TRUSTSHADOW:
SECURE EXECUTION OF UNMODIFIED APPLICATIONS
WITH ARM TRUSTZONE

14.11.2018
Florian Olschewski
OUTLINE

1) Introduction
2) Trustzone
3) Threat Model
4) Overview
5) Runtime System
6) Implementation
7) Evaluation
8) Future Work
1) INTRODUCTION

- Rapid evolution of IOT-Devices
- Problem: compromised OS
  - Leak of sensitive Data
- TrustShadow(TS): shields applications from untrusted OS
- TS uses ARM-Trustzone
  - Normal world $\rightarrow$ OS
  - Secure world $\rightarrow$ TEE : critical application
- Secure world is managed by a lightweight runtime system(RTS)
  - Forwards system calls + verifies responses
2) TRUSTZONE - ARCHITECTURE

- *Partition of SoC*: hardware + software in secure and normal world
- Processor can enter normal and secure state
  - Normal state: access to resources in normal world
  - Secure state: access to all resources
- To check permissions: Non-Secure bit
- Monitor mode software to switch between the worlds
2) TRUSTZONE - ADDRESS SPACE CONTROLLER + MEMORY MANAGEMENT UNIT (MMU)

- Set-up security **access permissions** for address regions
- **Controls data transfer** between processor and Dynamic Memory Controller
  - NS-bit must equal the security setting of memory region

- MMU: Translation of virtual to physical addresses
- Memory split in 2 worlds → 2 MMU’s for **independent** memory mapping
  - *Normal world*: only access to memory in non-secure state
  - *Secure world*: access to both memory states by tuning NS-bit
3) THREAT MODEL

- Shielding applications from completely hostile OS
  - Memory disclosure
  - Code injection attacks
  - Change program behavior
  - Side channel attacks (e.g. observe page fault pattern)

- No prevention for
  - DoS-attacks: OS refuses to boot / decline time slices for a process
  - Side channel like timing and power analysis
4) OVERVIEW

- Trusted application:
  - Customized system call:
    - “zombie“ HAP: normal world, never scheduled
    - „shadow“ HAP: secure world, ran by TrustShadow
- RTS forwards exceptions to Linux
- Data structures task_shared / task_private

![Figure 1: Architecture of TrustShadow](image)
5) RTS - MEMORY MANAGEMENT

- 3 partitions of physical memory:
  - Non-secure: ZONE_NORMAL – Linux OS
  - Secure: ZONE_TZ_RT – for runtime system
  - ZONE_TZ_APP – shadow-HAP’s

- Virtual memory:
  - user/kernel memory split of secure world equals Linux
    → execution of legacy code in secure world
  - RTS maps itself to ZONT_TZ_RT
  - maps memory holding Linux in the virtual address space
    → efficiently locate shared Data from OS

Figure 2: physical + virtual memory layout
5) RTS - FORWARDING EXCEPTIONS

Exception handling of ARM-Processors:

1. Pc points exception vector table
2. Store previous cpsr to spsr
   ▪ Every processor mode has its own spsr register (banked Register)
3. Setting cpsr to indicate the target mode
   ▪ Spsr reveals information of pre-exception processor mode

Reproduction by RTS (e.g. svc):

1. Set spsr in monitor mode to represent target mode (svc)
2. Switch to target mode (svc) + set it’s spsr to represent User-Mode
3. Switch back to monitor mode
4. Issue movs instruction
   ▪ Jump to target exception handler
   ▪ Copy spsr from current mode in cpsr
   ➔ OS catches exception at correct address + in the right mode (svc, step1)
   ➔ Spsr indicates: exception comes from user mode (step 2)

current program status register (cpsr)
saved program status register (spsr)
5) RTS - HANDLING PAGE FAULT

- Exception by MMU → no page table entry for accessed memory
- OS maintains page tables
- RTS maintains own page table in secure world
  - Uses Linux page fault handler for updating
  - For TS, the Linux handler was modified: it stores the updated entry value to task_shared

**Basic Page Table update:**

- Anonymous memory
  - RTS verifies that the provided entry of task_shared is within ZONE_TZ_APP
  - RTS duplicates page table entry
5) RTS - HANDLING PAGE FAULT

Figure 3: PageTableUpdate with integrity check
5) RTS - HANDLING PAGE FAULT

1: Untrusted OS installs PT
2: Runtime system installs PT
3: Runtime system decrypts the \( N \)-page to the \( S \)-page
4: Hash validation
5: HAP updates the \( S \)-page
6: On unmapping, runtime system updates the hash value
7: Runtime system encrypts the \( S \)-page back to the \( N \)-page

Figure 4: PageTableUpdate for Protected Files
5) RTS - INTERVENING SYSTEM CALLS

- OS has no access to user data from shadow HAP
  - system call parameters are values → RTS forwards them directly
  - Pointers: RTS marshals them in a world shared buffer
    → OS gets temporary access to the system call parameters

- procedures for signal handling and coordinating Futex

- Defeating Iago Attacks
  - Manipulate return of system call → leak used for return oriented programming
  - RTS checks the results for memory overlaps
  - If one is found: → HAP is killed
5) RTS - INTERNAL EXCEPTION HANDLING

Floating Point Computation
- Multiple processes enter VFP – Linux maintains VFP context for each process
  - Leaks User Data
- RTS duplicates code handling VFP

Random Number Generator
- Random numbers very important for cryptographic operations
- OS should not know key materials
- RTS utilizes on-board hardware RNG4
5) RTS - MANIFEST DESIGN

- Each HAP is bundled with a manifest
  - Provides meta data for security features
  - Per application secret key
  - Integrity metadata (vaddr, hash)
  - List of filenames that should be protected

- Manifest is stored on persistent storage
  - Encrypt per-application key by per-device public key
  - Append digital signature
6) IMPLEMENTATION

Normal World – changes on linux

- Added parameter to indicate ZONE_TZ_APP -> pages for HAPs come from this region
- Added a flag -> OS can distinguish HAPs
- New System call to start HAPs
- Changed ret_to_user -> OS pass execution back to shadow instead of zombie
- Hooked page fault handler
- Modifeid code handling signals

→ 300 LOC
6) IMPLEMENTATION

**Secure World**

→ 4.5 k LOC in C + 0.8k LOC of assembly

- Applicable for manual review or formal verification
- In addition: secure boot mechanism
7) EVALUATION

Microbenchmarks

- Overhead imposed by system calls
- Ran each benchmark with 1,000 iterations -> took average

<table>
<thead>
<tr>
<th>Test case</th>
<th>Latency (μs)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linux</td>
<td>Trust Shadow</td>
</tr>
<tr>
<td>null syscall</td>
<td>0.7989</td>
<td>1.6048</td>
</tr>
<tr>
<td>open/close</td>
<td>29.2168</td>
<td>40.7886</td>
</tr>
<tr>
<td>mmap (64m)</td>
<td>559.0000</td>
<td>784.0000</td>
</tr>
<tr>
<td>pagefault</td>
<td>4.7989</td>
<td>7.9764</td>
</tr>
<tr>
<td>signal handler</td>
<td>1.6257</td>
<td>3.8294</td>
</tr>
<tr>
<td>install</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signal handler</td>
<td>51.6111</td>
<td>57.0349</td>
</tr>
<tr>
<td>delivery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fork+exit</td>
<td>987.0000</td>
<td>2328.6000</td>
</tr>
<tr>
<td>fork+exec</td>
<td>1060.3333</td>
<td>2509.0000</td>
</tr>
<tr>
<td>select (200fd)</td>
<td>15.0707</td>
<td>18.8649</td>
</tr>
<tr>
<td>ctxsw 2p/0k</td>
<td>30.3700</td>
<td>32.7100</td>
</tr>
</tbody>
</table>
7) EVALUATION

File Operations
- 128 files, each 8Mb
- Sequential + random write
- Caching disabled
- File protection on → high overhead
- Encryption + hashing
- Solution: better cryptographic engine
7) EVALUATION

Embedded Web Server

- Impact on real world application
- Respond with HTML files in different size

- **Small files:** reduce throughput ~ 6-10%
- **Big files:** only ~2% from 256 kb
- **HTTPS:** TS-overhead overwhelmed by intensive cryptographic operations
- **Latency:** almost no overhead
8) FUTURE WORK

Remaining Attack Surface

- DoS-attacks: process scheduling / start application in normal world

- Manipulation of Manifest
  - Roll-back attack possible
  - Future: version number in manifest

- Side channel attacks still are possible
  - It is possible to adopt known techniques for prevention
  - E.g. cryptographic libraries like OpenSSL

- Physical attacks
  - Solution: store sensitive data on SoC components: harder to compromise
  - Future: extend iRAM
THANK YOU
## Application level view

### Privileged modes

<table>
<thead>
<tr>
<th>User mode</th>
<th>System mode</th>
<th>Hyp mode</th>
<th>Supervisor mode</th>
<th>Monitor mode</th>
<th>Abort mode</th>
<th>Undefined mode</th>
<th>IRQ mode</th>
<th>FIQ mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>R1_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>R2_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>R3_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>R4_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>R5_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>R6_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>R7_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>R8_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R8_fiq</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>R9_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R9_fiq</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>R10_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R10_fiq</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>R11_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R11_fiq</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>R12_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R12_fiq</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>SP_usr</td>
<td>SP_hyp †</td>
<td>SP_svc</td>
<td>SP_mon ‡</td>
<td>SP_abt</td>
<td>SP_und</td>
<td>SP_irq</td>
<td>SP_fiq</td>
</tr>
<tr>
<td>LR</td>
<td>LR_usr</td>
<td></td>
<td>LR_svc</td>
<td>LR_mon ‡</td>
<td>LR_abt</td>
<td>LR_und</td>
<td>LR_irq</td>
<td>LR_fiq</td>
</tr>
<tr>
<td>PC</td>
<td>PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APSR</td>
<td></td>
<td></td>
<td>SPSR_hyp †</td>
<td>SPSR_svc</td>
<td>SPSR_mon ‡</td>
<td>SPSR_abt</td>
<td>SPSR_und</td>
<td>SPSR_irq</td>
</tr>
<tr>
<td>CPSR</td>
<td></td>
<td></td>
<td></td>
<td>ELR_hyp ‡</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Hyp mode and the associated banked registers are implemented only as part of the Virtualization Extensions
‡ Monitor mode and the associated banked registers are implemented only as part of the Security Extensions
SECURE BOOT

Proprietary Boot ROM

Verify
FAIL → Reset
PASS

Boot Runtime System

Initialization

Decryption
Device Key

Protected device private key and Manifests

Install Manifests

Uboot

Boot Linux Kernel

Resume Linux Kernel