SKEE: A Lightweight Secure Kernel-level Execution Environment for ARM

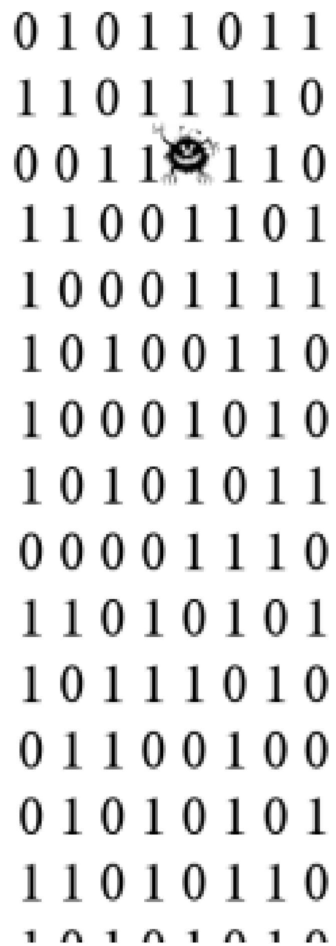
Ahmed M Azab, Kirk Swidowski, Rohan Bhutkar, Jia Ma, Wenbo Shen, Ruowen Wang, Peng Ning

Samsung KNOX R&D, Samsung Research America

Samsung Knox

Motivation

- Operating system kernels still suffer from exploits
  - CVE-20XX-XXXX
- Security tools
  - Monitor and protect the kernel
  - May have large code base
  - May introduce vulnerabilities
- Isolation is a key requirement for hosting security tools



Samsung Knox

Motivation (cont.)

- Previous approaches
  - Host security tools in hypervisors and hardware security features
    - *Designed with different objectives*
    - *Increase TCB size, increase attack surface*

  - Hypervisors and hardware security features may be compromised
    - *Due to the vulnerabilities introduced by security tools*
    - *Worse than kernel being compromised*
    - *Undermine the overall system security*

Secure Kernel-level Execution Environment

- Lightweight **in-kernel** isolation
  - Run at the same privilege level as kernel
  - Safe from potential kernel vulnerabilities
  - No requirement of active involvement from higher privileged layers
- Ability to inspect kernel state
  - Full access to entire kernel memory
  - Event driven monitoring
- Secure context-switching
  - Entry point exposed to the kernel yet secure from attacks

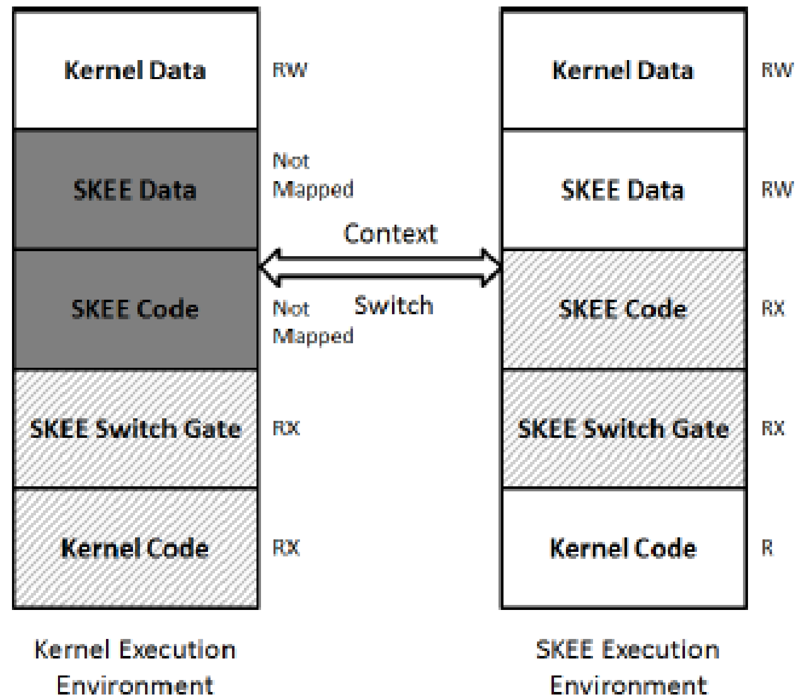
Scope

- Assumptions
  - The system is booted securely
  - The kernel code is validated and protected
    - *No kernel code injection*
    - *Valid assumption using existing techniques (e.g., W^X, DEP, PXN)*
- Threat model
  - All data attacks against the kernel are considered
    - *Including code-reuse attacks and non-control data modification*
  - SKEE guarantees a fully compromised kernel cannot:
    - *Revoke the isolation*
    - *Compromise the context switching*

Samsung Knox

SKEE Design

- Basic idea
  - A new self-protected virtual address space
- Both address spaces are initialized at boot up time
  - Secure boot is required
- Three basic requirements
  - Isolation
  - Secure context switching
  - Kernel monitoring and protection



Samsung Knox

Isolation

- Create a protected address space
  - Instrument the kernel translation tables
    - *Carve out SKEE's physical memory range*
- Restrict kernel access to the MMU
  - Revoke write access to kernel translation tables
    - *Enforce W^X protection, DEP and PXN of user code*
  - Remove op codes of certain instructions from kernel code
    - *E.g., set TTBR value, disable the MMU*
  - The kernel is forced to request MMU operations from SKEE
    - *Inspected to guarantee the isolation*

Secure Context Switching

- Atomic → Execution never returns to kernel while SKEE is accessible
  - Potential attacks
    - *Jump to the middle of the switch gate*
    - *Interrupt the switching logic execution*

- Deterministic →The switch gate shows same behavior regardless of:
  - Current system state
  - Input parameters

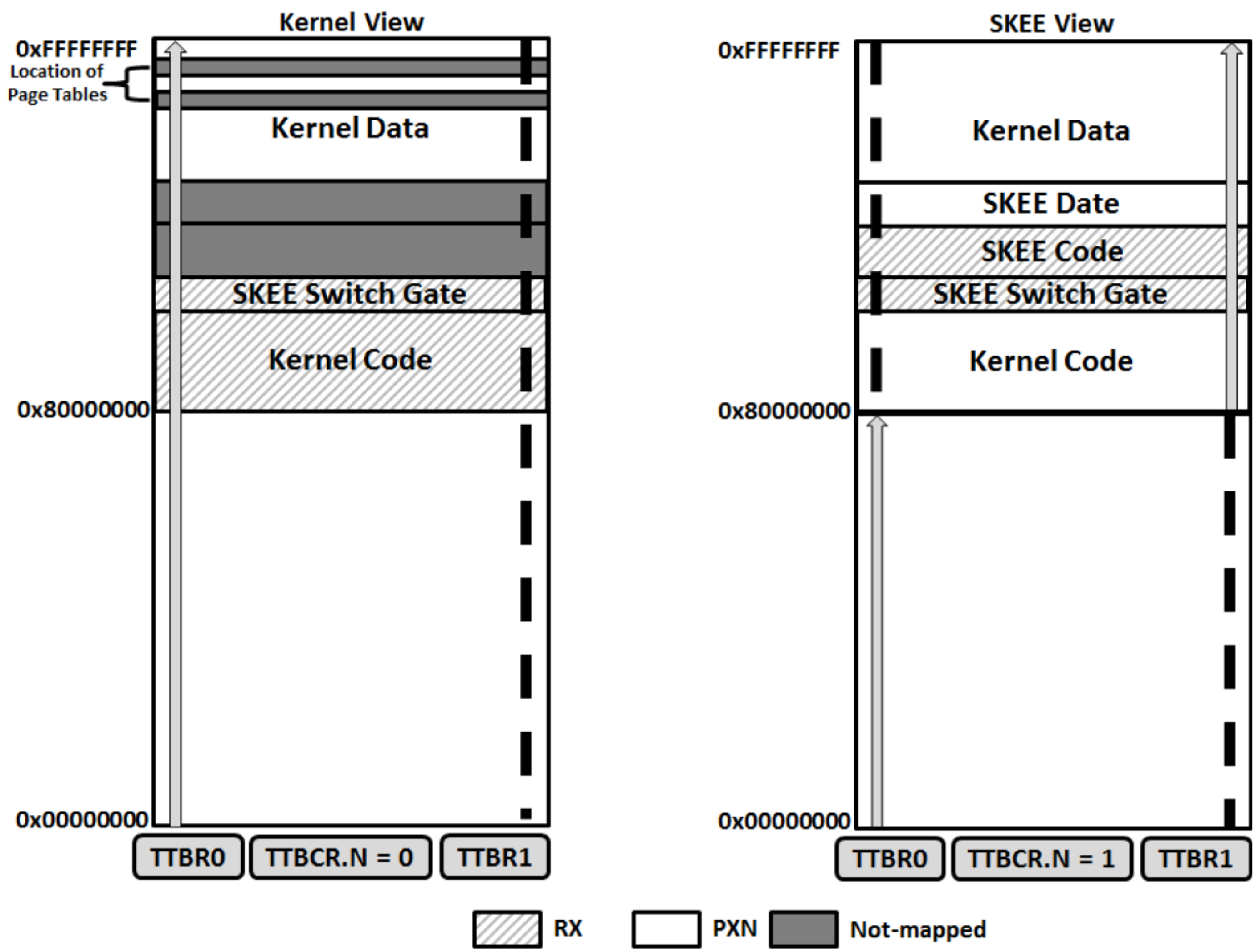
- Exclusive → The switch gate is the only entry point to SKEE

Secure Switching on 32-bit ARMv7

- Memory management in ARMv7
  - Two translation table base registers: TTBR0 & TTBR1
    - *TTBR holds the page table base, the same with CR3 in x86*
- Challenge
  - Cannot load values into TTBR0 & TTBR1 in kernel directly
    - *Compromise the isolation by loading unverified page tables*
- Solution:
  - Use dedicated registers for the kernel and SKEE
    - *Valid technical assumption (Android linux kernel only uses TTBR0)*
  - Context switching is done by updating TTBCR.N
    - *No direct value loading to TTBR*
    - *Non-zero value maps SKEE, zero value maps the kernel*

Samsung Knox

Secure Switching on 32-bit ARMv7 (Cont.)



Samsung Knox

ARMv7 Switch Gate

- Lines 2-5
  - Disable interrupts
- Lines 7-10
  - Load TTBCR
- Lines 12 and 13
  - Invalidate the TLB
- Line 15
  - Jumps to SKEE
- Exit in reverse order

```
1  /* Start of the SKEE Entry Gate */
2  mrs     r0, cpsr                 // Read the status register
3  push    {r0}                     // Save the status register value
4  orr     r0, r0, #0x1c0           // Set the mask interrupts bits
5  msr     cpsr, r0                 // load the modified value
6
7  mov     r0, #0x11
8  isb                              // Syncronization barrier
9  mcr     p15, 0, r0, c2, c0, 2    // Modify the TTBCR to activate SKEE
10 isb
11
12 mcr     p15, 0, r0, c8, c7, 0    // TLB invalidate
13 isb
14
15 bl      skee_entry               // Jump to SKEE entry point
16 /* End of the SKEE Entry Gate */
17
18 /* Start of the SKEE Exit Gate */
19 mov     r0, #0
20 isb
21 mcr     p15, 0, r0, c2, c0, 2    // Modify the TTBCR to deactivate SKEE
22 isb
23
24 mcr     p15, 0, r0, c8, c7, 0    // TLB invalidate
25 isb
26
27 pop     {r0}                     // Reload status register value
28 msr     cpsr, r0                 // Restore the original status register
29
30 bl      kernel_entry             // Jump back to the kernel
31 /* End of the SKEE Exit Gate */
```

Atomic Switch Gate

- Control flow change
  - Branching
  - Exceptions
  - Interrupts
- Threat
  - Skip interrupt disable
  - Use TLB cached code
- Solution
  - Instrument the interrupt handler
    - *Check TTBCR.N*
    - *Crash on non-zero (SKEE is exposed)*

```
/* Start of the SKEE Entry Gate */
mrs     r0, cpsr                // Read the status register
push    {r0}                    // Save the status register value
orr     r0, r0, #0x1c0          // Set the mask interrupts bits
msr     cpsr, r0                // load the modified value

mov     r0, #0x11
isb                             // Syncronization barrier
mcr     p15, 0, r0, c2, c0, 2   // Modify the TTBCR to activate SKEE
isb

mcr     p15, 0, r0, c8, c7, 0   // TLB invalidate
isb

bl      skee_entry              // Jump to SKEE entry point
/* End of the SKEE Entry Gate */

/* Start of the SKEE Exit Gate */
mov     r0, #0
isb
mcr     p15, 0, r0, c2, c0, 2   // Modify the TTBCR to deactivate SKEE
isb

mcr     p15, 0, r0, c8, c7, 0   // TLB invalidate
isb

pop     {r0}                    // Reload status register value
msr     cpsr, r0                // Restore the original status register

bl      kernel_entry            // Jump back to the kernel
/* End of the SKEE Exit Gate */
```

Deterministic and Exclusive Switch Gate

- Deterministic
  - No reliance on input
- Exclusive
  - No TTBR0, TTBR1 or TTBCR instructions exist in the kernel code

```
/* Start of the SKEE Entry Gate */
mrs     r0, cpsr                    // Read the status register
push    {r0}                        // Save the status register value
orr     r0, r0, #0x1c0              // Set the mask interrupts bits
msr     cpsr, r0                    // load the modified value

mov     r0, #0x11
isb                                 // Syncronization barrier
mcr     p15, 0, r0, c2, c0, 2       // Modify the TTBCR to activate SKEE
isb

mcr     p15, 0, r0, c8, c7, 0       // TLB invalidate
isb

bl      skee_entry                  // Jump to SKEE entry point
/* End of the SKEE Entry Gate */

/* Start of the SKEE Exit Gate */
mov     r0, #0
isb
mcr     p15, 0, r0, c2, c0, 2       // Modify the TTBCR to deactivate SKEE
isb

mcr     p15, 0, r0, c8, c7, 0       // TLB invalidate
isb

pop     {r0}                        // Reload status register value
msr     cpsr, r0                    // Restore the original status register

bl      kernel_entry                // Jump back to the kernel
/* End of the SKEE Exit Gate */
```

Secure Switching on 64-bit ARMv8

- Memory management in 64-bit ARMv8
  - Different virtual memory subranges for TTBR0 and TTBR1
    - *TTBR1: High address range; Typically used by kernel*
    - *TTBR0: Low address range; Typically used by user space*
- Challenge
  - TTBR0 and TTBR1 map mutually exclusive memory ranges
  - Cannot dedicate either registers to SKEE
- Solution
  - SKEE shares TTBR1 with the kernel
  - Entry gate uses a special encoding
    - *the Zero register (XZR)*
    - *Guarantee deterministic change of TTBR1*

Samsung Knox

Secure Switching on 64-bit ARMv8 (cont.)

- The presence of physical address 0x0
  - Provided by the hardware as a real physical address
    - *Don’t need hypervisor support*
  - Provided by the virtualization layer as an intermediate physical address (IPA)
    - *Need hypervisor to remap IPA0 x0 to SKEE*
    - *Don’t require any “runtime” hypervisor involvements*

ARMv8 Entry Gate

- Lines 2-4
  - Disable interrupts
- Lines 6-10
  - Save exiting TTBR1
  - Load TTBR1 using XZR
- Lines 12 and 13
  - Invalidate the TLB
- Lines 15 and 16
  - Jump to SKEE

```
1  /* Start of the SKEE Entry Gate */
2  mrs    x0, DAIF                // Read interrupt mask bits
3  str    x0, [sp, #-8]!          // Save interrupt mask bits
4  msr    DAIFset, 0x3            // Mask all interrupts
5
6  mrs    x0, ttbr1_el1           // Read existing TTBR1 value
7  str    x0, [sp, #-8]!          // Save existing TTBR1 value
8
9  msr ttbr1_el1, xzr             // Load the value Zero to TTBR1
10 isb
11
12 tlbi vmalle1                   // Invalidate the TLB
13 isb
14
15 adr x0, skee_entry             // Jump to SKEE entry point
16 br x0
17 /* End of the SKEE Entry Gate */
```

Samsung Knox

ARMv8 Entry Gate

- Atomic
  - Kernel cannot skip interrupt disable step
  - Jump to SKEE uses absolute address
- Deterministic
- Exclusive

```
/* Start of the SKEE Entry Gate */
mrs     x0, DAIF                // Read interrupt mask bits
str     x0, [sp, #-8]!          // Save interrupt mask bits
msr     DAIFset, 0x3            // Mask all interrupts

mrs     x0, ttbr1_el1           // Read existing TTBR1 value
str     x0, [sp, #-8]!          // Save existing TTBR1 value

msr ttbr1_el1, xzr              // Load the value Zero to TTBR1
isb

tlbi vmalle1                    // Invalidate the TLB
isb

adr x0, skee_entry              // Jump to SKEE entry point
br x0
/* End of the SKEE Entry Gate */
```

ARMv8 Exit Gate

Samsung Knox

- Lines 2-5
  - Memory padding
  - Pushing line 11 to the isolated page boundary
- Line 7
  - Mask interrupts
- Lines 9-11
  - Reload kernel's TTBR1
- Lines 15-17
  - Invalidate the TLB
- Lines 20-23
  - Restore interrupts and return to kernel

```
1  /* Start of the SKEE Exit Gate */
2  nop                          //no operation
3  nop                          // Fill the page with no operations to
4  nop                          // align the last instruction with the
5  nop                          // bottom of the isolated page boundry
6
7  msr    DAIFset, 0x3          // Mask all interrupts
8
9  ldr x0, [sp, #8]!            // Reload kernel TTBR1 value
10 dsb sy
11 msr ttbr1 el1, x0            // Restore TTBR1 to kernel value
12
13 /*-----------Isolated Page Boundry--------------*/
14
15 isb
16 tlbi vmalle1                 // Invalidate the TLB
17 isb
18
19
20 ldr x0, [sp, #8]!            // Reload interrupts mask bits
21 msr DAIF, x0                 // Restore interrupts mask bits register
22
23 ret
24 /* End of the SKEE Exit Gate */
```

ARMv8 Exit Gate

- Line 11
  - Load ttbr1 from stack
  - Can be exploited by attackers

```
1  /* Start of the SKEE Exit Gate */
2  nop                            //no operation
3  nop                            // Fill the page with no operations to
4  nop                            // align the last instruction with the
5  nop                            // bottom of the isolated page boundry
6
7  msr    DAIFset, 0x3            // Mask all interrupts
8
9  ldr x0, [sp, #8]!              // Reload kernel TTBR1 value
10 dsb sy
11 msr ttbr1 el1, x0              // Restore TTBR1 to kernel value
12
13 /*------------Isolated Page Boundry--------------*/
14
15 isb
16 tlbi vmalle1                   // Invalidate the TLB
17 isb
18
19
20 ldr x0, [sp, #8]!              // Reload interrupts mask bits
21 msr DAIF, x0                   // Restore interrupts mask bits register
22
23 ret
24 /* End of the SKEE Exit Gate */
```

ARMv8 Exit Gate

- Page on top
  - Only accessible to SKEE
- Page on bottom
  - Accessible to both SKEE and kernel

```
/* Start of the SKEE Exit Gate */
nop                                 //no operation
nop                                 // Fill the page with no operations to
nop                                 // align the last instruction with the
nop                                 // bottom of the isolated page boundry

msr    DAIFset, 0x3                 // Mask all interrupts

ldr x0, [sp, #8]!                   // Reload kernel TTBR1 value
dsb sy
msr ttbr1_el1, x0                   // Restore TTBR1 to kernel value

/*------------Isolated Page Boundry--------------*/

isb
tlbi vmalle1                        // Invalidate the TLB
isb


ldr x0, [sp, #8]!                   // Reload interrupts mask bits
msr DAIF, x0                        // Restore interrupts mask bits register

ret
/* End of the SKEE Exit Gate */
```

Samsung Knox

Fast Secure Switching using ASID

- Entire TLB invalidation
  - Potential performance overhead

- Using a dedicated ASID for SKEE
  - Non-global mapping of SKEE memory
  - TLB entries will only be associated with a particular ASID
  - No need to flush the TLB on every context switch

- Global mapping of the switch gate
  - Accessible to both the kernel and SKEE

Kernel Monitoring and Protection

- Control page table
  - Make sure the page table is properly set up, with W^X, DEP and PXN on user

- Replace the MMU instruction with hooks to SKEE
  - The hook will trap to SKEE
  - SKEE will check each operation

- For hosting security tools
  - Trap critical kernel events
  - Inspect kernel memory

Samsung Knox

Performance

- Secure context switching
  - No TLB invalidation → ASID is used

Processor	Average Cycles
ARMv7	868
ARMv7 (No TLB invalidation)	550
ARMv8	813
ARMv8 (No TLB invalidation)	284

Samsung Knox

Performance (cont.)

- Benchmark performance

ARMv7

Benchmark	Original	SKEE	Degradation (%)
CF-Bench	30933	29035	6.14%
Smartbench 2012	5061	5002	1.17%
Linpack	718	739	-2.93%
Quadrant	12893	12552	2.65%
Antutu v5.7	35576	34761	2.29%
Vellamo			
Browser	2465	2500	-1.42%
Metal	1077	1071	0.56%
Geekbench			
Single Core	1083	966	10.8%
Multi Core	3281	2747	16.28%

ARMv8

Benchmark	Original	SKEE	Degradation(%)
CF-Bench	75641	66741	11.77%
Smartbench 2012	14030	13377	4.65%
Linpack	1904	1874	1.58%
Quadrant	36891	35595	3.51%
Antutu v5.7	66193	67223	-1.56%
Vellamo			
Browser	3690	3141	14.88%
Metal	2650	2540	4.15%
Geekbench			
Single Core	1453	1235	15.00%
Multi Core	4585	4288	6.48%

Samsung Knox

Thank you