Hiding in the Shadows: Empowering ARM for Stealthy Virtual Machine Introspection

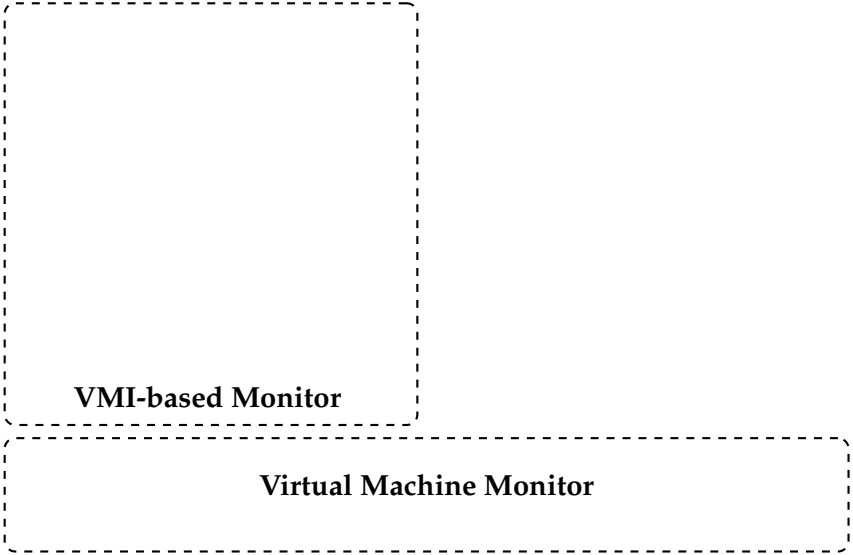
ACSAC 2018

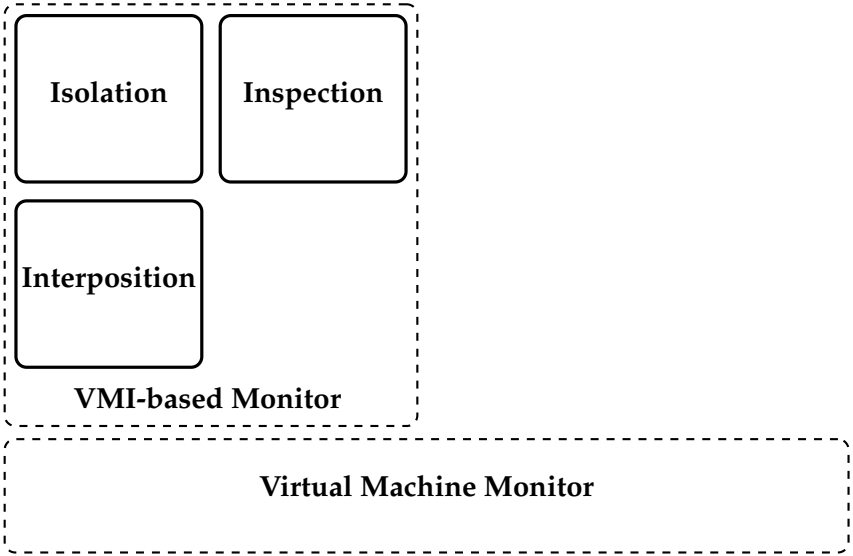
Sergej Proskurin, [1] Tamas Lengyel, [3] Marius Momeu, [1]
Claudia Eckert, [1] and Apostolis Zarras [2]

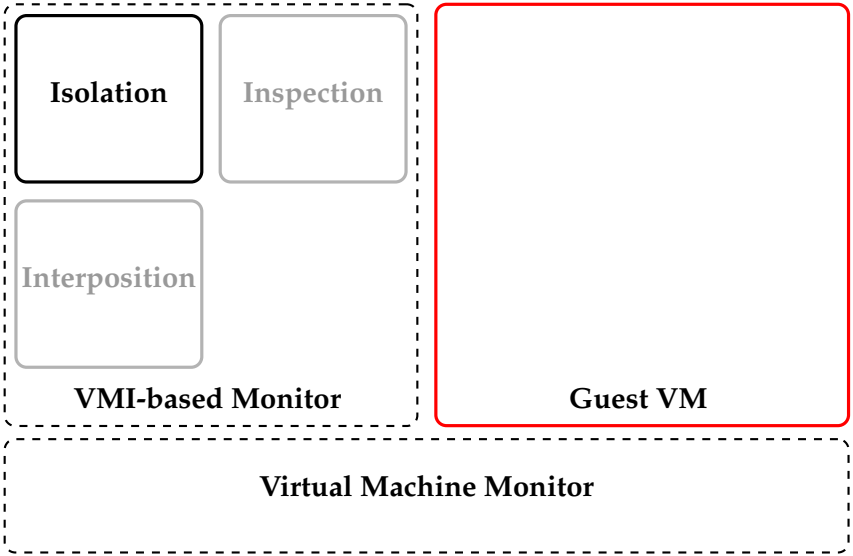
[1]Technical University of Munich
[2]Maastricht University
[3]The Honeynet Project

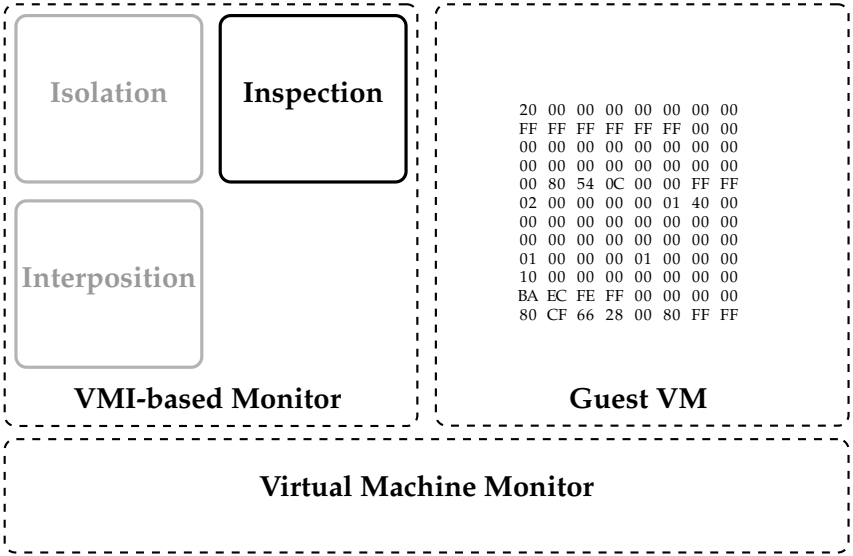
6th of December 2018

Stealthy malware analysis on ARM!


VMI-based Monitor
Virtual Machine Monitor

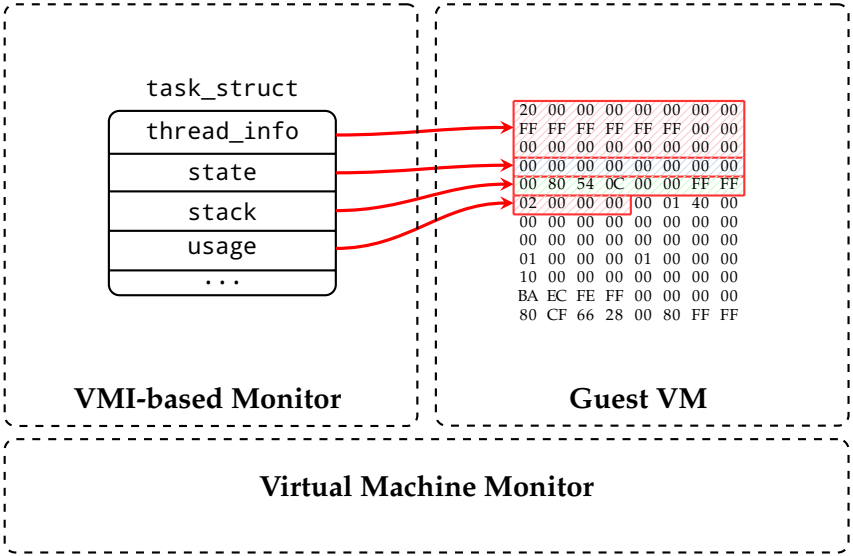

Isolation
Inspection
Interposition
VMI-based Monitor
Virtual Machine Monitor

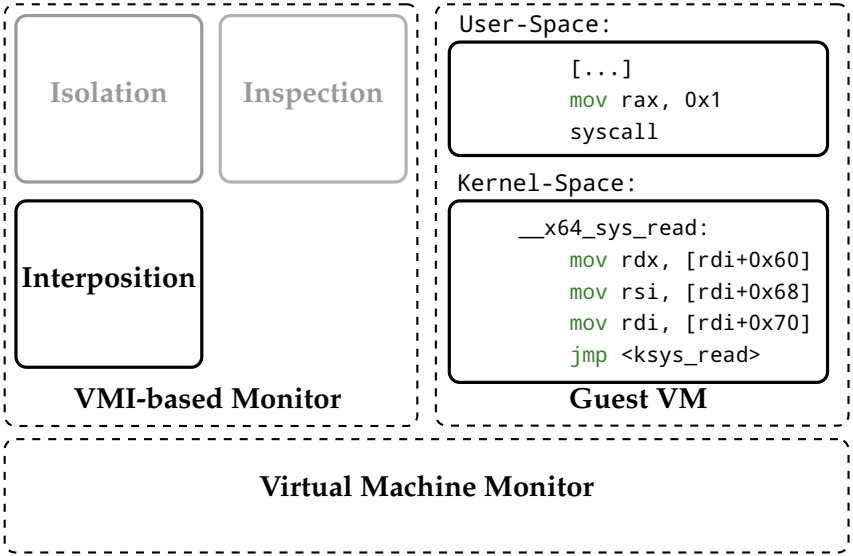

Isolation
Inspection
Interposition
VMI-based Monitor
Guest VM
Virtual Machine Monitor

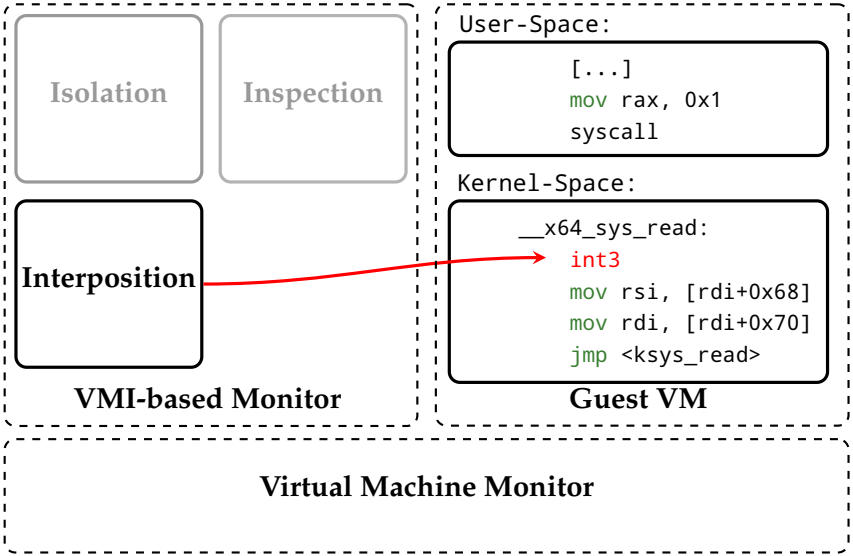

Isolation
Inspection
Interposition
VMI-based Monitor
20 00 00 00 00 00 00 00
FF FF FF FF FF FF 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 80 54 0C 00 00 FF FF
02 00 00 00 00 01 40 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
01 00 00 00 01 00 00 00
10 00 00 00 00 00 00 00
BA EC FE FF 00 00 00 00
80 CF 66 28 00 80 FF FF
Guest VM
Virtual Machine Monitor

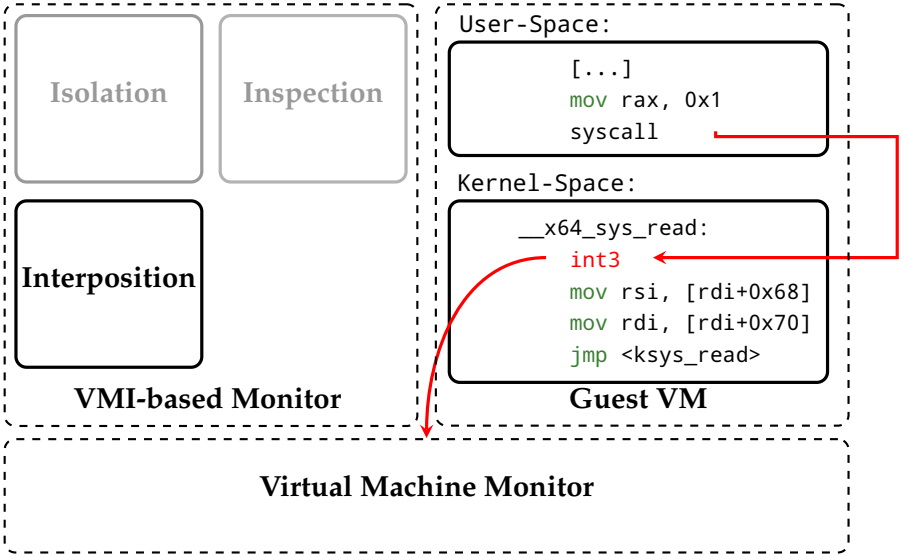
Motivation & Background

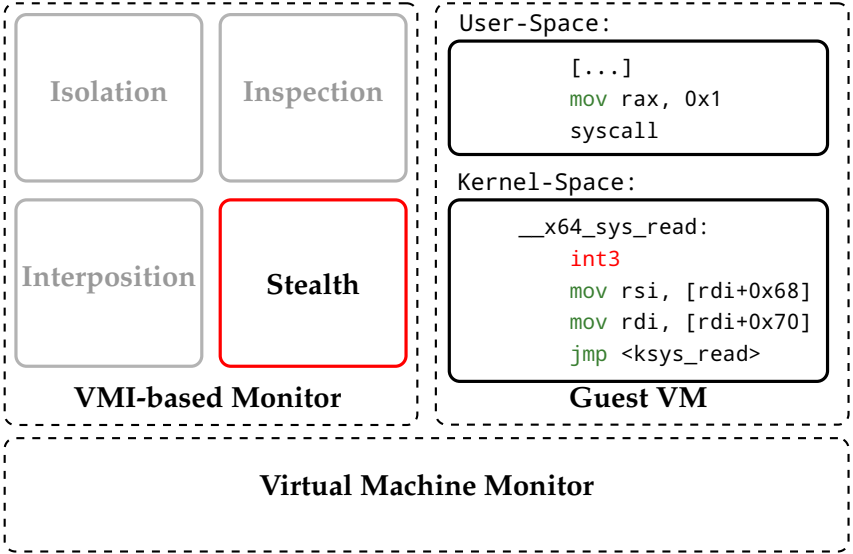
Virtual Machine Introspection Recap




Isolation
Inspection
Interposition
VMI-based Monitor
User-Space:
[...]
mov rax, 0x1
syscall
Kernel-Space:
__x64_sys_read:
mov rdx, [rdi+0x60]
mov rsi, [rdi+0x68]
mov rdi, [rdi+0x70]
jmp <ksys_read>
Guest VM
Virtual Machine Monitor

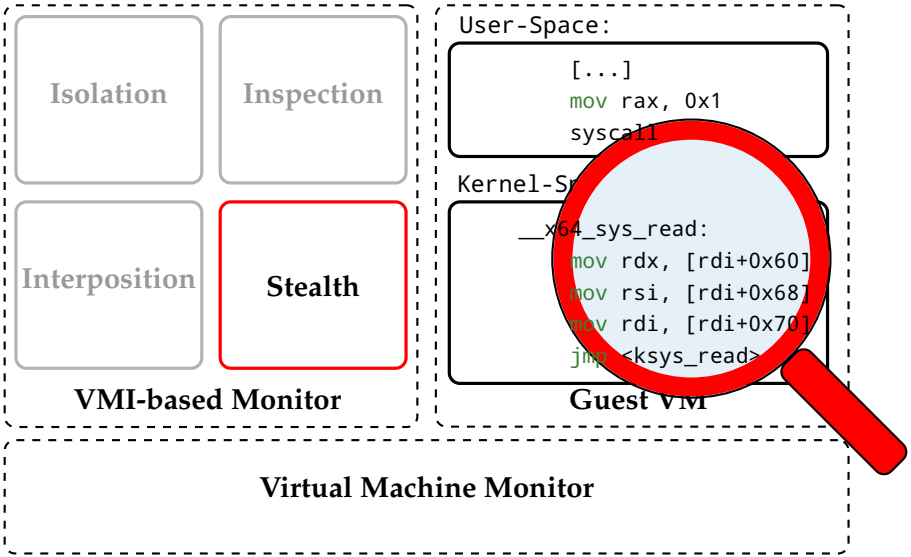

Isolation
Inspection
Interposition
VMI-based Monitor
User-Space:
[...]
mov rax, 0x1
syscall
Kernel-Space:
__x64_sys_read:
int3
mov rsi, [rdi+0x68]
mov rdi, [rdi+0x70]
jmp <ksys_read>
Guest VM
Virtual Machine Monitor


Isolation
Inspection
Interposition
VMI-based Monitor
User-Space:
[...]
mov rax, 0x1
syscall
Kernel-Space:
__x64_sys_read:
int3
mov rsi, [rdi+0x68]
mov rdi, [rdi+0x70]
jmp <ksys_read>
Guest VM
Virtual Machine Monitor


Isolation
Inspection
Interposition
Stealth
VMI-based Monitor
User-Space:
[...]
mov rax, 0x1
syscall
Kernel-Space:
__x64_sys_read:
int3
mov rsi, [rdi+0x68]
mov rdi, [rdi+0x70]
jmp <ksys_read>
Guest VM
Virtual Machine Monitor

Motivation & Background

Virtual Machine Introspection Recap



The Need for Stealthy Monitoring

Technical University of Munich

Split-personality malware

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

Perfect VM transparency is not feasible

- Insufficient to reveal virtual environments alone!

More interesting to know whether the system is being analyzed

→ Hide analysis artifacts from the guest

Requirements for Stealthy Monitoring

Technical University of Munich

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

Req. 1: Implementing Kernel Tap Points

Technical University of Munich

Use instructions as a trigger to trap into the VMM

- E.g., software breakpoints (`BRK`/`BKPT` instruction)

Better: Secure Monitor Call instruction (`SMC`)

- Guest is not able to subscribe to `SMC` traps
- `SMC` traps 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

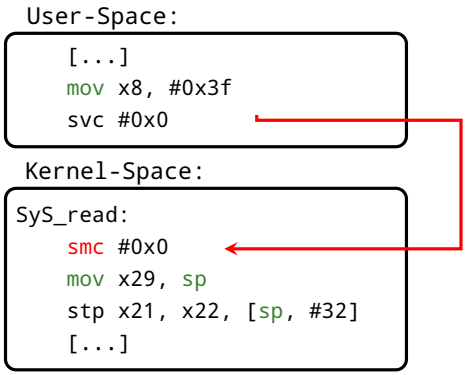
Technical University of Munich

Use instructions as a trigger to trap into the VMM

- E.g., software breakpoints (`BRK`/`BKPT` instruction)

Better: Secure Monitor Call instruction (`SMC`)

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



Req. 1: Implementing Kernel Tap Points

Technical University of Munich

Use instructions as a trigger to trap into the VMM

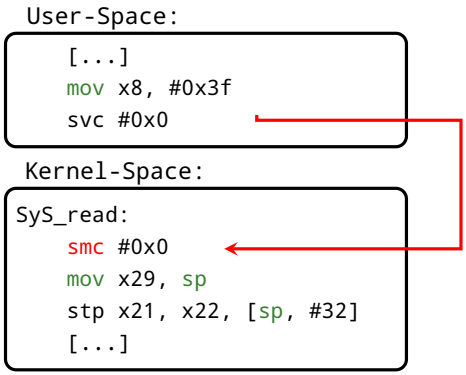
- E.g., software breakpoints (`BRK`/`BKPT` instruction)

Better: Secure Monitor Call instruction (`SMC`)

- Guest is not able to subscribe to `SMC` traps
- `SMC` traps 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



ARM does not support **stealthy** single-stepping

- Attackers can reveal the analysis framework

→ We need a novel, stealthy single-stepping scheme

Leverage the fixed-width ISA for single-stepping

- Locate instruction boundaries without a disassembler
- Use two `SMC`s to single-step one instruction

→ Multi-vCPU safe!

Req. 2: (Stealthy) Single-Stepping

Novel Single-Stepping Mechanism

ARM does not support **stealthy** single-stepping

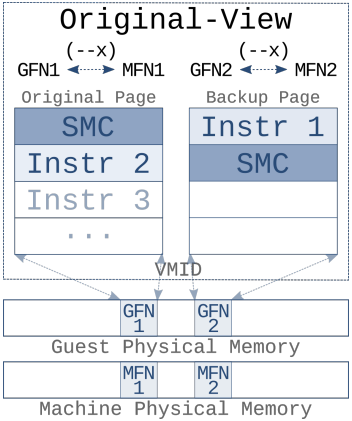
- Attackers can reveal the analysis framework

→ We need a novel, stealthy single-stepping scheme

Leverage the fixed-width ISA for single-stepping

- Locate instruction boundaries without a disassembler
- Use two `SMC`s to single-step one instruction

→ Multi-vCPU safe!



Req. 2: (Stealthy) Single-Stepping

Novel Single-Stepping Mechanism

ARM does not support **stealthy** single-stepping

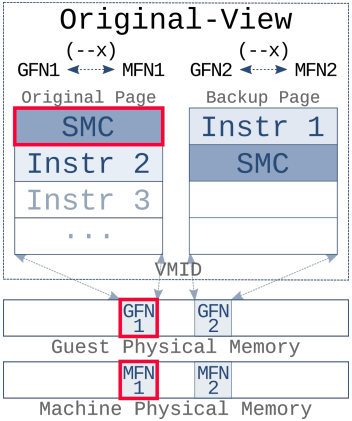
- Attackers can reveal the analysis framework

→ We need a novel, stealthy single-stepping scheme

Leverage the fixed-width ISA for single-stepping

- Locate instruction boundaries without a disassembler
- Use two `SMC`s to single-step one instruction

→ Multi-vCPU safe!



Req. 2: (Stealthy) Single-Stepping

Novel Single-Stepping Mechanism

ARM does not support **stealthy** single-stepping

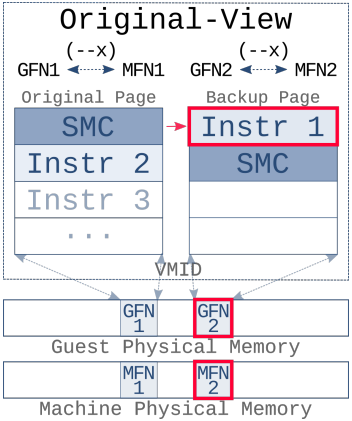
- Attackers can reveal the analysis framework

→ We need a novel, stealthy single-stepping scheme

Leverage the fixed-width ISA for single-stepping

- Locate instruction boundaries without a disassembler
- Use two `SMC`s to single-step one instruction

→ Multi-vCPU safe!



Req. 2: (Stealthy) Single-Stepping

Novel Single-Stepping Mechanism

ARM does not support **stealthy** single-stepping

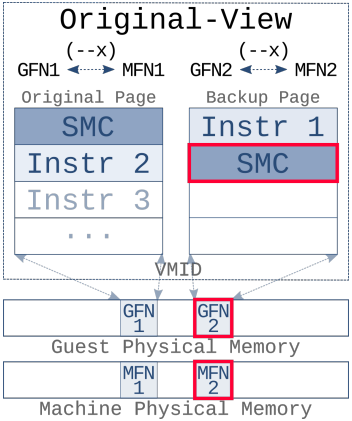
- Attackers can reveal the analysis framework

→ We need a novel, stealthy single-stepping scheme

Leverage the fixed-width ISA for single-stepping

- Locate instruction boundaries without a disassembler
- Use two `SMC`s to single-step one instruction

→ Multi-vCPU safe!



Req. 2: (Stealthy) Single-Stepping

Novel Single-Stepping Mechanism

ARM does not support **stealthy** single-stepping

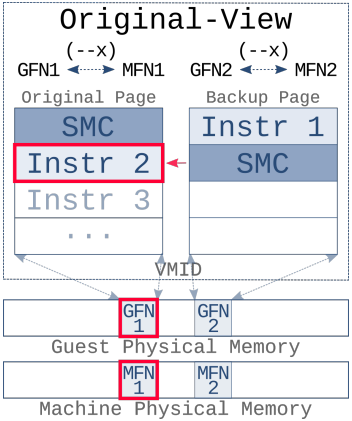
- Attackers can reveal the analysis framework

→ We need a novel, stealthy single-stepping scheme

Leverage the fixed-width ISA for single-stepping

- Locate instruction boundaries without a disassembler
- Use two `SMC`s to single-step one instruction

→ Multi-vCPU safe!



Req. 2: (Stealthy) Single-Stepping

Novel Single-Stepping Mechanism

ARM does not support **stealthy** single-stepping

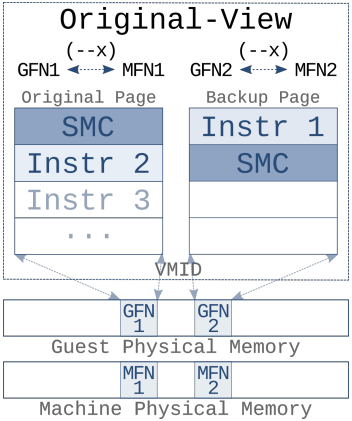
- Attackers can reveal the analysis framework

→ We need a novel, stealthy single-stepping scheme

Leverage the fixed-width ISA for single-stepping

- Locate instruction boundaries without a disassembler
- Use two `SMC`s to single-step one instruction

→ Multi-vCPU safe!



How do we hide injected `SMC` instructions?

- Employ Second Level Address Translation (SLAT)

Xen physical to machine (p2m) subsystem

- Uses Second Level Address Translation (SLAT)
- Translates guest-physical to machine-physical addresses
- Represents a single view on the guest's physical memory

Xen p2m allows to control access permissions of the guest's physical memory

- Hide injected `SMC` instructions by withdrawing read-permissions

Xen physical to machine (p2m) subsystem

- Uses Second Level Address Translation (SLAT)
- Translates guest-physical to machine-physical addresses
- Represents a single view on the guest's physical memory

Xen p2m allows to control access permissions of the guest's physical memory

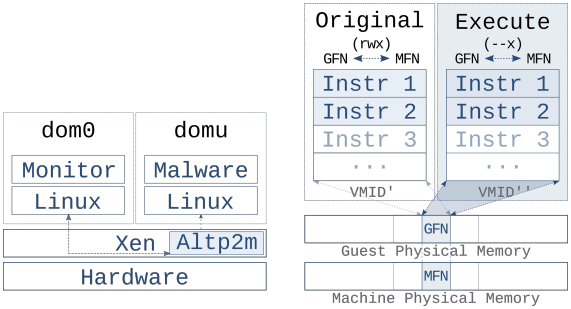
- Hide injected `SMC` instructions by withdrawing read-permissions

Issue: On integrity-checks permissions must be relaxed

- ϟ Walking the page tables is slow
- ϟ Another vCPU can access the memory without notifying the VMM

Req. 2: (Stealthy) Single-Stepping

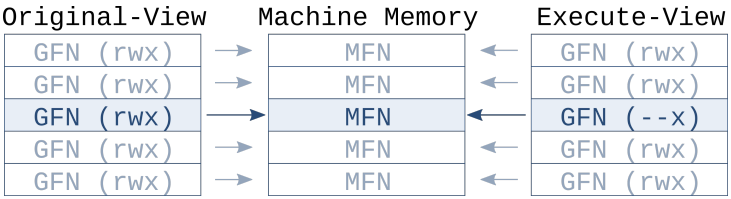
Xen altp2m Subsystem



Xen alternate p2m (altp2m) subsystem

- Maintains different views on the guest's physical memory
- Allows to allocate and assign different memory views to vCPUs

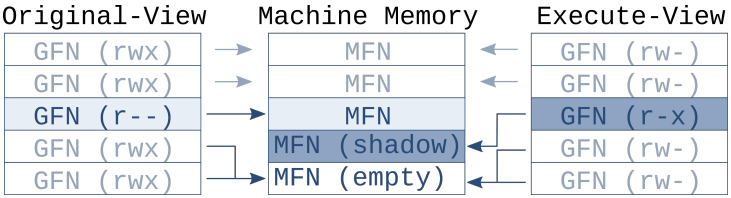
→ Switch views instead of relaxing permissions in a global view!



Xen alternate p2m (altp2m) subsystem

- Maintains different views on the guest's physical memory
- Allows to allocate and assign different memory views to vCPUs

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



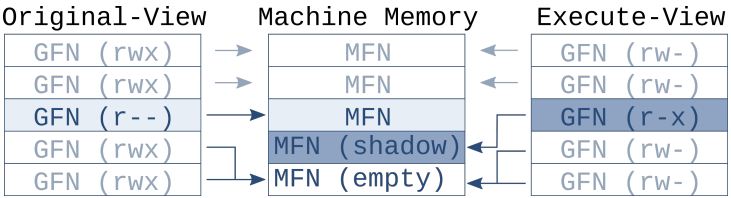
Xen alternate p2m (altp2m) subsystem

- Allows to remap same guest-physical to different machine-physical page frames

→ Facilitates, e.g., `SMC` injections in selected views

Req. 2: (Stealthy) Single-Stepping

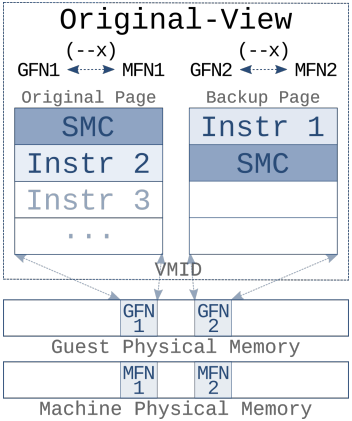
Xen altp2m Subsystem



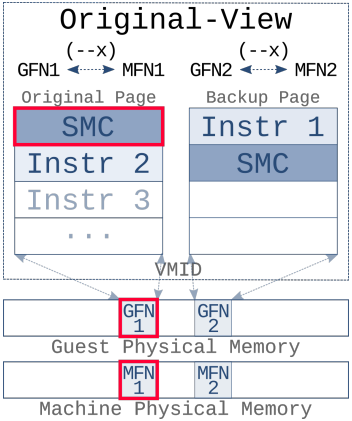
Issue: No ARM support

- Xen altp2m exclusively used on Intel CPUs

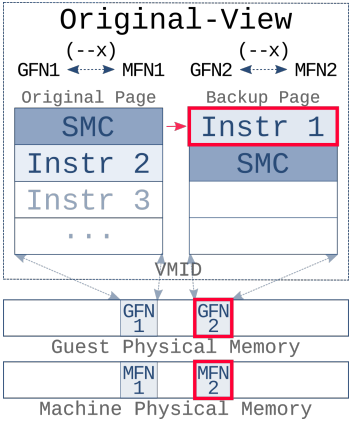
Google Summer of Code



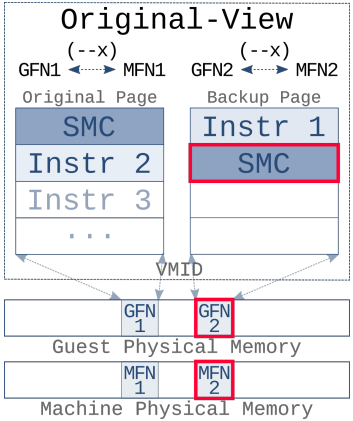
(a) Without Xen altp2m.



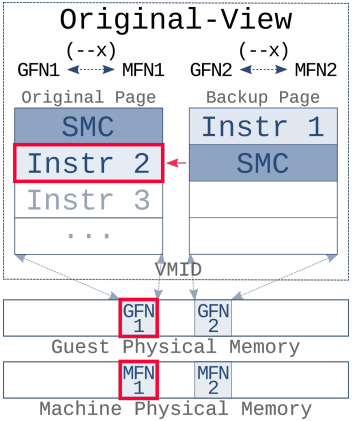
(a) Without Xen altp2m.



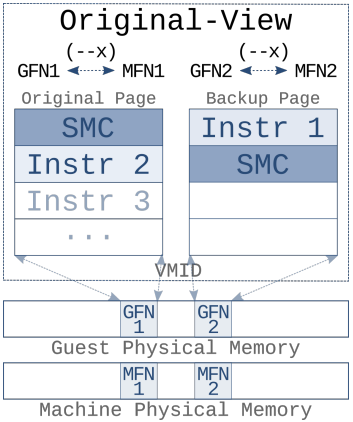
(a) Without Xen altp2m.



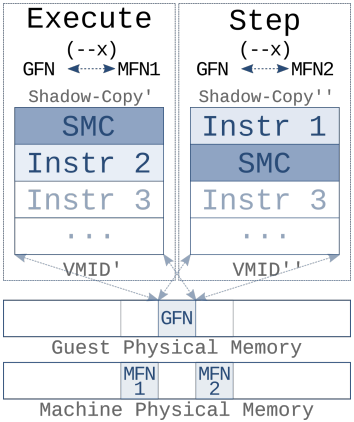
(a) Without Xen altp2m.



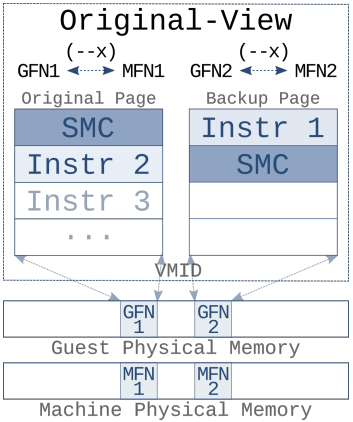
(a) Without Xen altp2m.



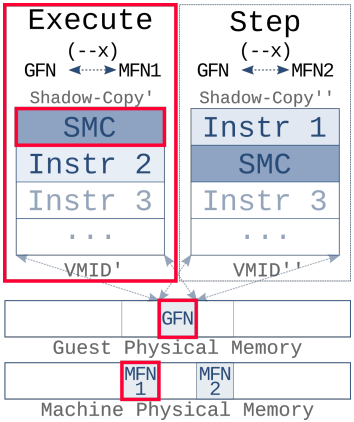
(a) Without Xen altp2m.



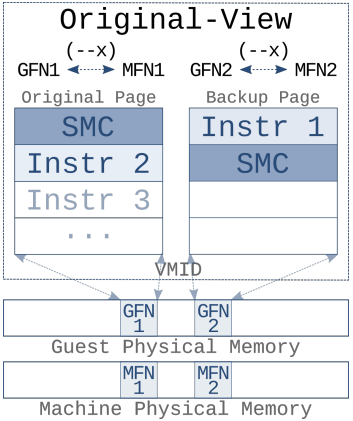
(a) With Xen altp2m.



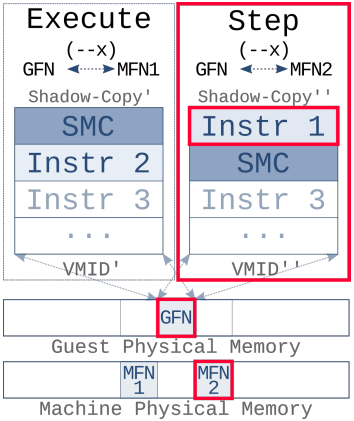
(b) Without Xen altp2m.



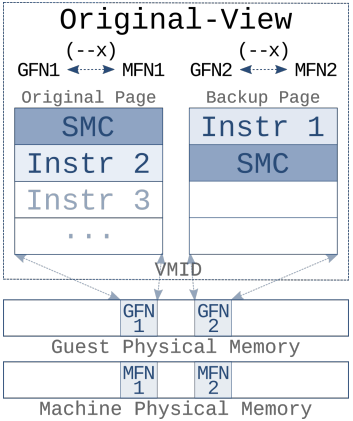
(b) With Xen altp2m.



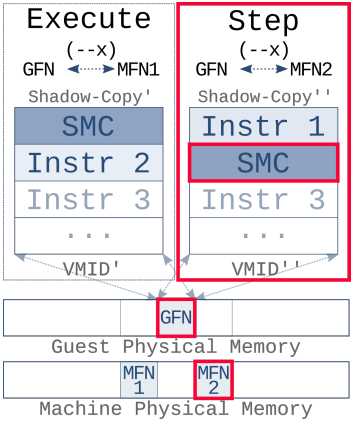
(b) Without Xen altp2m.



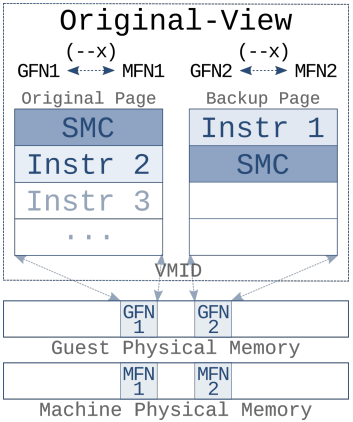
(b) With Xen altp2m.



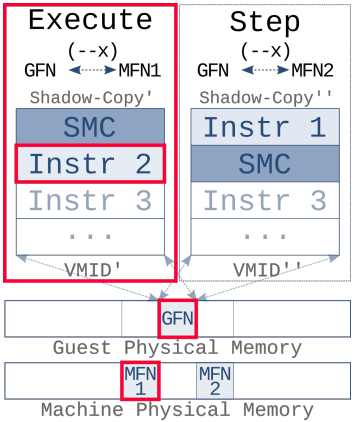
(b) Without Xen altp2m.



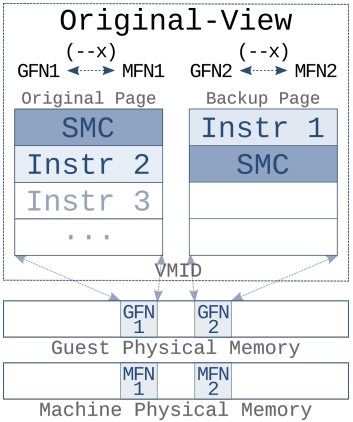
(b) With Xen altp2m.



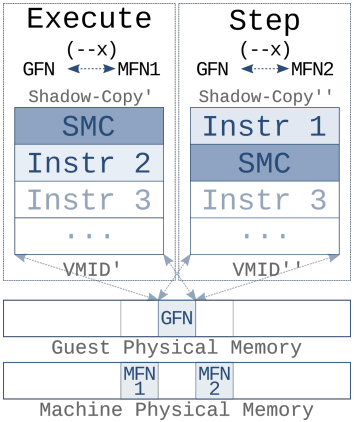
(b) Without Xen altp2m.



(b) With Xen altp2m.



(b) Without Xen altp2m.



(b) With Xen altp2m.

Req. 3: Execute-only Memory on AArch64

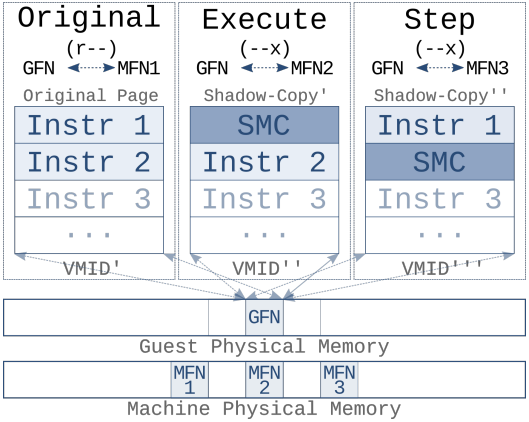
Stealthy Single-Stepping

Putting everything together (on AArch64)

- Allocate two additional views: Execute- 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



Requirements for Stealthy Monitoring on ARM

Technical University of Munich

1. Intercept the guest in kernel-space
   - ✓ Secure Monitor Call (SMC) instruction
2. A stealthy single-stepping mechanism
   - ✗ ARM has no hardware support for stealthy single-stepping
   - ✓ Stealthy single-stepping via Xen altp2m (when combined with execute-only memory)
3. Execute-only memory
   - ✓ AArch64
   - ✗ AArch32 lacks execute-only memory
   - ✓ Splitting the TLBs to hide injected tap points on AArch32

Technical University of Munich

Build the foundation for stealthy monitoring on ARM

- Implement Xen altp2m for ARM
- Equip DRAKVUF and LibVMI with our single-stepping primitives

Use DRAKVUF to trace system calls inside the guest VM

- HiKey LeMaker development board
- Guest runs a Linux v4.15 kernel
- Xen v4.11

Table: Monitoring overhead (OHD) of DRAKVUF utilizing Hardware-SS, Double-SMC-SS, and Split-TLB-SS primitives measured by Lmbench 3.0, in msec.

Benchmark	w/o	Hardware	(OHD)	Double-SMC				Split-TLB			
				Step-View	(OHD)	Backup Page	(OHD)	Step-View	(OHD)	Backup Page	(OHD)
fork+execve	1383.33	6053.67	4.38 ×	5567.33	4.02 ×	6033.00	4.36 ×	26690.66	19.29 ×	17057.00	12.33 ×
fork+exit	377.43	835.52	2.21 ×	787.14	2.09 ×	924.83	2.45 ×	5910.83	15.66 ×	4225.83	11.20 ×
fork+/bin/sh	3249.17	12542.00	3.86 ×	11672.67	3.59 ×	12737.33	3.92 ×	53134.66	16.35 ×	34231.33	10.54 ×
fstat	0.62	94.94	152.57 ×	78.65	126.40 ×	84.20	135.81 ×	103.52	166.97 ×	75.33	121.06 ×
mem read	1745.00	1692.33	0.97 ×	1692.33	0.97 ×	1738.00	1.00 ×	1730.33	0.99 ×	1735.33	0.99 ×
mem write	4687.67	4310.00	0.92 ×	4308.33	0.92 ×	4715.00	1.00 ×	4575.33	0.98 ×	4602.00	0.98 ×
open/close	5.44	202.67	37.25 ×	158.33	29.11 ×	179.26	35.95 ×	269.67	49.57 ×	184.65	33.94 ×
page fault	1.49	1.72	1.15 ×	1.74	1.16 ×	1.62	1.09 ×	1.90	1.28 ×	1.91	1.28 ×
pipe lat	12.26	371.92	30.34 ×	344.83	28.13 ×	425.28	34.69 ×	955.53	77.94 ×	482.60	39.36 ×
read	0.67	95.21	141.14 ×	79.10	117.27 ×	84.06	125.46 ×	99.34	148.27 ×	75.39	111.77 ×
select 500 fd	28.33	124.62	4.40 ×	110.23	3.89 ×	114.51	4.04 ×	124.47	4.39 ×	113.85	4.02 ×
signal handle	4.34	189.67	43.70 ×	150.33	34.64 ×	154.13	35.51 ×	178.00	41.01 ×	158.33	36.48 ×
signal install	0.51	95.00	186.27 ×	72.00	141.18 ×	75.13	147.31 ×	89.07	174.65 ×	73.73	144.58 ×
stat	2.63	99.97	38.06 ×	80.73	30.74 ×	85.30	32.43 ×	105.58	40.14 ×	83.57	31.82 ×
syscall	0.31	94.21	299.05 ×	75.15	238.55 ×	83.49	269.32 ×	98.48	317.68 ×	78.84	250.26 ×
write	0.47	95.34	203.32 ×	76.82	163.81 ×	83.86	178.43 ×	103.22	219.62 ×	73.77	157.31 ×

Conclusion

Technical University of Munich

Establish the foundation for stealthy malware analysis on ARM

- Introduce Xen altp2m to ARM
- Stealthy single-stepping approach for AArch{32 | 64}
- De-synchronize the TLB architecture on AArch32

DRAKVUF on ARM is open-source:

- `https://github.com/drakvuf-on-arm/drakvuf-on-arm`
- `https://youtu.be/mfhZBBdC-Jg` (Demo!)

Appendix 0: (Stealthy) Single-Stepping

Single-Stepping on ARM

ARM does not support **stealthy** single-stepping

→ Attackers can infer the presence of the analysis framework

AArch32: Use hardware breakpoints ("mismatching") for single-stepping

- CPU generates a debug event on instructions following the breakpoint
- ↯ Finite number of hardware breakpoints

AArch64: Use Software-Step exceptions (set `MDSCR_EL1.SS` and `PSTATE.SS` of `EL1`)

- ARM forbids access to `PSTATE.SS` in all exception levels
- ↯ Spill `PSTATE.SS` into the guest-accessible `SPSR_EL1`

Appendix 1: Xen altp2m Subsytem on Intel

Technical University of Munich

Xen altp2m exclusively used on Intel

- The VMCS has capacity for up to 512 EPTPs (memory views)
- Introduced to Xen to add support for the EPTP Switching functionality
  - Combine `VMFUNC` instruction with Virtualization Exceptions `#VE`
  - → No additional VM Exit overhead on memory violations!

External monitors can use altp2m

→ Unique tool for VMI applications

Appendix 2: No Execute-only Memory on AArch32

Splitting the TLBs

AArch32 does not support execute-only memory

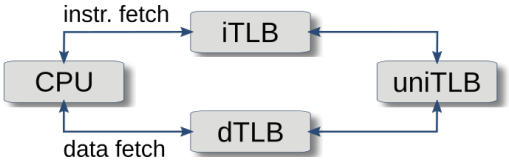
- Code-pages must be executable and readable

ARM uses VMIDs as TLB-tags to isolate translations

- Allocate two views with same `VMID` to de-synchronize the `iTLB` from the `dTLB`

Prime the TLBs in Original-View:

- `iTLB` holds the `SMC` from Execute-View
- `dTLB` holds `instr 1` from Original-View



Appendix 2: No Execute-only Memory on AArch32

Splitting the TLBs

AArch32 does not support execute-only memory

- Code-pages must be executable and readable

ARM uses VMIDs as TLB-tags to isolate translations

- Allocate two views with same `VMID` to de-synchronize the `iTLB` from the `dTLB`

Prime the TLBs in Original-View:

- `iTLB` holds the `SMC` from Execute-View
- `dTLB` holds `instr 1` from Original-View