

Virtualization-assisted Operating System Security

Workshop on Security of Software/Hardware Interfaces
SILM

Sergej Proskurin

3rd of July 2023

whoami

- ▶ Graduated at the Technical University of Munich
- ▶ In the past, I have contributed to:
 - ▶ Xen Project hypervisor
 - ▶ Honeynet Project
 - ▶ Binary Analysis System DRAKVUF
- ▶ Today, Senior Security Engineer @ BedRock Systems
- ▶ Generally interested in:
 - ▶ Virtualization Technology
 - ▶ Virtualization-assisted Operating System Security
 - ▶ Virtual Machine Introspection
 - ▶ Operating System design & security



Disclaimer

I do not speak for my employer.

All opinions and information conveyed today are my personal views.

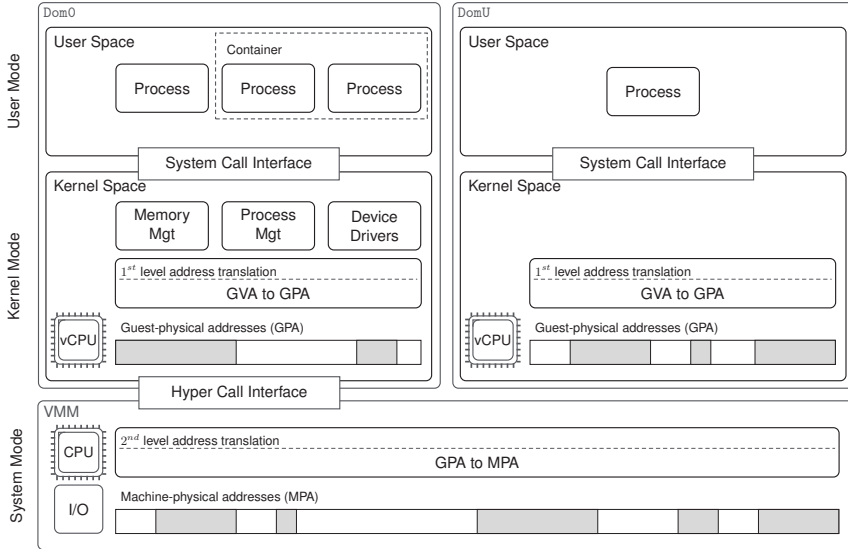
Virtualization-assisted primitives for
dynamic binary analysis and OS security



Virtualization-assisted primitives for dynamic binary analysis and OS security

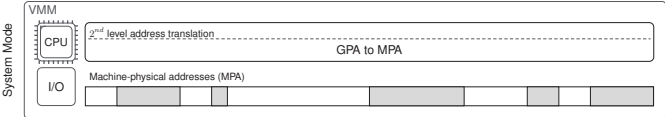
Key hypothesis: we can repurpose *HW-assisted virtualization extensions*
to introduce new primitives for:

- i facilitating **stealthy analysis**, despite potentially missing hardware capabilities
- ii strengthening the **isolation** capabilities of modern OSes to enhance security



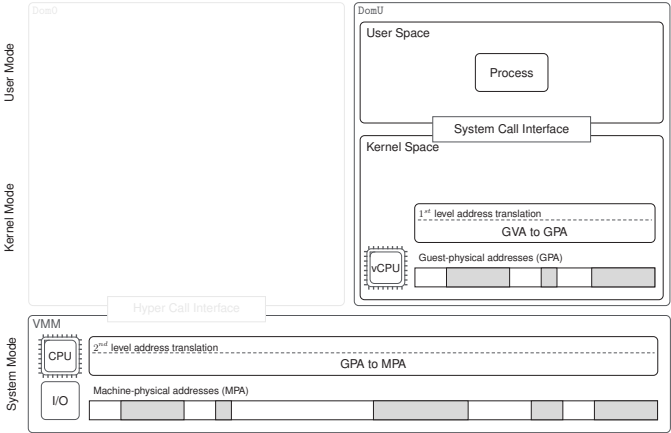
Organizational Structure

- i On-demand deployment of virtualization-assisted frameworks



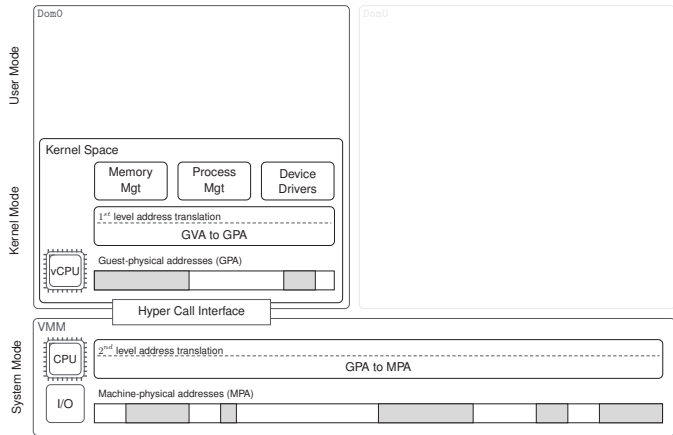
Organizational Structure

- i On-demand deployment of virtualization-assisted frameworks
- ii Primitives for stealthy malware analysis on Arm



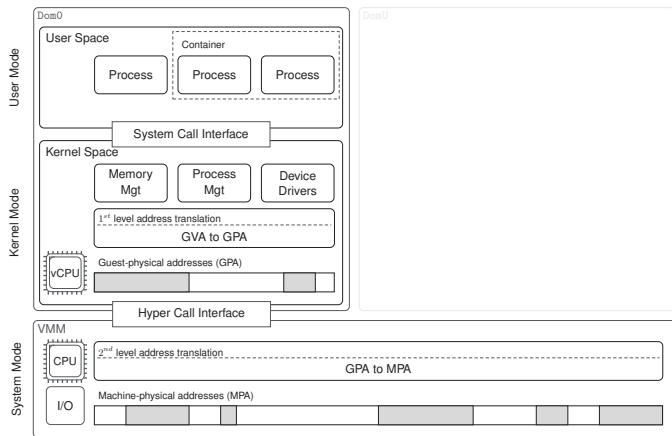
Organizational Structure

- i On-demand deployment of virtualization-assisted frameworks
- ii Primitives for stealthy malware analysis on Arm
- iii Virtualization-assisted memory protection primitives
 - ▶ In kernel space



Organizational Structure

- i On-demand deployment of virtualization-assisted frameworks
- ii Primitives for stealthy malware analysis on Arm
- iii Virtualization-assisted memory protection primitives
 - ▶ In kernel space
 - ▶ In user space



Part I

On-demand Deployment of Virtualization-assisted Frameworks



On-demand Virtualization

Motivation

To support system virtualization a VMM must be set up in advance

- ▶ Limited popularity of virtualization-assisted security frameworks in non-cloud environments

Observation: No need for a fully-fledged COTS VMM

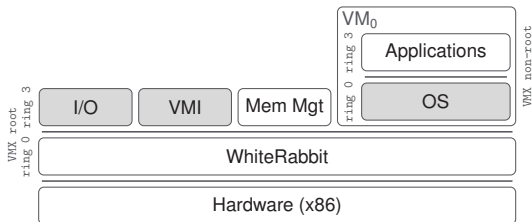
- ▶ OSes can ship their own minimalistic VMMs in form of a kernel subsystem
- ▶ Deploy VMMs on-demand

Various solutions available:

- ▶ Bareflank hypervisor, SimpleVisor, BluePill rootkit, etc.
- Our solution: WhiteRabbit virtualization-assisted security framework

WhiteRabbit VMM

On-the-Fly Virtualization

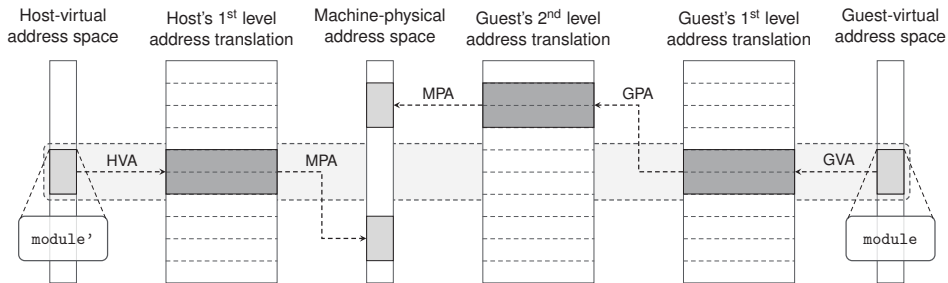


WhiteRabbit moves a running Linux OS into a virtual environment on Intel and Arm

- ▶ Loaded as kernel module
 - ▶ Inspired by the Blue Pill rootkit
- ▶ Microkernel architecture designed for on-the-fly virtualization
 - ▶ Only essential functionality in kernel space
 - ▶ Private subsystems placed in user space

WhiteRabbit VMM

Module Relocation & Isolation



WhiteRabbit has to be aware of split-personality malware

- ▶ WhiteRabbit removes in-guest artifacts and hides in memory
 - ▶ Relocates its own code and data segments
 - ▶ Maps `module'` and `module` to the same virtual address space (guest and host)
 - ▶ Initiates a [clean module destruction](#) inside of the guest OS

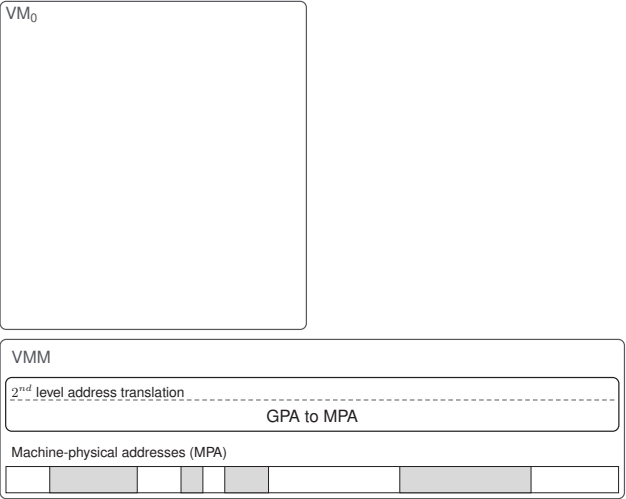
Part II

Virtual Machine Introspection



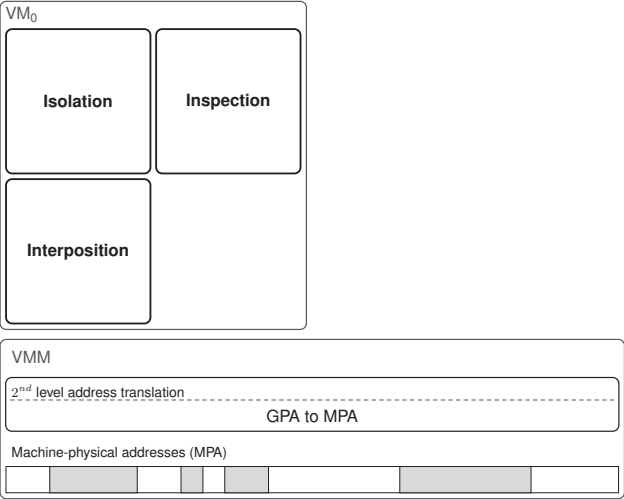
Virtual Machine Introspection

Recap



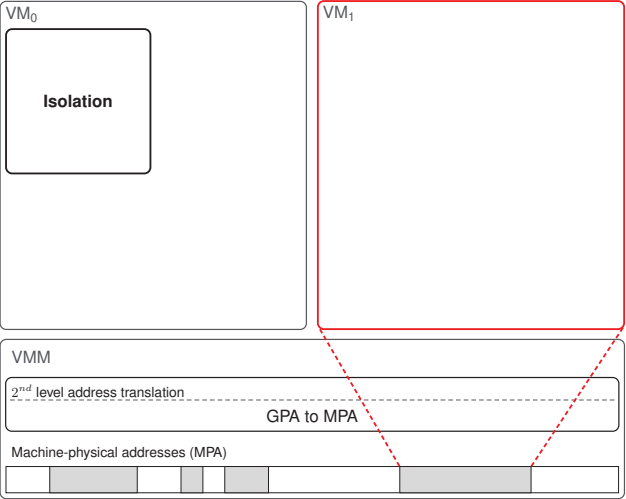
Virtual Machine Introspection

Recap



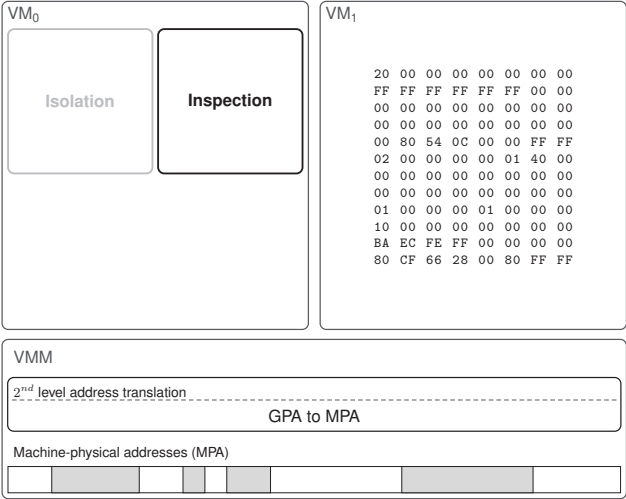
Virtual Machine Introspection

Recap



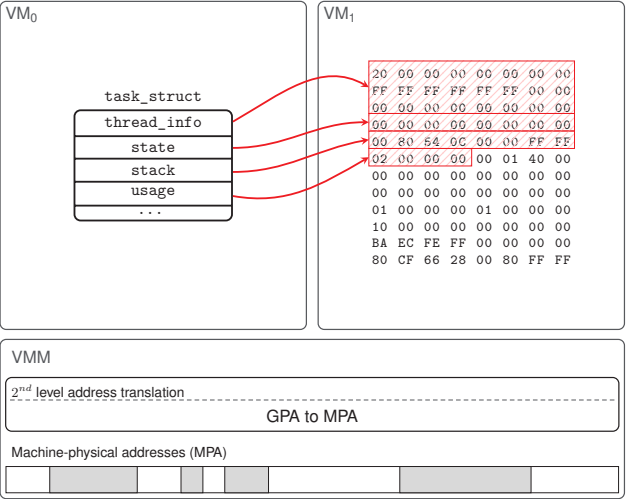
Virtual Machine Introspection

Recap



Virtual Machine Introspection

Recap



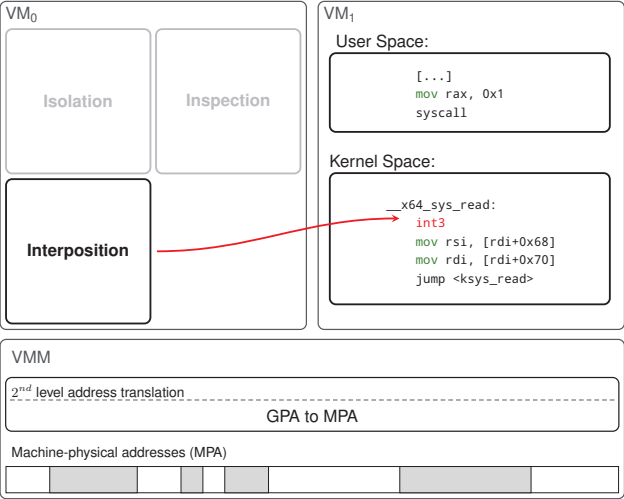
Virtual Machine Introspection

Recap



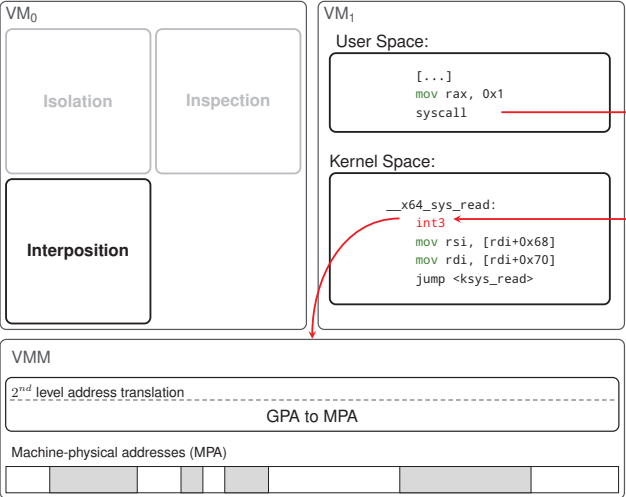
Virtual Machine Introspection

Recap



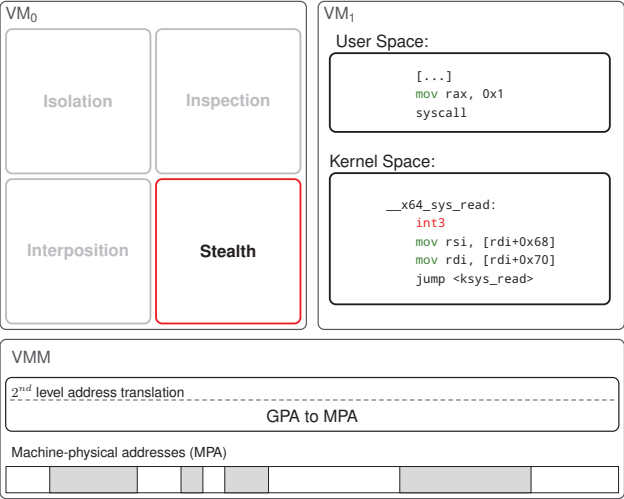
Virtual Machine Introspection

Recap



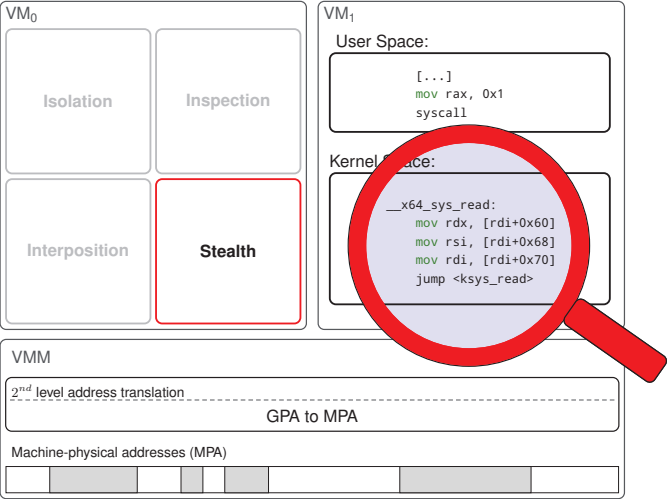
Virtual Machine Introspection

Recap



Virtual Machine Introspection

Recap



The Need for Stealthy Monitoring

Motivation

Split-personality malware

- ▶ Employ anti-virtualization to reveal a VMM (red pills)

Perfect VM transparency is not feasible

- ▶ Insufficient to reveal virtual environment alone!

More interesting to know whether the system is being analyzed

- ▶ Hide analysis artifacts from the guest

Requirements for Stealthy Monitoring

- ① Intercept the guest in kernel space
- ② A stealthy single-stepping mechanism
- ③ Execute-only memory

Req. 1: Implementing Kernel Tap Points

Instruction of Choice: [Secure Monitor Call](#) (SMC)

- ▶ Guest is not able to subscribe to SMC traps
- ▶ SMCs do not have to be re-injected into the guest
- ▶ Can only be executed in the guest's kernel

User Space:

```
[...]  
mov x8, #0x3f  
svc #0x0
```

Kernel Space:

```
SyS_read:  
stp x29, x30, [sp, #-64]  
mov x29, sp  
stp x21, x22, [sp, #32]  
[...]
```

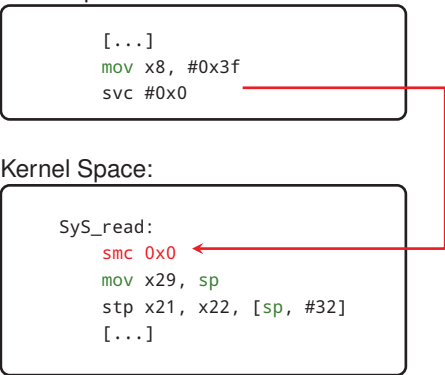
Req. 1: Implementing Kernel Tap Points

Instruction of Choice: [Secure Monitor Call](#) (SMC)

- ▶ Guest is not able to subscribe to SMC traps
- ▶ SMCs do not have to be re-injected into the guest
- ▶ Can only be executed in the guest's kernel

User Space:

```
[...]  
mov x8, #0x3f  
svc #0x0
```



Kernel Space:

```
SyS_read:  
smc 0x0  
mov x29, sp  
stp x21, x22, [sp, #32]  
[...]
```

Req. 1: Implementing Kernel Tap Points

Instruction of Choice: **Secure Monitor Call (SMC)**

- ▶ Guest is not able to subscribe to **SMC** traps
- ▶ **SMCs** do not have to be re-injected into the guest
- ▶ Can only be executed in the guest's kernel

Issues: How to remain stealthy and in control?

- ⚡ Removing tap points introduces race conditions
- ⚡ No hardware support for stealthy single-stepping

User Space:

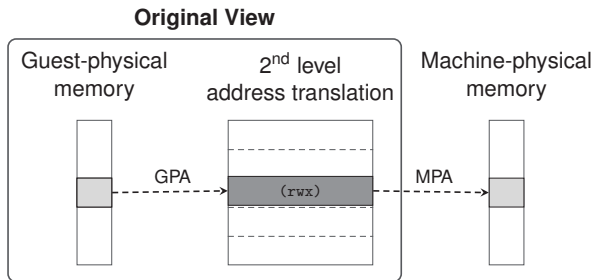
```
[...]  
mov x8, #0x3f  
svc #0x0
```

Kernel Space:

```
SyS_read:  
smc 0x0  
mov x29, sp  
stp x21, x22, [sp, #32]  
[...]
```

Req. 2: Stealthy Single-Stepping

The Xen `altp2m` Subsystem on Arm

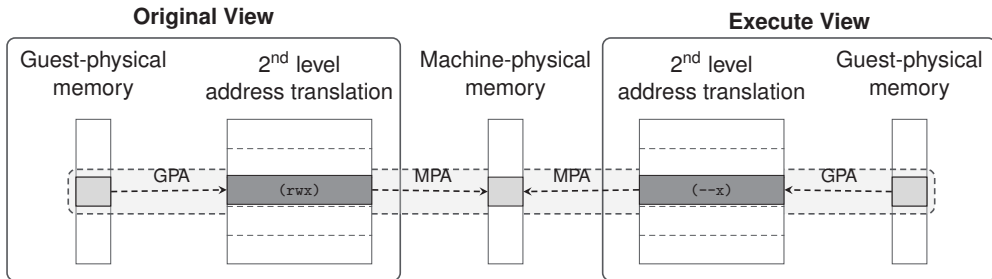


Typically, a VMM uses one set of second level address translation tables (SLAT)

- ▶ Defines the guest's **global view on the physical memory**
- Changes in the global view are perceived by all vCPUs

Req. 2: Stealthy Single-Stepping

The Xen `altp2m` Subsystem on Arm



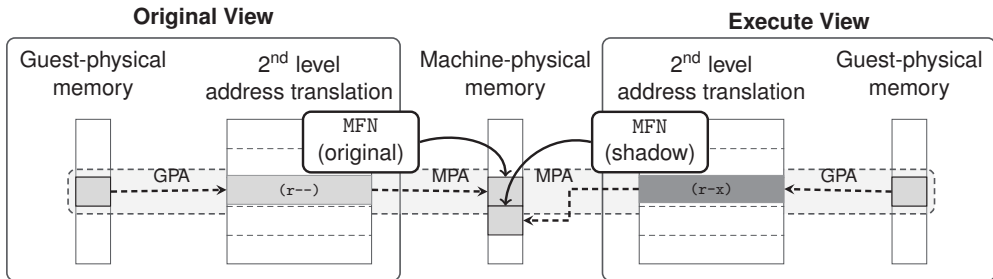
Implement Xen alternate p2m (`altp2m`) subsystem for Arm

- ▶ Maintains different views on the guest's physical memory
- ▶ Allocates and assigns different memory views to vCPUs

→ **Switch views** instead of relaxing permissions in a global view!

Req. 2: Stealthy Single-Stepping

The Xen `alt2m` Subsystem on Arm

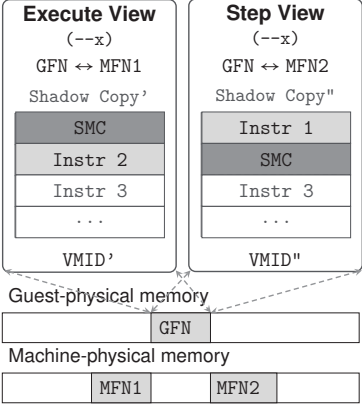


Implement Xen alternate p2m (`alt2m`) subsystem for Arm

- ▶ Allows to remap same guest-physical to different machine-physical page frames
- Facilitates race-free SMC injections in selected views

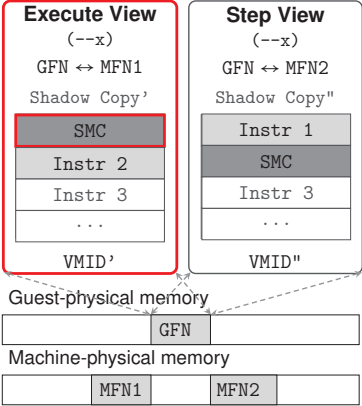
Req. 2: Stealthy Single-Stepping

Race-free Single-Stepping Scheme



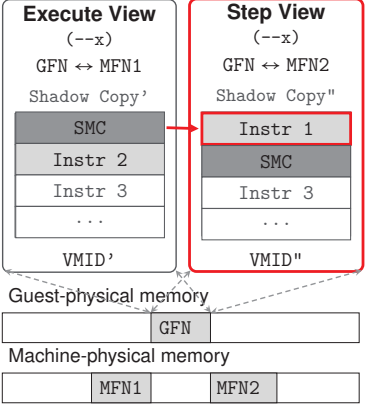
Req. 2: Stealthy Single-Stepping

Race-free Single-Stepping Scheme



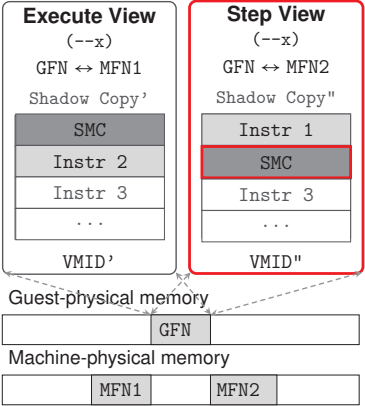
Req. 2: Stealthy Single-Stepping

Race-free Single-Stepping Scheme



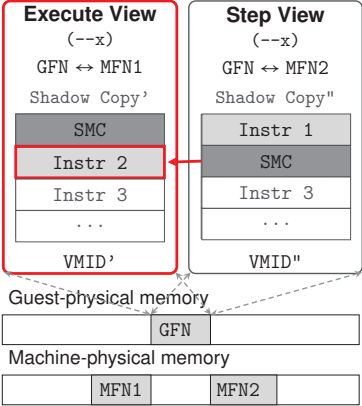
Req. 2: Stealthy Single-Stepping

Race-free Single-Stepping Scheme



Req. 2: Stealthy Single-Stepping

Race-free Single-Stepping Scheme



Req. 3: Execute-only Memory on AArch64

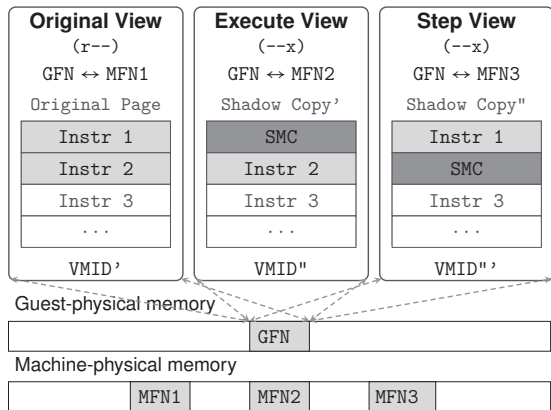
Stealthy Single-Stepping Scheme

Putting everything together (on AArch64)

- ▶ Allocate two additional views:
Execute View and **Step View**
- ▶ Duplicate the original page twice
 - ▶ Replace **Instr 1** with **SMC** in **Shadow Copy'**
 - ▶ Replace **Instr 2** with **SMC** in **Shadow Copy''**
- ▶ Map both duplicates as execute-only

On read-requests, switch to the Original View

- ▶ Satisfies integrity checks



Pitfalls of Virtual Machine Introspection

High Potential & High Maintenance

VMI is a **custom-tailored suit/Kevlar vest**

- ▶ Dependencies on **the OS**:
 - ▶ Bind the VMI tools to (an existing) OS kernel
 - ▶ Self-patching, race conditions with multi-vCPUs, etc.
 - ▶ Licensing questions
- ▶ Dependencies on **the compiler**:
 - ▶ Optimizations, function inlining, etc.
 - ▶ How to generate reliable OS profiles?
- ▶ Dependencies on the **OS profile**:
 - ▶ Incomplete and fragile profile generation
 - ▶ How to verify that a profile fits the OS?



Pitfalls of Virtual Machine Introspection

High Potential & High Maintenance

VMI is a *custom-tailored suit/Kevlar vest*

- ▶ Performance overhead:
 - ▶ VMI tools tend to “over-subscribe” to events
 - ▶ Irrelevant events must be injected into the guest



Virtual Machine Introspection

Key Takeaways

VMI shows great potential (for analysis and defense)

- ▶ Analyze and manipulate the state of guest OSES [from the outside](#)
- ▶ Compromised VMs cannot easily manipulate or delude security tools

VMI has its place, e.g., [in sandboxing/analysis environments](#)

- ▶ Controlled environments simplify maintaining fragile profiles
- ▶ Performance is not a hard requirement
- ▶ The value of the [stealth property](#) receives the highest priority

Part III

Virtualization-assisted OS Security



Virtualization-assisted OS Security

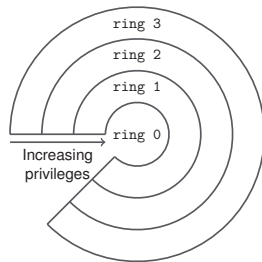
Motivation & Background

Problem: (Bounded) Hierarchical Privilege Separation

The OS kernel is responsible for:

- ▶ Protecting and isolating applications in **user space**
- ▶ **Protecting itself from unauthorized accesses**

Who protects the OS kernel from malicious entities with same privileges?



Virtualization-assisted OS Security

Motivation & Background

Design the OS kernel with virtualization-assisted security in mind

- ▶ Alleviate the strict separation between the OS and a VMM
 - ▶ Leverage virtualization extensions for defense purposes
 - ▶ Deploy a thin VMM in form of an OS subsystem (or retrospectively on-demand)
- ▶ Equip the OS subsystems with security primitives offered by the VMM
 - ▶ Partition memory and subsystems into individual security domains
 - ▶ Define flexible security policies as part of the OS
 - ▶ No need to fully export security services to the VMM
- ▶ Leverage a small and well-tested, or (ideally) a **formally verified VMM**
 - ▶ Leave the security of the highest privilege level to math!

Selective Memory Protection (xMP)

xMP in a Nutshell

Leverage virtualization extensions to **define xMP domains** in kernel and user space

1. **Partition** selected memory regions **into isolated xMP domains**
2. Empower Linux to enforce **fine-grained memory permissions** in xMP domains
3. **Protect the integrity of pointers** to xMP domains

→ Utilize Xen **alt2m** to establish efficient xMP domains

xMP Primitives

1: Partition Memory into xMP Domains

Guest-physical
memory



Machine-physical
memory

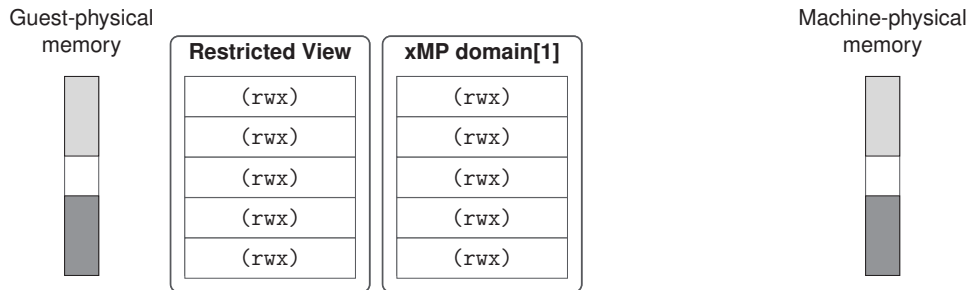


Leverage Xen `altp2m` to configure multiple disjoint xMP domains

- ▶ Only a single `altp2m` view can be active at a given time
- Propagate permissions of each xMP domain across all available `altp2m` views

xMP Primitives

1: Partition Memory into xMP Domains

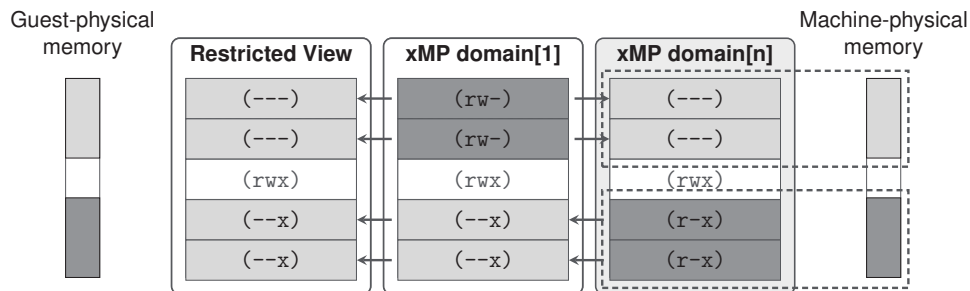


One xMP domain requires **2 altp2m views** (restricted and relaxed view)

- ▶ The **restricted view** unifies memory restrictions of all xMP domains
 - ▶ Configured as the default view on all vCPUs

xMP Primitives

1: Partition Memory into xMP Domains



For n xMP domains, we define $n + 1$ `altp2m` views

- ▶ Each $\{domain[i] \mid i \in \{1, \dots, n\}\}$
 - ▶ Relaxes the permissions of sensitive memory in xMP domain i
 - ▶ Restricts access to memory regions belonging to xMP domains $\neq i$

xMP Primitives

2: Empower Linux to Isolate Memory in xMP Domains

Equip Linux with **memory isolation primitives**

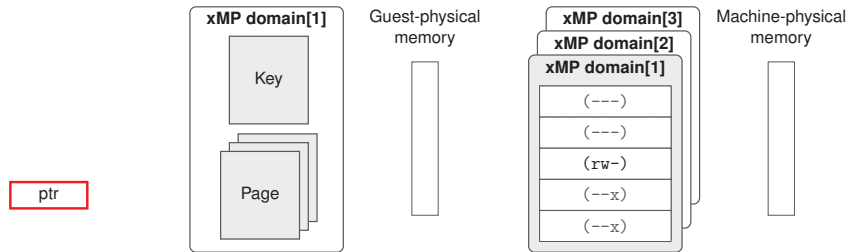
- ▶ Interface Linux with access to Xen altp2m (hypercalls)
- ▶ Govern sensitive data in isolated and disjoint xMP domains

Leverage **Intel's fast EPTP switching** and **Virtualization Exceptions (#VE)**

- ▶ Use the **VMFUNC** instruction to **dynamically switch xMP domains**
- ▶ Illegal accesses trap into the in-kernel **#VE** handler
- ▶ **No VMM interaction required**

xMP Primitives

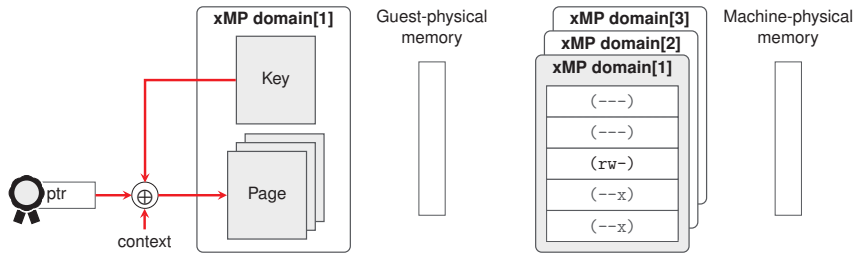
3: Context-bound Pointer Integrity



Ensure the integrity of pointers to sensitive data within xMP domains

xMP Primitives

3: Context-bound Pointer Integrity

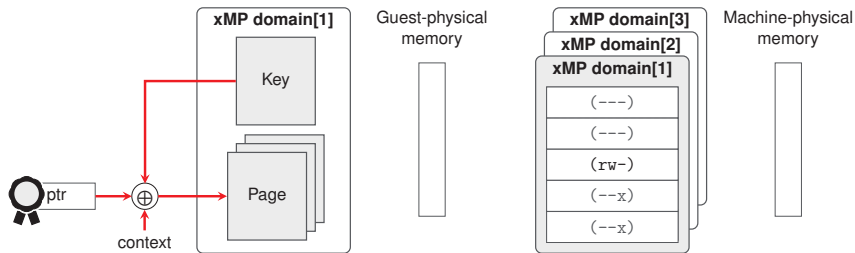


Ensure the integrity of pointers to sensitive data within xMP domains

- ▶ xMP uses (SipHash-generated) HMACs to authenticate selected pointers
 - ▶ Stores truncated HMAC in bits [48 – 63] of the pointer

xMP Primitives

3: Context-bound Pointer Integrity

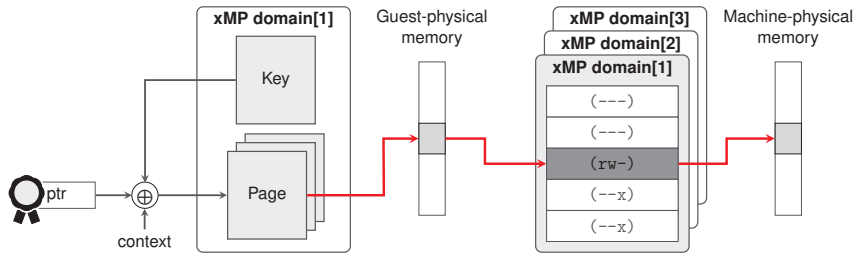


Guard read-only keys per xMP domain

- ▶ Keys can be read only inside the corresponding xMP domains
- ▶ xMP domains dedicate the same GFN for accessing different keys
 - ▶ Remap the GFN (which holds the key) of each xMP domain **to different MFNs**

xMP Primitives

3: Context-bound Pointer Integrity



Bind pointers to immutable context

- ▶ Use context that is **unique and immutable** (e.g., `&task_struct`)
- ▶ Pointer authentication succeeds only in the right context

Integrating xMP into Linux

Combine Memory Management with Virtualization Extensions

Integrate xMP primitives into the Linux memory management system

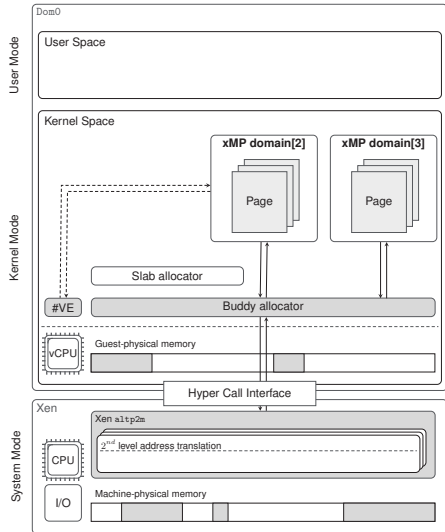
- ▶ Closely couple memory management with the capabilities of virtualization extensions

Targeted memory management components

- ▶ The (zoned) buddy allocator
- ▶ The slab allocator (kmalloc)

Establish controlled access to:

- ▶ Page tables
- ▶ Process credentials



Integrating xMP into Linux

Combine Memory Management with Virtualization Extensions

Integrate xMP primitives into the Linux memory management system

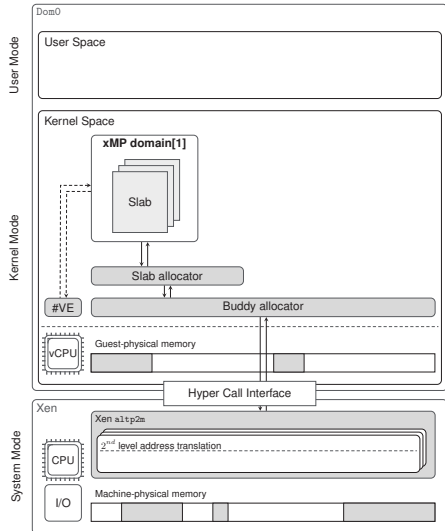
- ▶ Closely couple memory management with the capabilities of virtualization extensions

Targeted memory management components

- ▶ The (zoned) buddy allocator
- ▶ The slab allocator (`kmalloc`)

Establish controlled access to:

- ▶ Page tables
- ▶ Process credentials



Pitfalls of Virtualization-assisted OS Security

If you want something done right, ...

Retrofitting virtualization-assisted security primitives into **existing OS kernels** can be hard

- ▶ How to best partition an existing OS kernel's code/data?
 - ▶ *Subject* vs. *object* security domains
 - ▶ Type-based security domains (2 vs n-coloring schemes)
 - ▶ We are in need for privilege and data sensitivity metrics
- ▶ Determine the **right granularity** of security domains
 - ▶ Reduce information leakage across security domains
 - ▶ Avoid over-privileged security domains
- ▶ Can we automate partitioning of existing code bases?

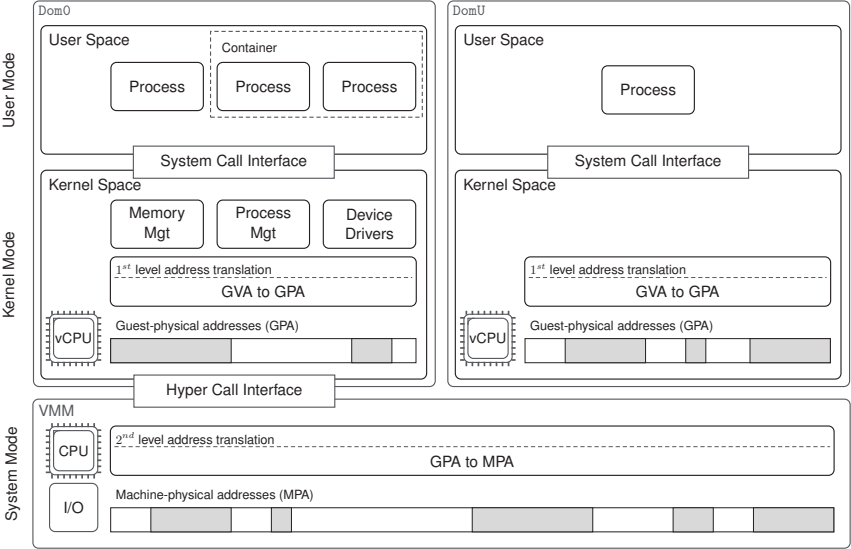
Design future OSes with virtualization in mind!

Conclusion

Food for Thought

- ▶ **Virtualization-assisted security** has not been explored to its full extent
 - ▶ Receives increasing acceptance from the industry
 - ▶ CPU manufacturers continue announcing novel HW extensions
- ▶ **Alleviate the strict separation** between the OS and a VMM
 - ▶ Leverage virtualization extensions as inherent building blocks from OS subsystems
 - ▶ Dedicate a subsystem, or deploy virtualization-assisted services via a thin VMM
- ▶ Investigate further **isolation primitives** for security-sensitive subsystems
 - ▶ For instance, isolated drivers and containers, secure memory allocators, etc.
- ▶ Investigate **formal requirements for virtualization-assisted security** architectures
 - ▶ Extend open-source ISAs and introduce novel open standards to the community
 - ▶ Influence hardware-vendors to develop dedicated features

Fin. Questions?



Contact

✉ sergej@bedrocksystems.com

🌐 <https://www.linkedin.com/in/sergej-proskurin/>

🐦 @proskurinserg