29th June 2024 – **REcon 2024**

# Hypervisor-enforced Paging Translation

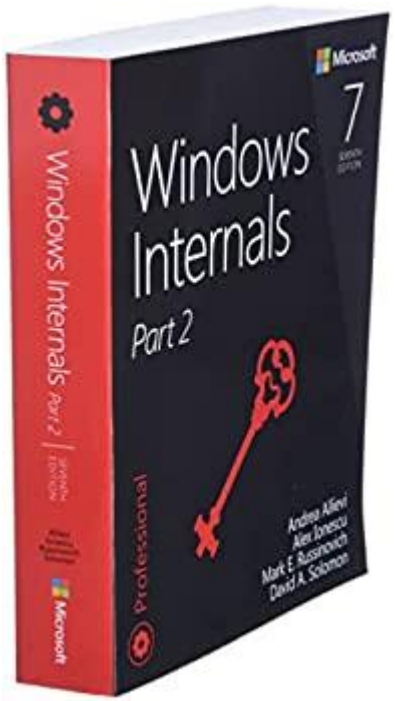
## The end of non data-driven Kernel Exploits?

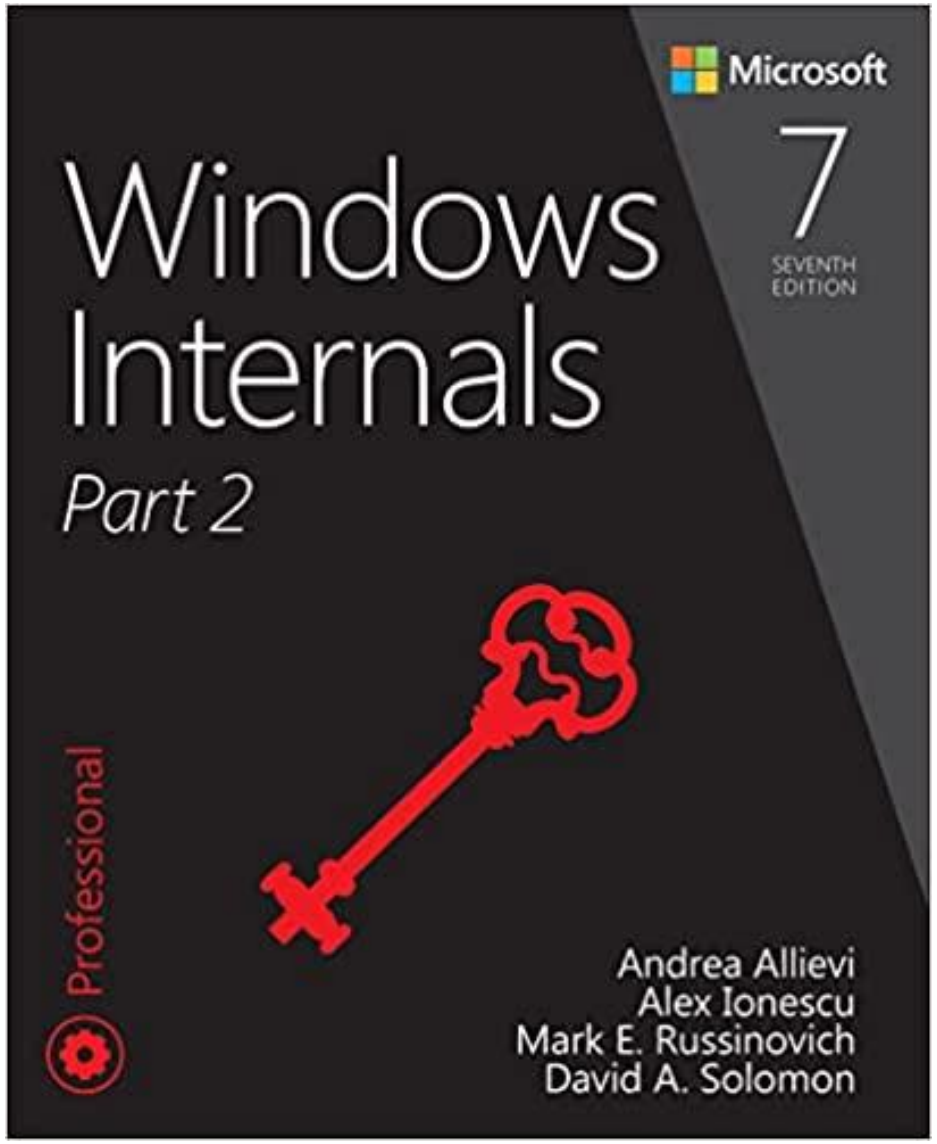
Andrea Allievi (@aall86)
Satoshi Tanda (@standa_t)

# 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
  - @standa_t
  - @tandasat
  - @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

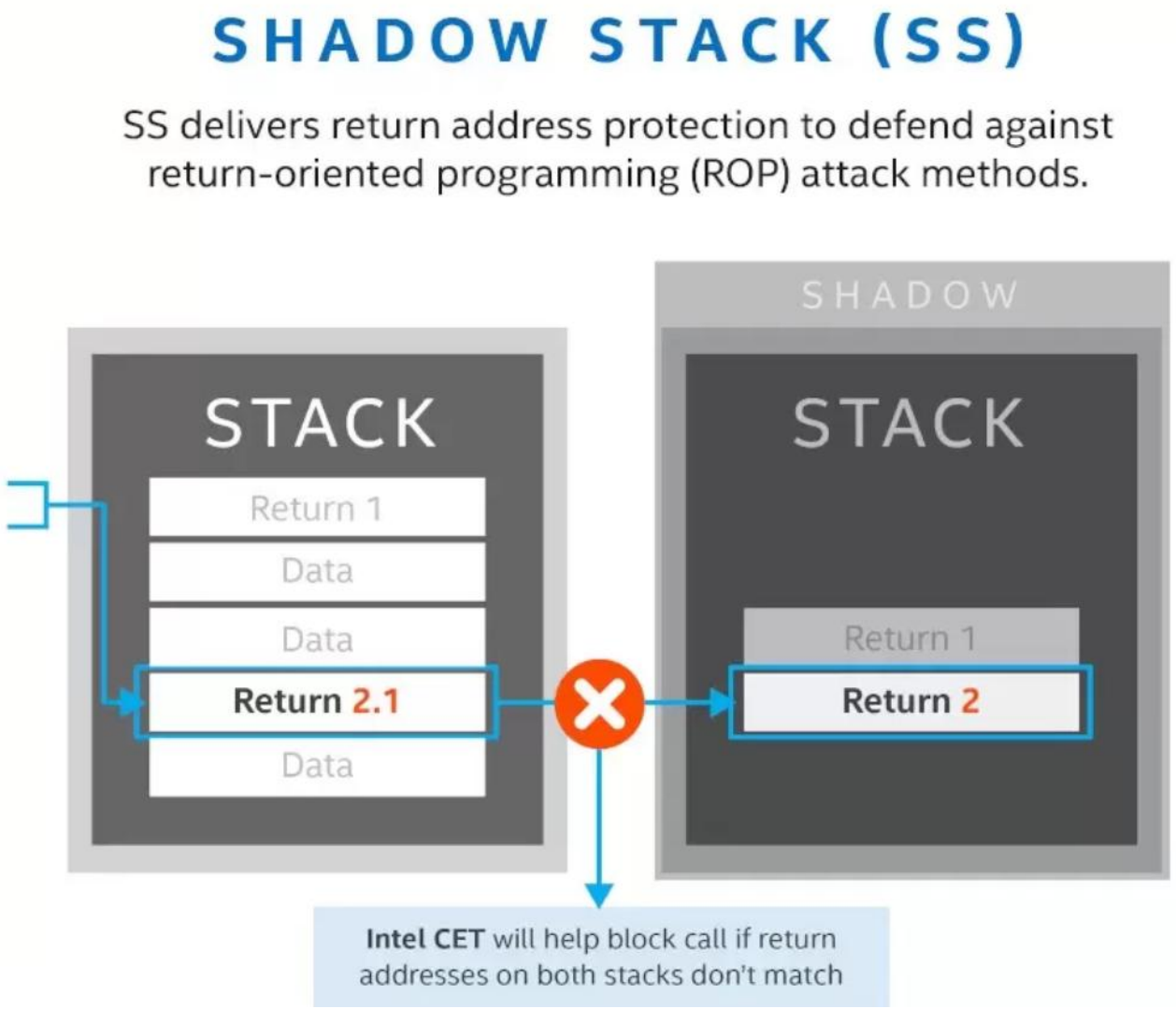
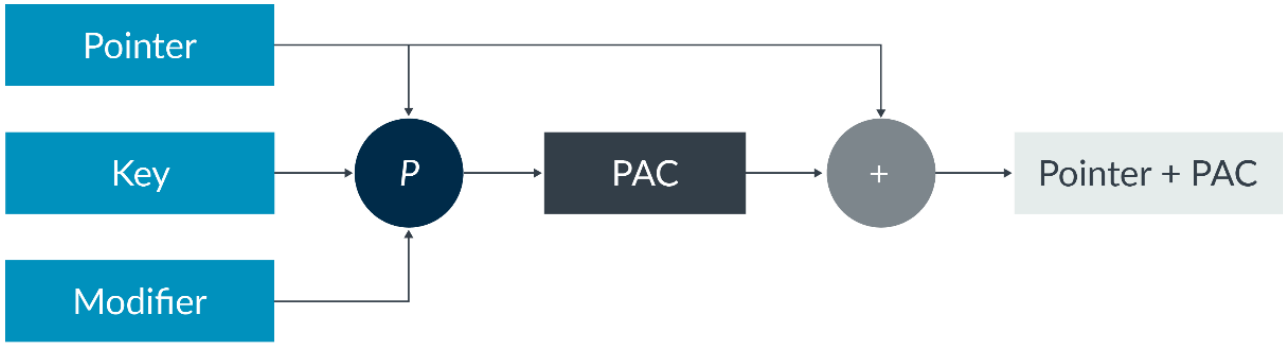



*Image courtesy of Phoronix
(I am too lazy to draw it myself)

# 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

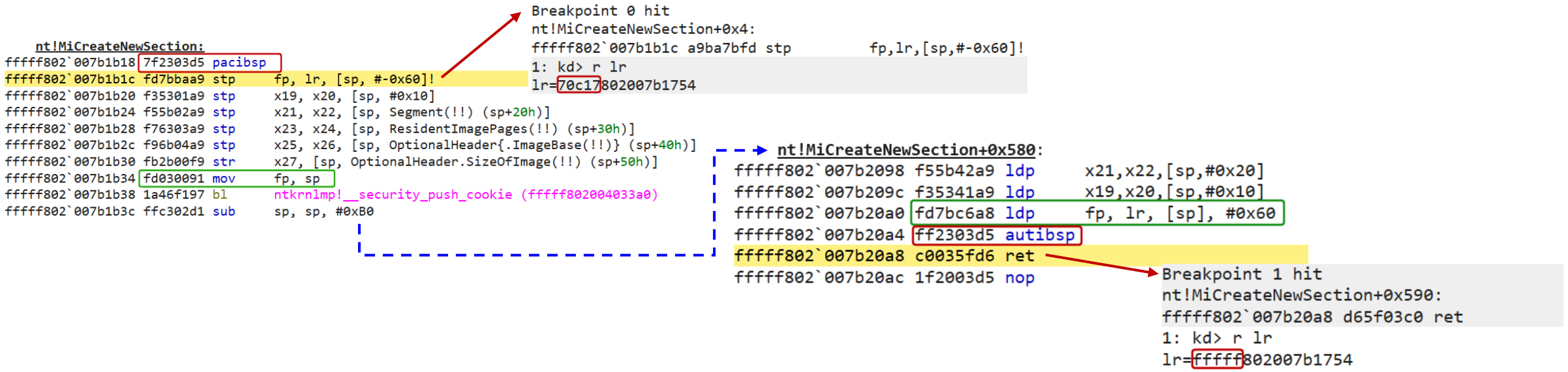
- On function **entry**, the compiler always emit the PACIBSP opcode, which stands for

  `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**.



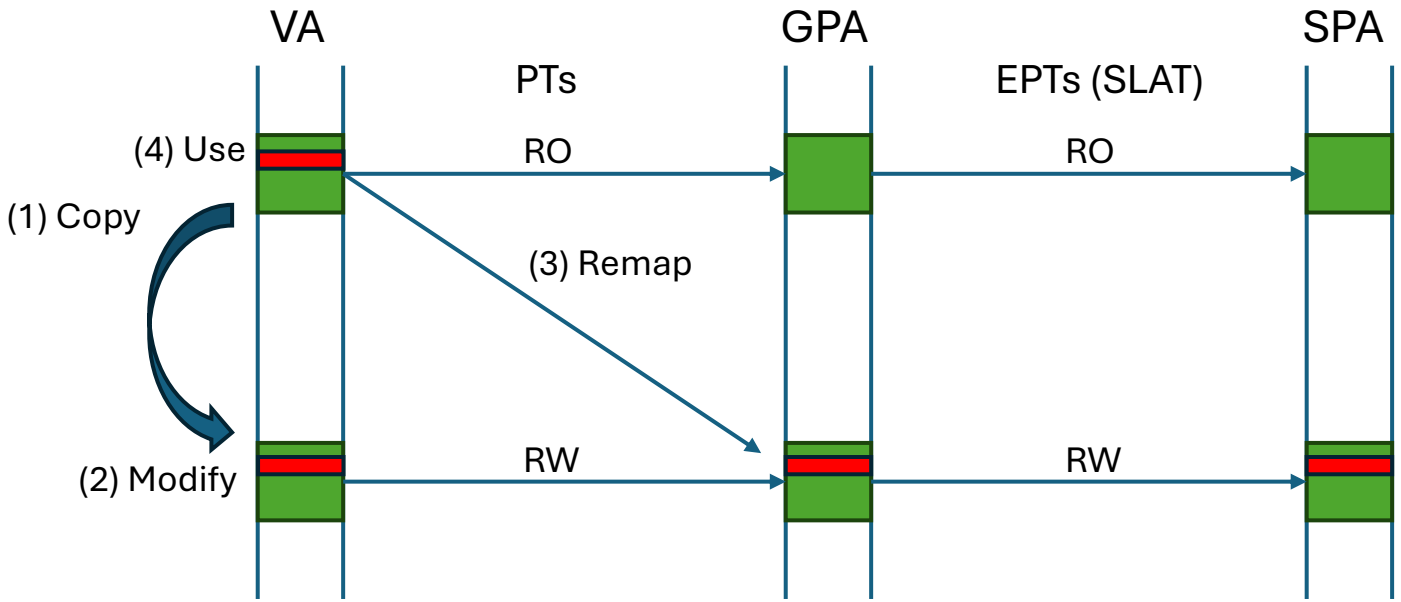
# Kernel Data Protection (KDP)

- 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.

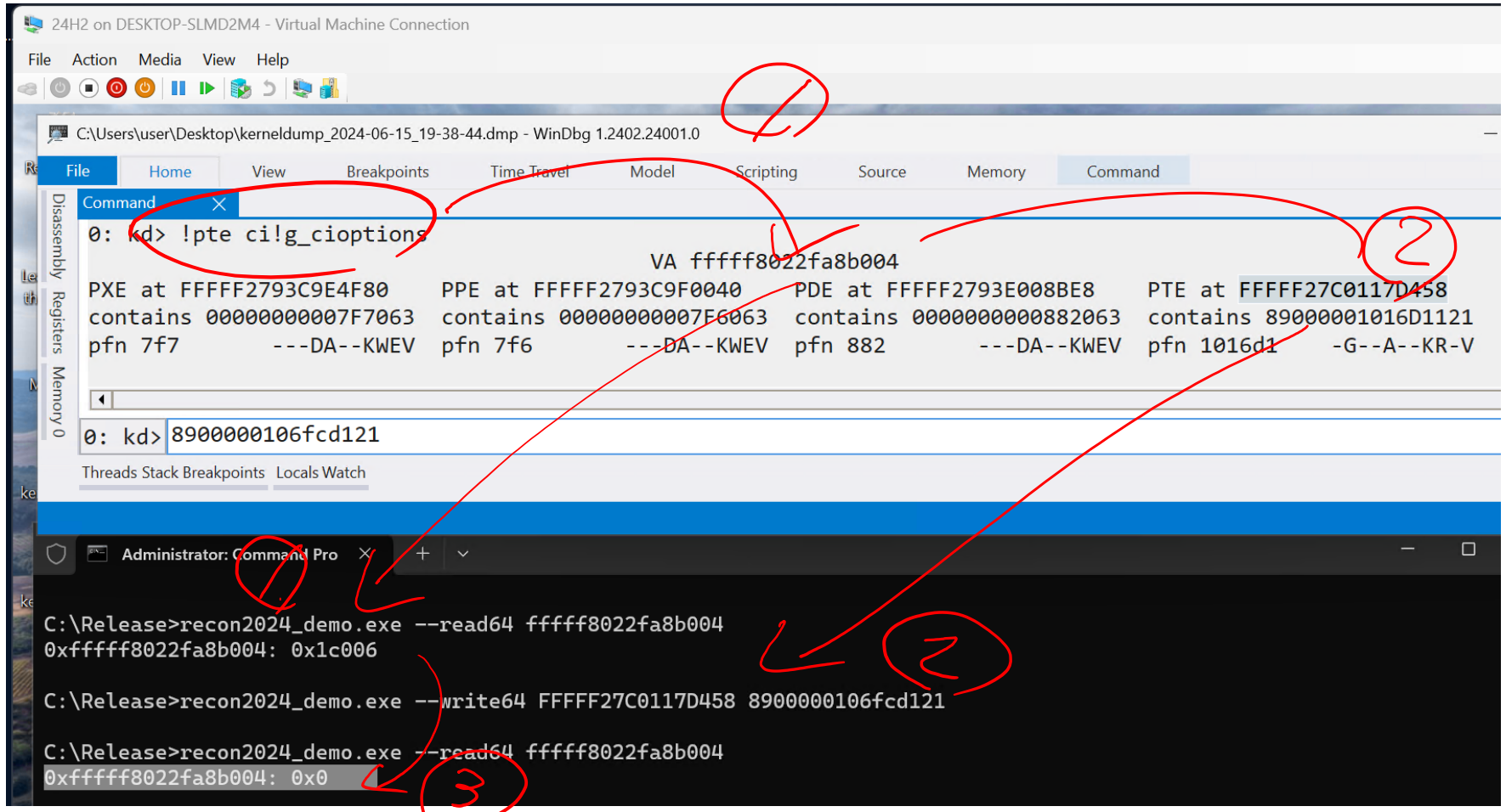
# … Now let’s move on and introduce HLAT …

# 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



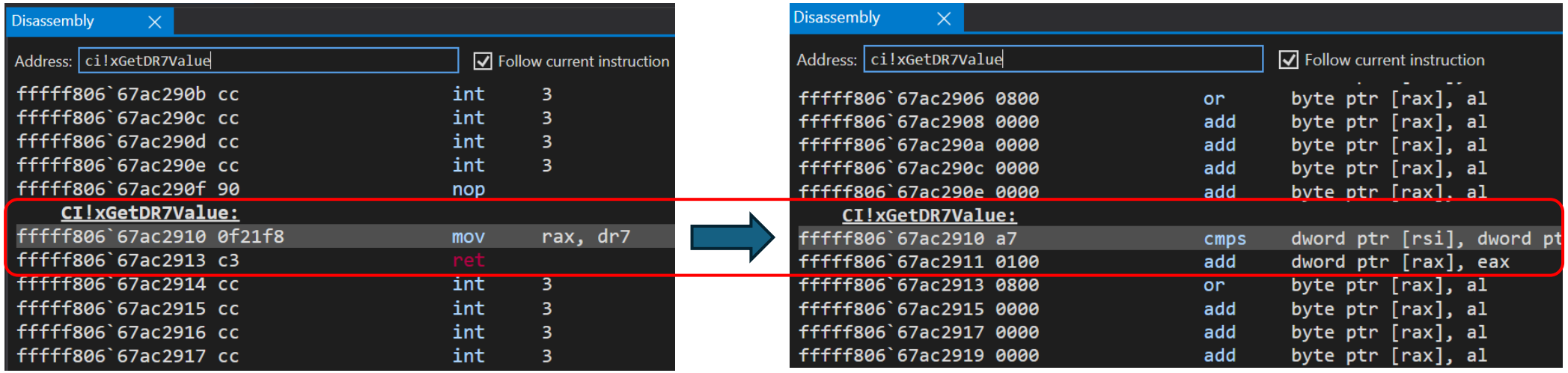
# Demo – Zeroing ci!g_CiOptions under KDP




- 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



# Introduction to VT-rp (redirect protection)

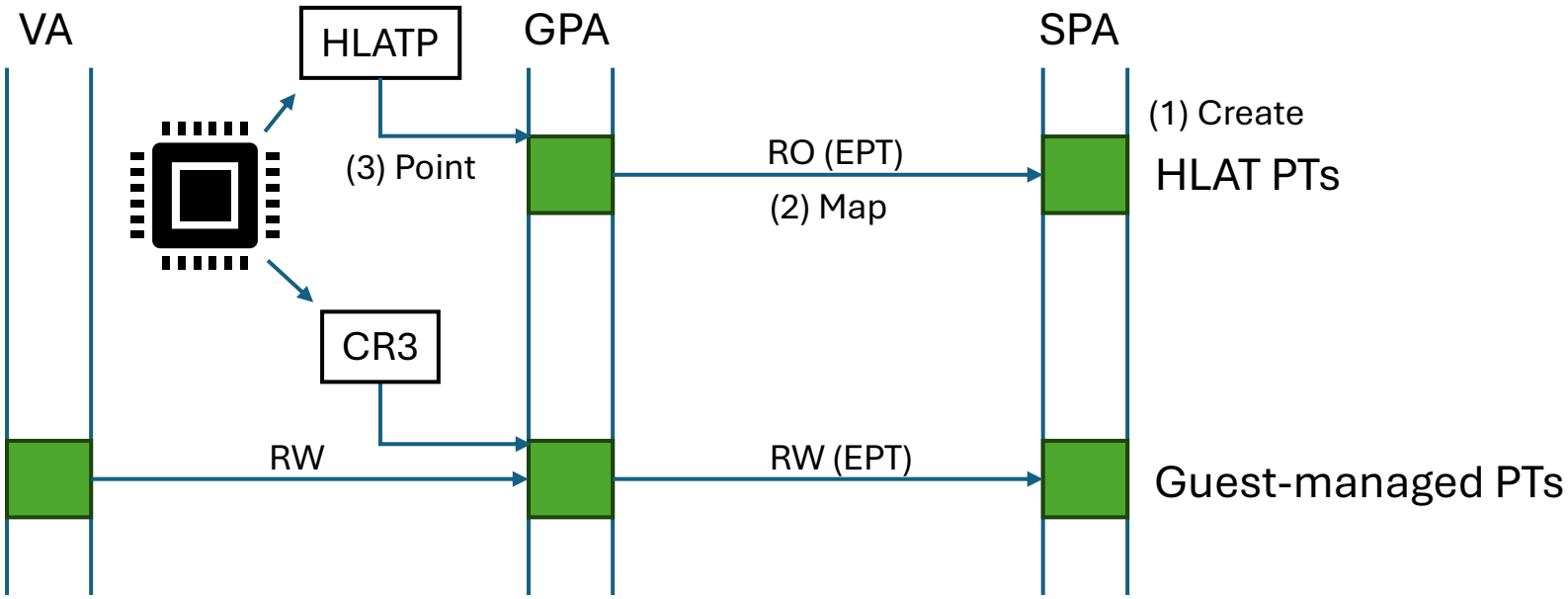
- 3 features: HLAT, PW, GPV
- Available in the subset of Intel 12+ gen
- HLAT is the focus of this talk

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

| Bit Position(s) | Name | Description |
|---|---|---|
| 0 | LOADIWKEY exiting | This control determines whether executions of LOADIWKEY cause VM exits. |
| 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. |

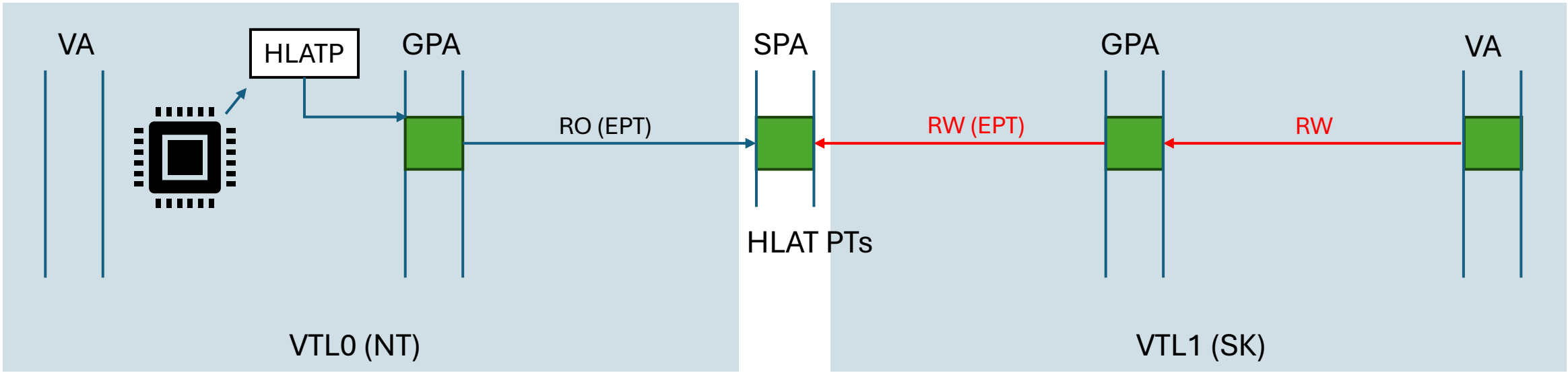
# How HLAT works (simplified example)

- 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



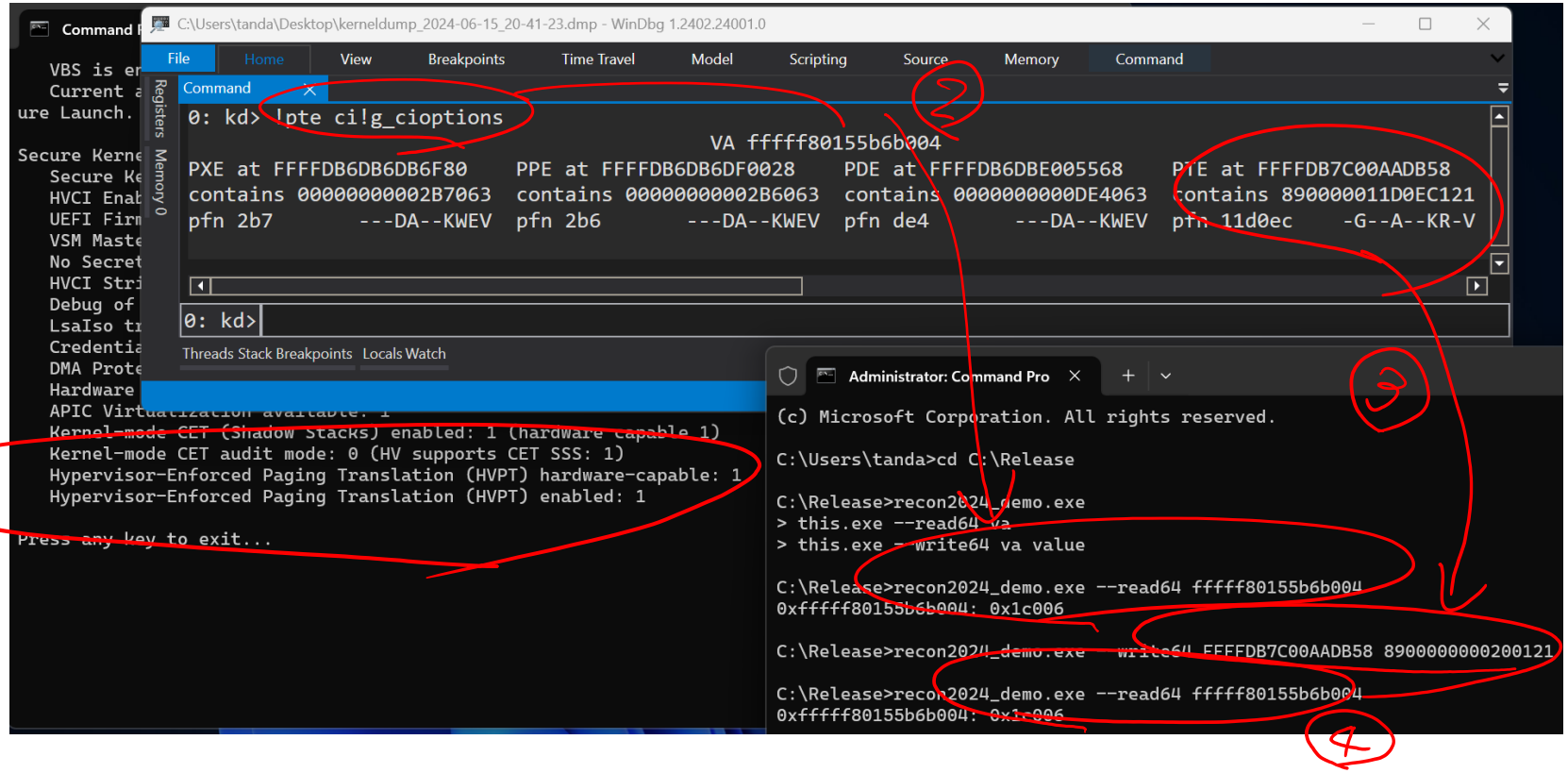
# HLAT on Windows

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



# Demo – HLAT in action on 24H2

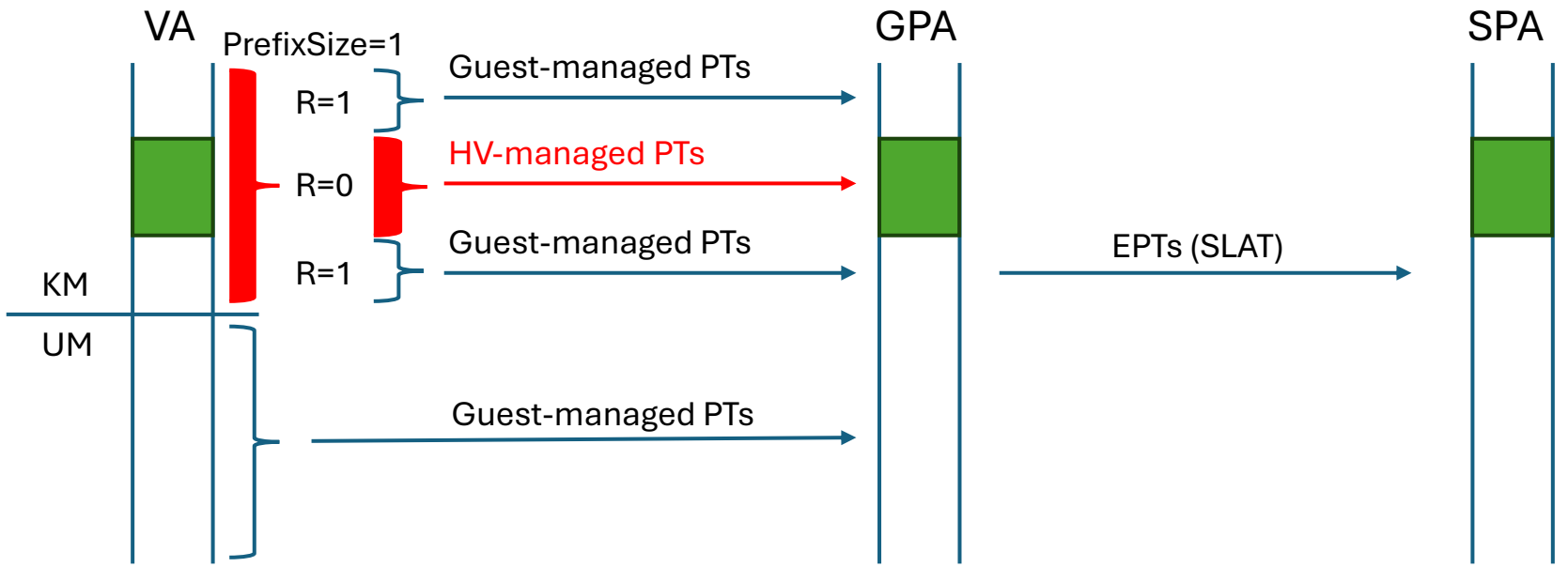
- Called “Hypervisor-Enforced Paging Translation” on Windows




- 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

# Unprotected areas and thoughts

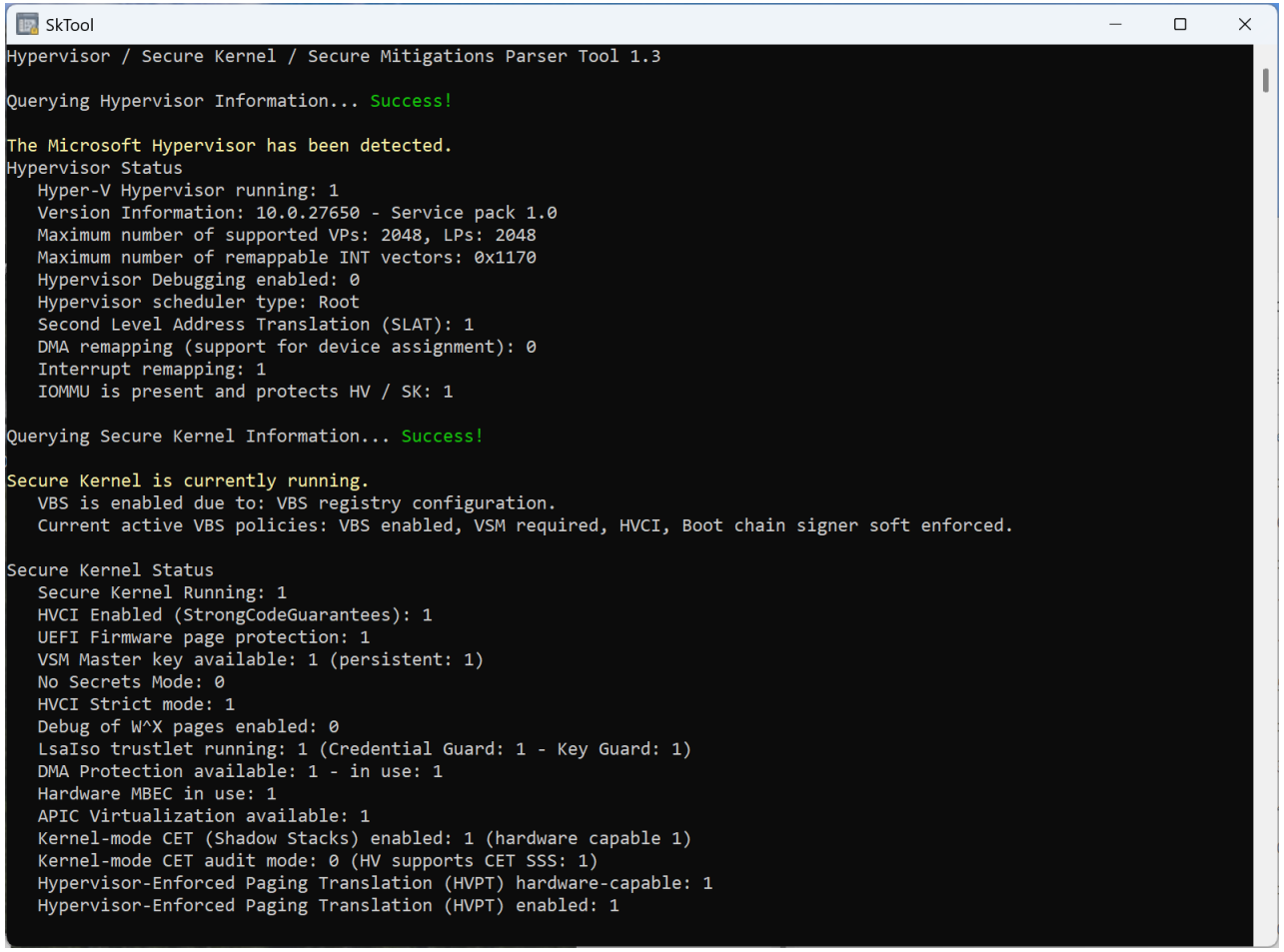
- UEFI runtime driver code is unprotected
  - Vulnerable to code remapping
- All but a few data sections are just not KDP-ed
  - Many functions pointers and flags
- What about systems without support of VT-rp?
- Regardless, great security improvement killing a class of attacks

# 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



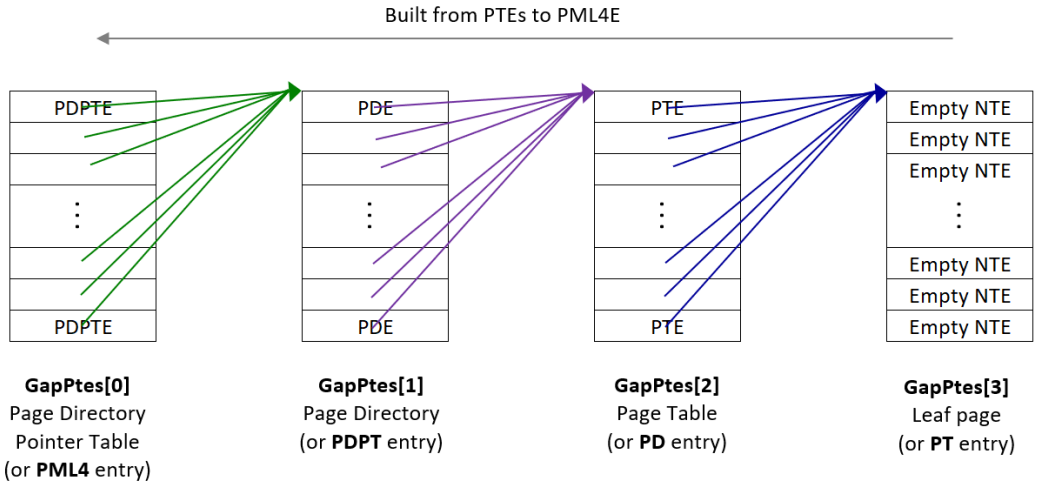

- Ready to dump the HVPT information for you
- Able to identify why VBS **is** or is **not** enabled
- A lot of other cool features!
- Free and available in the Windows SDK

# 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 (**NTE**s)
- 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!


```
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
  - 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
- 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


- 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) or a message in Twitter (@aall86) in case interested


# Thanks for Attending Recon!

**Andrea Allievi** (@aall86)

**Satoshi Tanda** (@standa_t)

Another Windows Internals - Fundamentals course will start on July 11th for whoever is interested: https://www.oreilly.com/live-events/windows-internals-fundamentals/0636920095044/

Hypervisor Development for Security Researchers: https://tandasat.github.io/