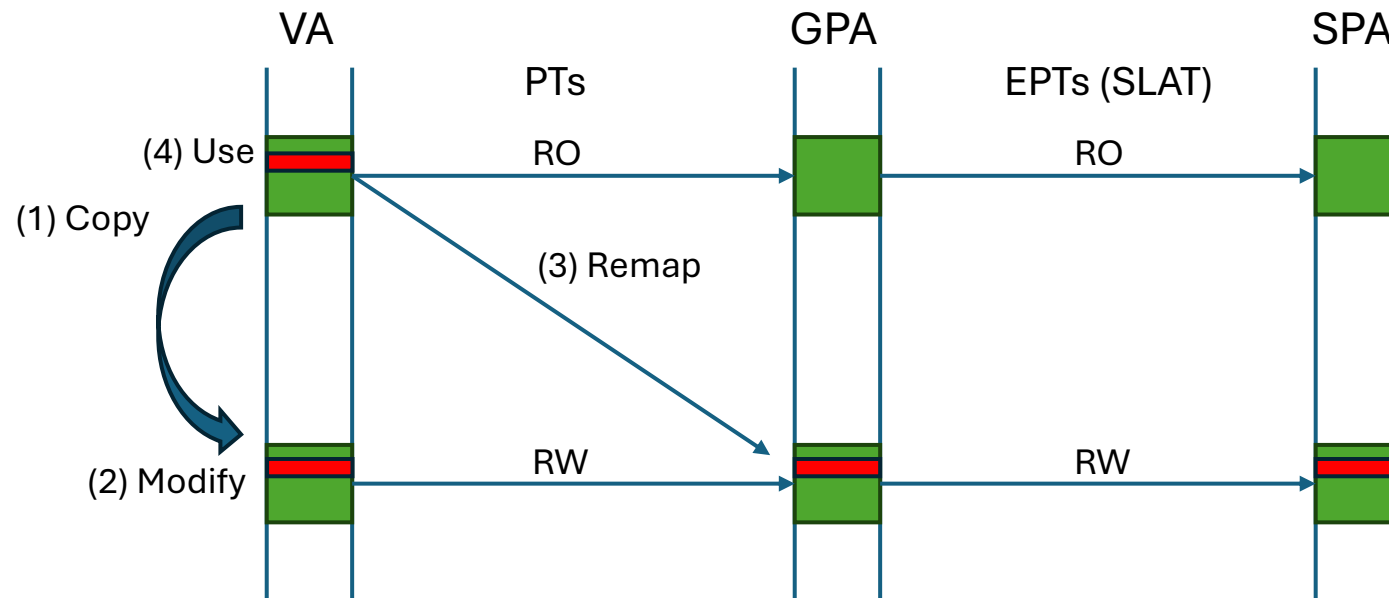
# Kernel Data Protection (KDP)

- What about **data**?
  - Kernel modules have often important data fields that should be protected too.
  - Drivers data structures, globals and static variables are stored in read-only (or writable) **data sections** (while stack variables are usually stored in kernel stacks).
  - Good example of kernel module that has important data to protect is **CI** – Code Integrity
- HLAT, CET or Pointer authentication can **not** help with volatile data.
- We designed another protection called Kernel data protection (**KDP**).
  - NT asks SK to apply SLAT memory protection also to Data pages\*
  - Implemented in two sides: Static and Dynamic (deprecated) KDP
  - Static KDP is implemented in NT and SK. Single entry point: **MmProtectDriverSection**.
  - Kernel modules that have protected sections are by default **not unloadable**, unless the MM\_PROTECT\_DRIVER\_SECTION\_ALLOW\_UNLOAD flag is specified.
- KDP is well documented in [MSDN](#) and in the [Windows Security blog](#).

... Now let's move on and introduce HLAT ...

# Part 2 – Remapping attacks and HLAT

- KDP enforces RO permission on GPA, not VA
- VA -> GPA translation is controlled by the guest
- Remapping attack: Guest can remap VA to another unprotected GPA



# Demo – Zeroing ci!g\_CiOptions under KDP

The screenshot shows a WinDbg window with the Command tab selected. The command `!pte ci!g_cioptions` is entered, and the output displays four memory entries with their physical addresses and contents. A red circle labeled '1' is around the command, and a red circle labeled '2' is around the output. Below the WinDbg window, a Command Prompt window shows the execution of `recon2024_demo.exe` with three commands: `--read64 fffff8022fa8b004`, `--write64 FFFFF27C0117D458 8900000106fcd121`, and `--read64 fffff8022fa8b004`. A red circle labeled '3' is around the first command, and a red circle labeled '2' is around the second command. Red arrows connect the circles in the Command Prompt to the corresponding parts in the WinDbg window.

```
0: kd> !pte ci!g_cioptions
VA fffff8022fa8b004
PXE at FFFFF2793C9E4F80 PPE at FFFFF2793C9F0040 PDE at FFFFF2793E008BE8 PTE at FFFFF27C0117D458
contains 00000000007F7063 contains 00000000007F7063 contains 0000000000882063 contains 89000001016D1121
pfn 7f7 ---DA--KWEV pfn 7f6 ---DA--KWEV pfn 882 ---DA--KWEV pfn 1016d1 -G--A--KR-V

0: kd> 8900000106fcd121
```

```
C:\Release>recon2024_demo.exe --read64 fffff8022fa8b004
0xfffff8022fa8b004: 0x1c006

C:\Release>recon2024_demo.exe --write64 FFFFF27C0117D458 8900000106fcd121

C:\Release>recon2024_demo.exe --read64 fffff8022fa8b004
0xfffff8022fa8b004: 0x0
```

- <https://youtu.be/taeyoq70nBA>

# Applications of remapping

- Modify contents of VA 👉 change sensitive global vars
- Modify contents of code 👉 execute different code
  - HEXACON2023 - Bypassing the HVCI memory protection by Viviane Zwanger and Henning Braun ([Recording](#))
  - ⚠️ an attacker cannot change (generate) code. Can only re-purpose existing exec. pages

```
Disassembly x
Address: ci!xGetDR7Value [x] Follow current instruction
fffff806`67ac290b cc      int      3
fffff806`67ac290c cc      int      3
fffff806`67ac290d cc      int      3
fffff806`67ac290e cc      int      3
fffff806`67ac290f 90      nop
CI!xGetDR7Value:
fffff806`67ac2910 0f21f8 mov     rax, dr7
fffff806`67ac2913 c3      ret
fffff806`67ac2914 cc      int      3
fffff806`67ac2915 cc      int      3
fffff806`67ac2916 cc      int      3
fffff806`67ac2917 cc      int      3
```



```
Disassembly x
Address: ci!xGetDR7Value [x] Follow current instruction
fffff806`67ac2906 0800    or      byte ptr [rax], al
fffff806`67ac2908 0000    add     byte ptr [rax], al
fffff806`67ac290a 0000    add     byte ptr [rax], al
fffff806`67ac290c 0000    add     byte ptr [rax], al
fffff806`67ac290e 0000    add     byte ptr [rax], al
CI!xGetDR7Value:
fffff806`67ac2910 a7      cmps    dword ptr [rsi], dword ptr [rax]
fffff806`67ac2911 0100    add     dword ptr [rax], eax
fffff806`67ac2913 0800    or      byte ptr [rax], al
fffff806`67ac2915 0000    add     byte ptr [rax], al
fffff806`67ac2917 0000    add     byte ptr [rax], al
fffff806`67ac2919 0000    add     byte ptr [rax], al
```



# Introduction to VT-rp (redirect protection)

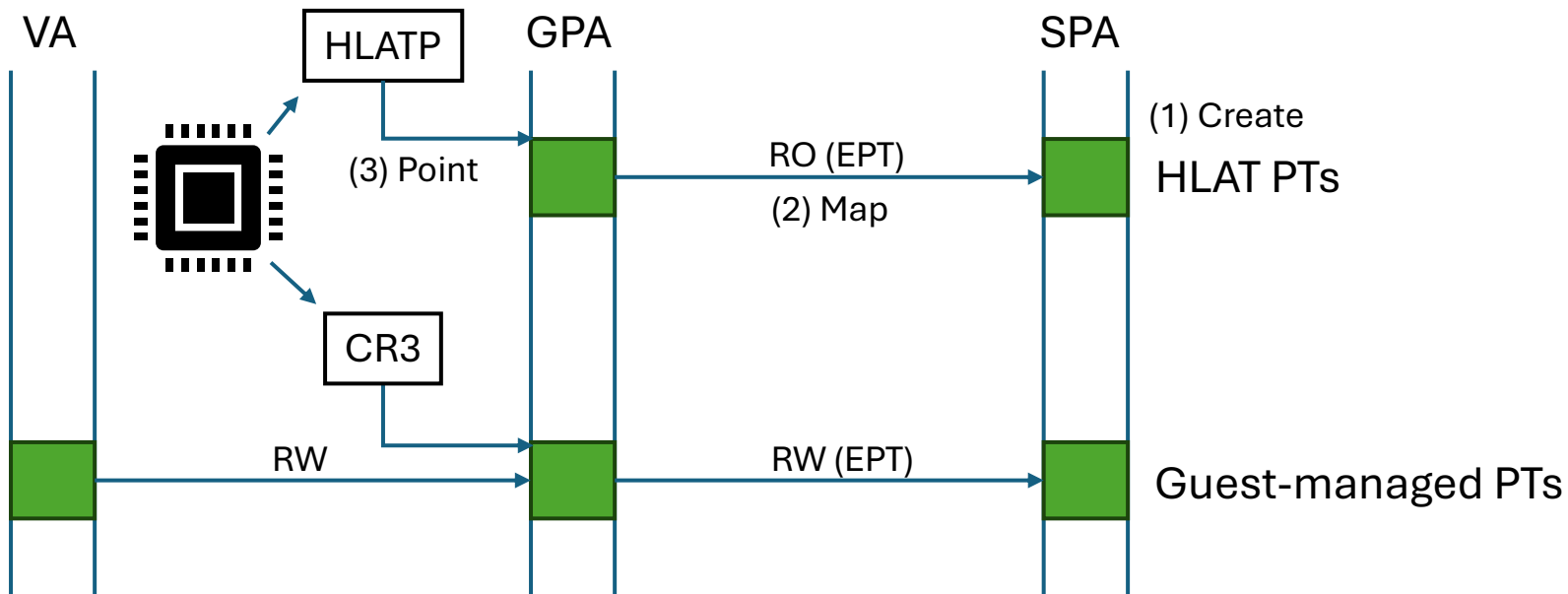
- 3 features: HLAT, PW, GPV
- Available in the subset of Intel 12+ gen
- HLAT is the focus of this talk

**Table 25-8. Definitions of Tertiary Processor-Based VM-Execution Controls**

Bit Position(s)	Name	Description
0	<del>LOADIWKEY exiting</del>	<del>This control determines whether executions of LOADIWKEY cause VM exits.</del>
1	Enable HLAT	This control enables hypervisor-managed linear-address translation. See Section 4.5.1.
2	EPT paging-write control	If this control is 1, EPT permissions can be specified to allow writes only for paging-related updates. See Section 29.3.3.2.
3	Guest-paging verification	If this control is 1, EPT permissions can be specified to prevent accesses using linear addresses whose translation has certain properties. See Section 29.3.3.2.

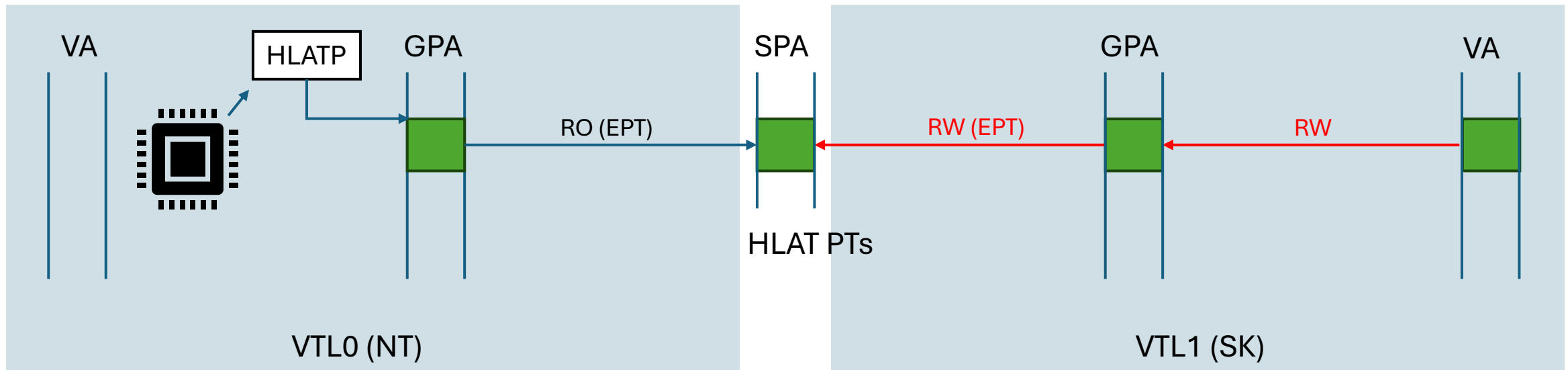
# How HLAT works (simplified example)

- Allows a HV to control VA -> GPA
- HV may create and manage the HLAT PTs, map it to GPA in RO, configure HLATP
- Processor uses HLATP, not CR3



# HLAT on Windows

- HLAT PTs is managed by VTL1
- Lets the HV offload more work to the less privileged component (VTL1)



# Demo – HLAT in action on 24H2

- Called “Hypervisor-Enforced Paging Translation” on Windows

The screenshot shows a WinDbg window with the command prompt interface. The command `!pte cilg_cioptions` is entered, and the output displays memory ranges and their contents. A red circle (1) highlights the output text, and a red circle (2) highlights the command. Below the WinDbg window, a Command Prompt window shows the execution of `recon2024_demo.exe` with various arguments. Red circles (3) and (4) highlight specific command arguments and their outputs.

```
0: kd> !pte cilg_cioptions
VA fffff80155b6b004
PXE at FFFFDB6DB6DB6F80 contains 0000000002B7063 pfn 2b7 ---DA--KWEV
PPE at FFFFDB6DB6DF0028 contains 0000000002B6063 pfn 2b6 ---DA--KWEV
PDE at FFFFDB6DBE005568 contains 000000000DE4063 pfn de4 ---DA--KWEV
PTE at FFFFDB7C00AADB58 contains 8900000011D0EC121 pfn 11d0ec -G--A--KR-V
```

```
C:\Users\tanda>cd C:\Release
C:\Release>recon2024_demo.exe
> this.exe --read64 va
> this.exe --write64 va value

C:\Release>recon2024_demo.exe --read64 fffff80155b6b004
0xfffff80155b6b004: 0x1c006

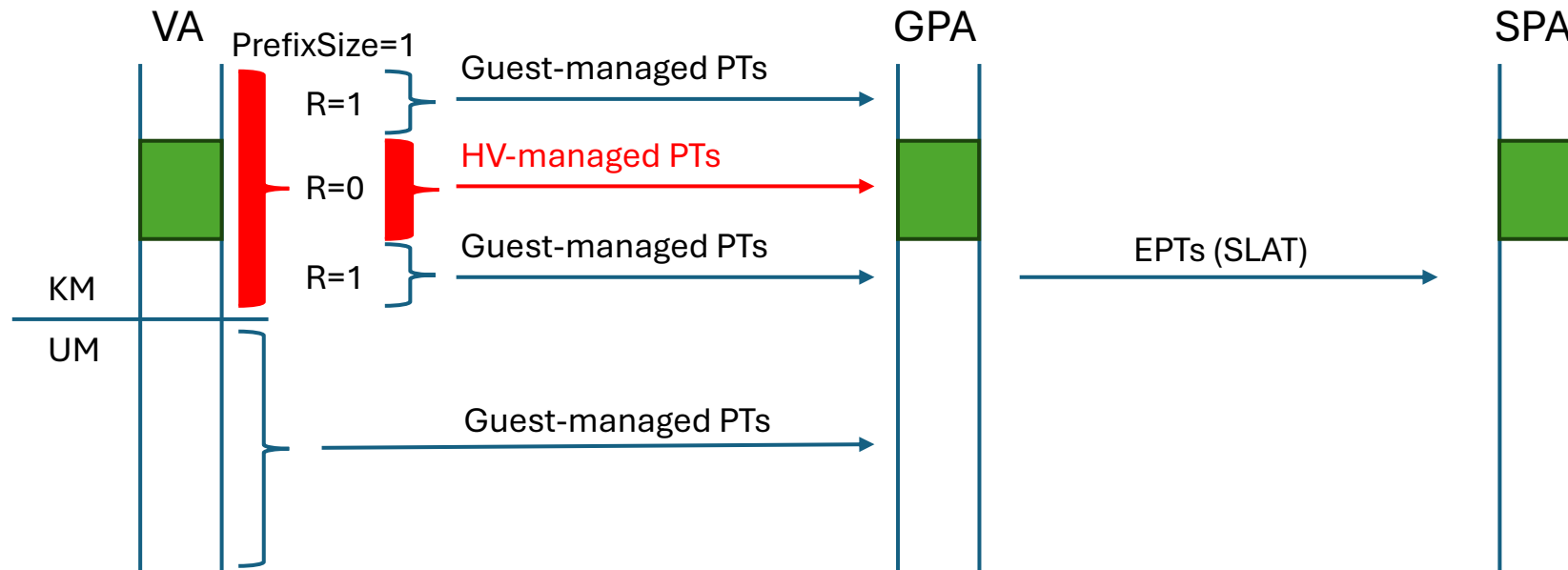
C:\Release>recon2024_demo.exe --write64 FFFFDB7C00AADB58 8900000000200121

C:\Release>recon2024_demo.exe --read64 fffff80155b6b004
0xfffff80155b6b004: 0x1c006
```

- <https://youtu.be/l8IMTGTbSKM>

# Configuring HLAT ranges

- HLAT prefix size VMCS to enable HLAT for KM address-only
- Restart bit to enable HLAT with page graduality
- Allows the OS to manage most of non-security sensitive VA -> GPA translations



# Areas protected by HLAT on Windows 24H2 Not exhaustive

- KDP:
  - Dynamic: Secure pool
  - Static: CiPolicy, CFGRO, SecKdpSe
- Exec. sections of all kernel modules
  - Prevents code remapping
- Others:
  - IDT, GDT, kCFG bitmap

## Spoiler:

- Everything that is represented by a normal table entry (NTE)



[github.com/tandasat/hvext](https://github.com/tandasat/hvext)

# Unprotected areas and thoughts

- UEFI runtime driver code is unprotected
  - Vulnerable to code remapping
- All but a few data sections are just not KDP-ed
  - Many functions pointers and flags
- What about systems without support of VT-rp?
- Regardless, great security improvement killing a class of attacks

# How HLAT is implemented in the NT kernel?



# Part 3 – The NT implementation

- HVPT – Hypervisor-enforced Paging Translation, is the name of HLAT implemented in Windows NT
- HVPT is enabled only if HVCI is on.
- To understand why, we should discuss how the Secure Kernel tracks the VTL 0 virtual memory.
- The architecture of VBS, the Secure Kernel, and VSM is already explained in the Windows Internals book (Part 2)
  - Secure images, NTEs and SK CI validation introduced also in a good article by Connor: <https://connormcgarr.github.io/secure-images/>
- ... so, we are **not** going to details of the NT and SK memory manager ...

# SkTool

```
SkTool
Hypervisor / Secure Kernel / Secure Mitigations Parser Tool 1.3

Querying Hypervisor Information... Success!

The Microsoft Hypervisor has been detected.
Hypervisor Status
Hyper-V Hypervisor running: 1
Version Information: 10.0.27650 - Service pack 1.0
Maximum number of supported VPs: 2048, LPs: 2048
Maximum number of remappable INT vectors: 0x1170
Hypervisor Debugging enabled: 0
Hypervisor scheduler type: Root
Second Level Address Translation (SLAT): 1
DMA remapping (support for device assignment): 0
Interrupt remapping: 1
IOMMU is present and protects HV / SK: 1

Querying Secure Kernel Information... Success!

Secure Kernel is currently running.
VBS is enabled due to: VBS registry configuration.
Current active VBS policies: VBS enabled, VSM required, HVCI, Boot chain signer soft enforced.

Secure Kernel Status
Secure Kernel Running: 1
HVCI Enabled (StrongCodeGuarantees): 1
UEFI Firmware page protection: 1
VSM Master key available: 1 (persistent: 1)
No Secrets Mode: 0
HVCI Strict mode: 1
Debug of W^X pages enabled: 0
LsaIso trustlet running: 1 (Credential Guard: 1 - Key Guard: 1)
DMA Protection available: 1 - in use: 1
Hardware MBEC in use: 1
APIC Virtualization available: 1
Kernel-mode CET (Shadow Stacks) enabled: 1 (hardware capable 1)
Kernel-mode CET audit mode: 0 (HV supports CET SSS: 1)
Hypervisor-Enforced Paging Translation (HVPT) hardware-capable: 1
Hypervisor-Enforced Paging Translation (HVPT) enabled: 1
```

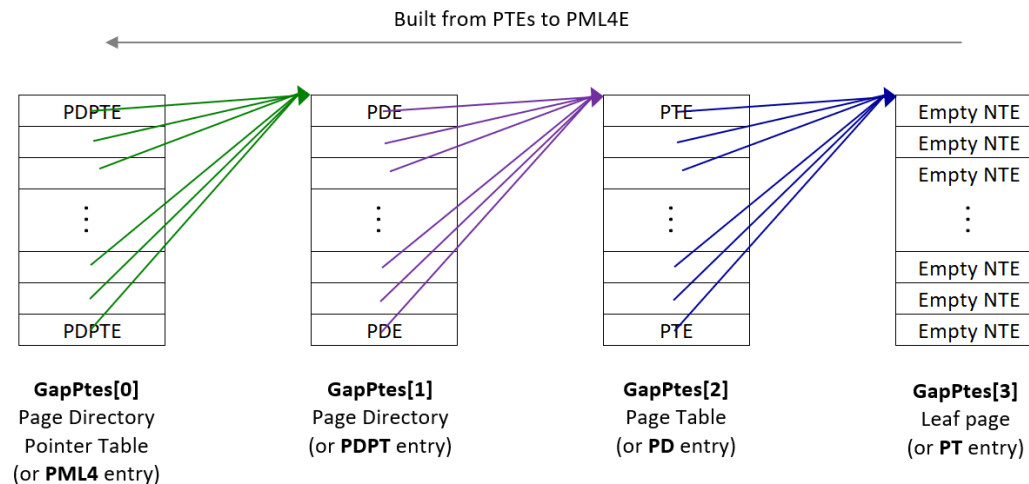
- Ready to dump the HVPT information for you
- Able to identify why VBS **is** or **is not** enabled
- A lot of other cool features!
- Free and available in the Windows SDK

# Private and shared pages

- The NT Memory Manager (Mm), supports two kind of physical memory pages backing Kernel virtual address space: **Private** and **shared** pages.
- A Private page is a page that is **not** in any Control Area (CA), not sharable, usually belongs to Boot drivers or pages in which **private fixups** are applied (two distinct concepts to not be confused\*)
- When HVCI is on, all operations on kernel memory needs to be “blessed” by SK
- This implies that SK needs a way to **track** both shared and private pages.
- Two entities exists for tracking kernel VA space: Secure Images and Normal Table Entries (**NTEs**)
- A Secure image is a data structure, used also to track **shared pages** via its prototype PTEs.
- A NTE is allocated for each chunk of valid Kernel VA, and, before HVPT, was used to track only **private pages**.

# Normal Table Entry (NTE)

- On startup, SK **reserves** NTEs for the entire NT kernel address space, with a technique called Sparse mapping (only the needed leaf pages are really allocated, all the rest use gap frames)
- A NTE was a **software** construct. HVPT requires the HLAT table to be physically mapped in **both** SK and NT.
- In Windows, the NTE table is built by VTL 1 on boot and is usually empty, entirely mapped with gap frames.



- During the boot phases, the SK **expands** and **fills** the NTE table with leaf entries when processing all the boot-loaded drivers (runtime drivers do the same, later in the boot process).
- Different kinds of NTEs, the most important being **Privileged** vs **Unprivileged** NTEs

# How the black magic work – the born of HVPT

- In the Intel specs, HLAT requires 2 new bits in the leaf **EPT** entries and a brand-new **HLAT table**
- HLAT table entries are almost identical to classical page tables entries, except for the “**Restart**” bit (#11)
- When the restart bit is **1**, all the other bits are ignored by the processor.
- This means that we can **re-use** and **expose** the NTE table as HLAT table 😊
- Required some redesign on the NTE states, and especially now NTE tracks also **shared** pages\*
- In particular, SK now needs to intercept also all the TLB flushes from NT.
  - SK and NT must be completely **in synch**.
- SK **builds** the NTE table, and then asks to the HV to set the HLATP pointer in the VMCS (for NT)
- ... so, when HVCI is enabled, HLAT table **is** the NTE table... All the Kernel modules / VA space tracked by NTEs is protected!



# Session space, problems and solutions...

- The HVPT implementation discussed in this talk is **extremely** simplified.
- Multiple features still missed back in the days:
  - The HV did **not** support cross-VTL TLB flush, needed for HLAT
  - Session space was a big problem, since uses a subset of the Kernel address space per session (1 top-level PTE entry = 512GB on X64)
  - Win32k was **heavily** depending on Session space
  - HLAT is managed by the Secure Kernel, and SK does not know or implement any session
- So two choices were available:
  1. Remove entirely the “session” concept from NT
  2. Add the “session” support in SK

# Session space

- The Session space was originally created when the entire NT kernel was fitting in 1 or 2 GB of 32-bit address space to isolate and fit multiple sessions of big servers (Terminal server is a good example)
- Now the only supported architecture is 64-bit only, meaning that the kernel address space is 256TB big. **No need for sessions anymore!**
- Win32k uses an internal array to store data structure needed for different sessions
- All in kernel space!



**Yarden Shafir**  
@yarden\_shafir

Is Microsoft preparing to get rid of session space in 24H2? Or is something else going on here?



**Andrea Allievi** @aall86 · 18 ago 2023

Yes, Session space is gone. There is a big motivation for it, but is too early :-)

2

2

17

3.998



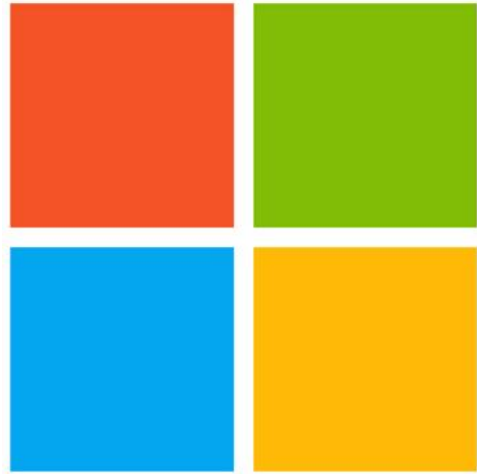
```
Processor Commit:          2157 (      8628 Kb)
Unable to get offset of nt!_MM_SESSION_SPACE.WsListEntry
Session Commit:           0 (         0 Kb)
Shared Commit:          418155 (    1672620 Kb)
Special Pool:             0 (         0 Kb)
Kernel Stacks:           30788 (    123152 Kb)
Pages For MDLs:           68009 (    272036 Kb)
ContigMem Pages:         48128 (    192512 Kb)
Partition Pages:          0 (         0 Kb)
Pages For AWE:            0 (         0 Kb)
NonPagedPool Commit:     193738 (    774952 Kb)
PagedPool Commit:        242532 (    970128 Kb)
Driver Commit:           44390 (    177560 Kb)
Boot Commit:             3524 (     14096 Kb)
PFN Array Commit:        98815 (    395260 Kb)
SmallNonPagedPtesCommit: 1764 (     7056 Kb)
```

# Resources

- FUTURE INTEL® ARCHITECTURE INSTRUCTION EXTENSIONS AND FEATURES
  - [https://community.intel.com/legacyfs/online/drupal\\_files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf](https://community.intel.com/legacyfs/online/drupal_files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf)
  - Captures background better than SDM. Some features names are old, but great resource.
- SkTool
  - [Windows SDK](#)
  - (unofficially) <https://github.com/AaLl86/WindowsInternals/tree/master/Training>
- C code to check HVPT availability
  - <https://gist.github.com/tandasat/890d4aad0c54f784f749ba5c894954d6>
- Demo code
  - [https://github.com/tandasat/recon2024\\_demo](https://github.com/tandasat/recon2024_demo)
- HLAT articles (details also on PW and GPV)
  - Initial HLAT analysis - <https://www.andrea-allievi.com/blog/alder-lake-and-the-new-intel-features/>
  - Satoshi HLAT series: <https://tandasat.github.io/blog/2023/07/05/intel-vt-rp-part-1.html>



# We are hiring talented people



Windows Kernel  
Team



- Are you brave enough and want to deal with design like HVPT?
- Do you like low-level engineering, security and the OS architecture?
- So, the job can be for you 😊
- <https://jobs.careers.microsoft.com/global/en/job/1728506/Software-Engineer-2> is a good example
- Drop me a mail ([andreaa@microsoft.com](mailto:andreaa@microsoft.com)) or a message in Twitter ([@aall86](https://twitter.com/aall86)) in case interested



# Thanks for Attending Recon!

---

**Andrea Allievi** (@aall86)

**Satoshi Tanda** (@standa\_t)

Another Windows Internals - Fundamentals course will start on July 11<sup>th</sup> for whoever is interested:

<https://www.oreilly.com/live-events/windows-internals-fundamentals/0636920095044/>

Hypervisor Development for Security Researchers:

<https://tandasat.github.io/>