Dynamic Virtual Address Range Adjustment for Intra-Level Privilege Separation on ARM

Yeongpil Cho, Donghyun Kwon, Hayoon Yi, Yunheung Paek

Seoul National University
System Software

For example,
- Operating System
- Hypervisor

System Software plays roles of
- Resource Manager
- Trusted Computing Base

A variety of system software has the monolithic design
- so, the entire can suffer from small exploits.
Privilege Separation

One of fundamental security principles
- Protect security critical parts by separating from the others
  - ex) key management, page table management, system monitoring, ...

How to enforce this principle to system software?
- Relying on higher privileged entity

One fundamental question
“how to enforce this principle to the higher privileged system software?”
Intra-Level Privilege Separation

- Divide the monolithic body of system software into the outer domain and inner domain
  - Two domains run at the physically same but logically different privilege level
  - Two domains have asymmetric memory view

Two core mechanisms of intra-level privilege separation
- intra-level isolation mechanism
  - prevent the outer domain from accessing the inner domain
- domain switching mechanism
  - transfer control between the outer and inner domains
Motivation of our work

- To efficiently and securely implement two core mechanisms, a hardware feature for memory protection is used.

- The type of system software that can be supported is determined by the used hardware features.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Architecture</th>
<th>Key Hardware Feature</th>
<th>Target System Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyperSafe [S&amp;P’10]</td>
<td>x86 64-bit</td>
<td>Write-Protection</td>
<td>Hypervisor, Normal OS*</td>
</tr>
<tr>
<td>Nested Kernel</td>
<td>x86 64-bit</td>
<td>Write-Protection</td>
<td>Normal OS, Hypervisor*</td>
</tr>
<tr>
<td>[ASPLOS’15]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKEE [NDSS’16]</td>
<td>ARM 32-bit</td>
<td>TTBCR</td>
<td>Normal OS, Secure OS*</td>
</tr>
<tr>
<td>SKEE [NDSS’16]</td>
<td>ARM 64-bit</td>
<td>Extended Paging</td>
<td>Normal OS</td>
</tr>
</tbody>
</table>

*: not mentioned in the paper, but can be supported with the same technique.
Motivation of our work

- ARM’s 64-bit architecture (a.k.a AArch64) is widely used in mobile devices.

- Various types of system software coexist to enrich the functionality of the AArch64-based devices.
  - Exception Level has the same meaning as Privilege Level.

<table>
<thead>
<tr>
<th>Exception Level</th>
<th>Normal World</th>
<th>Secure World</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal OS</td>
<td>Secure OS</td>
</tr>
<tr>
<td>2</td>
<td>Hypervisor</td>
<td></td>
</tr>
</tbody>
</table>

- Not only normal OS but also secure OS and hypervisor suffer from exploits.
A technique that can enforce intra-level privilege separation to a variety of system software on AArch64

<table>
<thead>
<tr>
<th>Exception Level 1</th>
<th>Exception Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal World</td>
<td>Secure World</td>
</tr>
<tr>
<td>Normal OS</td>
<td>Secure OS</td>
</tr>
<tr>
<td>Outer domain</td>
<td>Outer domain</td>
</tr>
<tr>
<td>Inner domain</td>
<td>Inner domain</td>
</tr>
<tr>
<td>Hypervisor</td>
<td></td>
</tr>
</tbody>
</table>

To achieve the goal of Hilps, two core mechanisms for intra-level isolation and domain switching must be applicable regardless of exception level.
Hilps

AArch64 contains a hardware feature that allows dynamically adjusting valid virtual address range at each exception level

- Hilps can create a special memory region that is temporally hidden from other memory regions
Hilps

Outer Domain

Inner Domain

- Page Table Management
- System Control Register Configuration
- Security Application

Intra-level Isolation Mechanism

- Outer Domain Region
- Inner Domain Region
- Valid Virtual Address Range

Domain Switching Mechanism

- Outer Domain Region
- Inner Domain Region
- Valid Virtual Address Range
Background: Address Translation on AArch64

- System control registers for address translation
  - TTBRx_ELx (Translation Table Base Register)
    - points to the base address of the current page table
  - TCR_ELx (Translation Control Register)
    - controls address translation

![Virtual Address Space Diagram]

- TTBR1_EL1 region
- Never mapped
- TTBRO_ELx region

- Points to the base address:
  - \(0xFFFFFFFF_FFFF_FFFF\)
  - \(0xFFFF_0000_0000_0000\)
  - \(0x0000_FFFF_FFFF_FFFF\)
  - \(0x0\)
Each exception level has its own control registers

<table>
<thead>
<tr>
<th>Exception Level</th>
<th>Normal World</th>
<th>Secure World</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal OS</td>
<td>Secure OS</td>
</tr>
<tr>
<td></td>
<td>TTBR1_EL1</td>
<td>TTBR1_EL1</td>
</tr>
<tr>
<td></td>
<td>TCR_EL1</td>
<td>TCR_EL1</td>
</tr>
<tr>
<td>2</td>
<td>Normal OS</td>
<td>Secure OS</td>
</tr>
<tr>
<td></td>
<td>TTBR0_EL2</td>
<td>TTBR1_EL1</td>
</tr>
<tr>
<td></td>
<td>TCR_EL2</td>
<td>TCR_EL1</td>
</tr>
<tr>
<td>Hypervisor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Background: Virtual Address Range Adjustment

- TxSZ-field of TCR_ELx determines the valid range of the virtual address space translated by the paired TTBRx_ELx

![Diagram of TCR_EL1 and TCR_EL2 with circled fields](image)
Change of valid virtual address range depending on TxsSZ-field

- TTBR1_EL1 region
- Never mapped
- TTBR0_ELx region

- $0xFFFF_FFFF_FFFF$
- $2^{64} - 2^{(64 - TCR_{EL1,T1SZ})}$
- $0xFFFF_0000_0000_0000$
  (Boundary, when $TCR_{EL1,T1SZ} = 16$)
- $0x0000_{FFFF_FFFF_FFFF}$
  (Boundary, when $TCR_{ELx,T0SZ} = 16$)
- $2^{(64 - TCR_{ELx,T0SZ})} - 1$
- $0x0$
Background: Virtual Address Range Adjustment

- Typically, multi-level page tables are used for effective management
  - 1st-level page table is directly referenced by virtual address
- When valid virtual address range changes, the number of valid 1st-level page table entries also varies proportionally

![Diagram of virtual to physical address mapping]

- Expanded Virtual Address
- TTBR
- 1st-level page table
- 2nd-level page table
- Physical Address
Intra-level isolation mechanism

- System software that runs with TTBR0_ELx
  - ex) Hypervisor

- Valid virtual address space and valid 1st-level page table entries change in the same direction

Valid Virtual Address Space

<table>
<thead>
<tr>
<th>base addr</th>
<th>0x0</th>
<th>0x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1FFF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Valid 1st-level page table entries

<table>
<thead>
<tr>
<th>base addr</th>
<th>0x0</th>
<th>0x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3FFF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expand valid VA range

Normal OS

Secure OS
Intra-level isolation mechanism

- System software that runs with TTBR0_ELx
  - ex) Hypervisor

- Valid virtual address space and valid 1st-level page table entries change in the same direction

<table>
<thead>
<tr>
<th>Valid Virtual Address Space</th>
<th>Valid 1st-level page table entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0 to 0x1FFF</td>
<td>0x0 to 0x3FFF</td>
</tr>
<tr>
<td>base addr 0x0</td>
<td>base addr 0x0</td>
</tr>
<tr>
<td>expand valid VA range</td>
<td></td>
</tr>
<tr>
<td>Inner Domain</td>
<td></td>
</tr>
</tbody>
</table>
Intra-level isolation mechanism

- System software that runs with TTBR1_EL1
  - ex) Normal OS and secure OS

- Valid virtual address space and valid 1st-level page table entries change in the opposite direction

---

Valid Virtual Address Space

- Base addr: 0xFFFF
- 0xE000
- Expand valid VA range

Valid 1st-level page table entries

- VA[12]:
  - 1
  - 0
- VA[12:13]:
  - 3
  - 0

---

Summary of Exception Levels:

- Exception Level 1:
  - Normal OS
  - Secure OS
- Exception Level 2:
  - Hypervisor
**Intra-level isolation mechanism**

- **System software that runs with TTBR1_EL1**
  - ex) Normal OS and secure OS

- **Valid virtual address space and valid 1st-level page table entries change in the opposite direction**

![Diagram showing the intra-level isolation mechanism with valid virtual address space and 1st-level page table entries.](attachment:diagram.png)
Intra-level isolation mechanism

- System software that runs with TTBR1_EL1
  - ex) Normal OS and secure OS

- Valid virtual address space and valid 1st-level page table entries change in the opposite direction

<table>
<thead>
<tr>
<th>Valid Virtual Address Space</th>
<th>Valid 1st-level page table entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>base addr</td>
<td>Valid 1st-level page table entries</td>
</tr>
<tr>
<td>0xFFFF</td>
<td>VA[12] 1</td>
</tr>
<tr>
<td>0xE000</td>
<td>0</td>
</tr>
<tr>
<td>expand valid VA range</td>
<td></td>
</tr>
<tr>
<td>base addr</td>
<td>Valid 1st-level page table entries</td>
</tr>
<tr>
<td>0xFFFF</td>
<td>VA[12:13] 7</td>
</tr>
<tr>
<td>0x8000</td>
<td>0</td>
</tr>
</tbody>
</table>

Inner Domain
Domain switching mechanism

A way to enter the inner domain

```
mrs x5, DAIF
stp x30, x5, [sp, #-16]!
msr DAIFset, 0x3
1:
    mrs x5, tcr_el1
    and x5, x5, #0xfffffffffdffff
    orr x5, x5, #0x400000
    msr tcr_el1, x5
    isb
    mov x6, #0xc03f
    mov x7, #0x1b
    movk x6, #0xc07f, lsl #16
    movk x7, #0x8059, lsl #16
    and x5, x5, x6
    cmp x5, x7
    b.ne 1b
    mrs x6, mpidr_el1
    ubfx x5, x6, #8, #4
    and x6, x6, #0xf
    orr x6, x6, r5, lsl #2
    add x6, x6, #1
    adrp x5, InnerDomain_stack
    add x5, x5, x6, lsl #12
    mov x6, sp
    mov sp, x5
    str sp, x5
    adrp x5, InnerDomain_handler
    blr x5
```
Domain switching mechanism

A way to return back to the outer domain

- ldp x6, [sp], #8
- mov sp, x6
- mrs x5, tcr_el1
- and x5, x5, #0xffffffffffffffffff
- orr x5, x5, #0x20000
- msr tcr_el1, x5
- mov x6, #0xc03f
- mov x7, #0x1b
- movk x6, #0xc07f, lsl #16
- movk x7, #0x801b, lsl #16
- and x5, x5, x6
- cmp x5, x7
- b.ne 2b
- ldp x30, x5, [sp], #16
- msr DAIF, x5
- isb
- ret

Switch to the outer domain stack

Configure TCR to reduce valid virtual address range and hide the inner domain

Verify the value of TCR

Enable interrupts
Evaluation

- V2M-Juno r1 platform
  - Cortex-A57 1.15 GHz dual-core
  - Cortex-A53 650 MHz quad-core
  - 2 GB of DRAM

- Target
  - AArch64 Linux Kernel 3.10 of Android 5.1.1
Evaluation

- Round-Trip Cycles between the outer and inner domains
  - Big core: 424 cycles, Little core: 210 cycles
- LMbench to measure the kernel performance
  
  ![Graph showing LMbench performance](image)
  
  Avg: 9.77%

- Application benchmarks to measure the system performance
  
  ![Graph showing application benchmarks](image)
  
  Avg: 0.84%
Thank you!
Intra-level isolation mechanism
Intra-level isolation mechanism
Intra-level isolation mechanism

An attacker in the outer domain would access the inner domain through the cached TLB entries.

How to prevent the outer domain from referencing cached TLB entries for the inner domain?

- at Exception Level 1
  - use different ASIDs between the two domains

- at Exception Level 2
  - invalidate cached TLB entries when switching between the two domains
An attacker would cause a malicious interrupt to bypass the verification process for TCR and access to the inner domain
- thwarting this attack by inserting a code snippet to verify the value of TCR

Subverting control-flow

Entry gate of IDC

Jump

Interrupt disabling
TCR configuration
Verification of TCR
...

Interrupt

Interrupt Vector Table

at EL1:
  if TCR_EL1.T1SZ == 27
at EL2 or EL3:
  if TCR_ELx.T0SZ == 27

Yes

Interrupt Handler

No

Halting the system

Exit gate of IDC

Jump

... TCR configuration
Verification of TCR
Interrupt enabling

Interrupt

Inner Domain

28
Efficiency of the domain switching mechanism

- Round-Trip Cycles between the outer and inner domains
  - Measured by the performance monitor provided by AArch64

<table>
<thead>
<tr>
<th></th>
<th>Big core with ASID</th>
<th>Big core with TLB inv.</th>
<th>Little core with ASID</th>
<th>Little core with TLB inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC</td>
<td>424</td>
<td>832</td>
<td>210</td>
<td>249</td>
</tr>
</tbody>
</table>
Efficiency of Hilps

LMBench to measure the kernel performance

<table>
<thead>
<tr>
<th></th>
<th>Big core with ASID</th>
<th>Big core with TLB inv.</th>
<th>Little core with ASID</th>
<th>Little core with TLB inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>null syscall</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>2.33 %</td>
<td>2.33 %</td>
</tr>
<tr>
<td>open/close</td>
<td>-0.31 %</td>
<td>1.10 %</td>
<td>0.16 %</td>
<td>0.71 %</td>
</tr>
<tr>
<td>stat</td>
<td>-0.38 %</td>
<td>0.38 %</td>
<td>0.99 %</td>
<td>1.8 %</td>
</tr>
<tr>
<td>handler inst</td>
<td>0.00 %</td>
<td>1.47 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
</tr>
<tr>
<td>handler ovh</td>
<td>0.31 %</td>
<td>1.84 %</td>
<td>-0.67 %</td>
<td>-0.17 %</td>
</tr>
<tr>
<td>pipe latency</td>
<td>11.40 %</td>
<td>43.48 %</td>
<td>6.89 %</td>
<td>19.10 %</td>
</tr>
<tr>
<td>page fault</td>
<td>27.66 %</td>
<td>102.13 %</td>
<td>31.32 %</td>
<td>96.44 %</td>
</tr>
<tr>
<td>fork+exit</td>
<td>19.20 %</td>
<td>61.89 %</td>
<td>14.57 %</td>
<td>44.95 %</td>
</tr>
<tr>
<td>fork+execv</td>
<td>19.42 %</td>
<td>55.34 %</td>
<td>12.44 %</td>
<td>41.71 %</td>
</tr>
<tr>
<td>mmap</td>
<td>20.36 %</td>
<td>71.85 %</td>
<td>11.45 %</td>
<td>44.35 %</td>
</tr>
<tr>
<td>average</td>
<td>9.77 %</td>
<td>33.95 %</td>
<td>7.95 %</td>
<td>25.12 %</td>
</tr>
</tbody>
</table>

Application benchmarks to measure the system performance

<table>
<thead>
<tr>
<th></th>
<th>with ASID</th>
<th>with TLB inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF-Bench</td>
<td>2.68 %</td>
<td>12.96 %</td>
</tr>
<tr>
<td>GeekBench</td>
<td>single core</td>
<td>-0.21 %</td>
</tr>
<tr>
<td></td>
<td>multi core</td>
<td>0.59 %</td>
</tr>
<tr>
<td>Quadrant</td>
<td>0.56 %</td>
<td>-0.02 %</td>
</tr>
<tr>
<td>Smartbench</td>
<td>productivity</td>
<td>2.07 %</td>
</tr>
<tr>
<td></td>
<td>gaming</td>
<td>1.74 %</td>
</tr>
<tr>
<td>Vellamo</td>
<td>browser</td>
<td>0.07 %</td>
</tr>
<tr>
<td></td>
<td>metal</td>
<td>-0.13 %</td>
</tr>
<tr>
<td>Antutu</td>
<td>0.17 %</td>
<td>1.79 %</td>
</tr>
<tr>
<td>aberage</td>
<td>0.84 %</td>
<td>1.71 %</td>
</tr>
</tbody>
</table>