Introduction to Secure Boot

Jonathan Ben-Avraham

TkOS

June 5, 2019

Revision 0.2

Lecture Overview

Tk Open Systems Ltd

- Embedded Linux Consulting Company founded in 1996
- Offices in Jerusalem (2), Yokneam (1)
- 21 Employees, 3 contractors, looking to hire new talent
- Hundreds of Linux kernel patches contributed, U-Boot, Buildroot, linux-mtd
- Board Support Packages and Board Bring-up
- ARM, Intel, Atmel and Xilinx MPSoC
- Linux, FreeRTOS, SysGo PikeOS and Greenhills Integrity
- DevOps, Linux sysadmin
- yba@tkos.co.il

Topics Covered

- Attack types and defenses
- Description of the **bootchain** common to most systems
- Significance of an immutable **root of trust** for secure boot
- Description of **basic secure boot**
- Relation of TPM to secure boot
- Isolation technologies for protecting the boot environment
- Isolation technologies in the run-time environment

Embedded Device Attack Surfaces

- Physical attack surface
  - Embedded devices - consumer, medical, industrial
  - Communication infrastructure - routers, deployed equipment
  - Weapons, drones
- Logical attack surface
  - Servers, routers, other physically secure infrastructure
  - Telephones, laptops
- Human attack surface
  - Security policy, separation of duties, training, controls

Attack Types

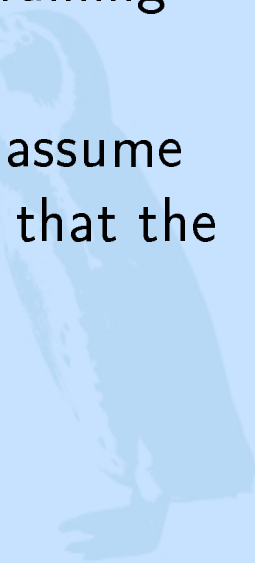
- Invasive, SoC level
  - Electron microscope plus Chipjuice, similar tools
- Non-invasive, board level
  - JTAG
  - Logic analyzer
  - Side-channel analysis
- Network probe or exploit
- Human factor attack
  - spies, corruption, information leakage, self-attack

Security Limitations

- Any circuit can be reverse engineered and copied
- Any persistent storage can be read unless erased
- But data can be hidden or defensively destroyed

Basic Defenses

- Hardware level - tamper fuses, anti-tamper memory components, self-destructive components
- Logic level - unique keys per individual device to protect against class exploits
- Human level - security policies, separation of duties, training and control
- For the remainder of this talk about secure boot, We assume that all of these defenses have been implemented and that the attack surface is logical at the board level



Lecture Overview

Overview of Basic Secure Boot

- Bootchain and boot environment
- Authentication with digital signatures
- Optional encrypting executables
- Hardware and software isolation of the boot context

Boot Chain

- The boot chain is the set of instructions that device executes between power-on and application initialization.

Typical Embedded Bootchain

- Microcode ROM bootloader, First stage bootloader, Primary bootloader
- SPL, (“Second Program Loader”) MLO (“memory locator”) in TI terminology
- Main bootloader - aboot, x-loader, U-Boot, GRUB
- OS kernel or hypervisor
- Initramfs (“initial RAM filesystem”)
- Root filesystem
- Device application code

ROM Bootloader

- Installed on the SoC by the vendor but sometimes can be updated
- Size in tens of KB
- No user interface - does not assume a serial port is available
- Runs from internal SoC NVRAM
- Some implementations can be fused OTP (optionally) to make immutable

ROM Bootloader Functions

- Initializes clocks, power supplies and media peripherals
- Selects boot media (optional)
- Reads SPL file from boot media and loads to SRAM
- Verifies SPL digital signature (optional)
- Decrypts SPL (optional)
- Starts SPL execution

Second Program Loader (SPL)

- Small enough (typically less than 64KB) to run in SRAM
- Resides on internal or external media
- Runs from persistent media or SRAM
- Initializes just enough hardware (SDRAM at least) to boot the next stage
- Can boot an OS kernel directly (called “Falcon mode” in U-Boot)
- Highly SoC-specific
- Might not even initialize a serial port for console output but usually does

Main Bootloader

- Provides extra boot functionality such as network boot
- Typically 200-700 KB size
- Runs from SDRAM
- Supports a wide range of peripheral devices
- Can select kernel, device tree blobs (DTB's), initramfs (initial RAM filesystem) files
- Can have a command line interface
- Can be scriptable - can be used to automate software installation
- Can often use SoC crypto engine to verify digital signatures
- Not always required or desired, i.e. fixed boot configuration and minimal boot time required

OS Kernel

- Re-allocates SDRAM, replacing memory-resident main bootloader code
- Can mount an initramfs or mount a root filesystem directly
- Can have encrypted filesystem drivers to mount from encrypted media

Initramfs (Initial RAM Filesystem)

- Typically a small filesystem of 1-5 MB containing a `libc` of some type and a shell (command-line) interpreter
- Can load additional device driver files (kernel modules)
- Can check read-only filesystem signatures or hashes
- Can install OTA OS updates
- Can mount encrypted root filesystem and transfer control (`switch_root`)
- Not always required or desired - increases boot time

Basic Secure Boot Chain of Trust

- Digital certificates, decryption keys and hashes stored in the SoC's OTP flash and fused (immutable root of trust)
- Boot ROM - fused read-only (immutable) by developer in production mode
- SPL, main bootloader - encrypted and signed, verified by boot ROM
- Kernel and initramfs - encrypted and signed, verified by main bootloader
- Initramfs mounts encrypted root filesystem from media using encryption key in OTP NVRAM
- Root filesystem script checks digital signature of application file, then executes

Comparison with PC Bootchain

- BIOS or UEFI is provided by board vendor
- Cannot easily be modified or replaced by developer
- Source code is not usually provided
- BIOS and UEFI provide equivalent of boot ROM, SPL and some features of main bootloader
- BIOS's do not provide encryption services or immutable data storage
- UEFI provides encryption services, but...
  - Requires vendor-provided hardware immutability such as "Intel Boot Guard".

Basic Secure Bootchain Vulnerabilities

- Basic secure bootchain is actually pretty good, but:
  - No hardware enforced media isolation
  - Application programs have access to media containing SPL, main bootloader and kernel, allowing copying for later analysis
- Direct access to TPM or SoC crypto engine
  - Application programs have direct access to SoC crypto engine or TPM allowing exploitation of vulnerabilities
- Later bootchain crypto keys must be stored in later images
  - A root filesystem dm_crypt key must be sequestered in the bootloader, or initramfs

Lecture Overview

NXP HABv4 Overview

- Provides infrastructure for basic secure boot on i.MX50, i.MX53, i.MX6, i.MX7 and i.MX8
- Uses RSA public key cryptography
- Implemented in silicon in the SoC crypto engine IP block
- Extends the SoC boot ROM functionality
- Provides digital signature checking and decryption functions for a boot image
- Adds from 15 to 25 ms per MB of boot image to boot time
- Extends the U-Boot SPL and main bootloader to provide digital signature checking and decryption
- Well documented, provides tools for image signing

Typical HABv4 Use Case

- i.MX6 ROM boot authenticates and decrypts combined boot image on eMMC boot partition
- Encrypted boot image contains SPL, U-Boot and Linux kernel and initramfs
- Initramfs can mount a `dm_crypt` root filesystem using a cleartext encryption key
- The initramfs `switch_root`s to the root filesystem
- The kernel frees the initramfs memory, destroying the cleartext dm_crypt key

Lecture Overview

TPM Overview

- ISO/IEC 11889
- A TPM is similar to a USB or parallel-port software license dongle
- Can tie the software on a device to a unique physical encryption key
- Can contain hashes of files or filesystems to verify integrity
- Together with external persistent storage provides immutable root of trust
- Contains an unknowable RSA private key (the SRK) for sub-key encryption
- Facilitates per-unit media encryption keys

TPM Functions

- Provides a fixed set of standardized functions, both at boot time and afterwards:
  - Random number generation
  - Automatic RSA key generation
  - SHA-1 and SHA-256 hash generation
  - Hash-based software metrics to detect modified software and to enforce DRM
  - Asymmetric (and symmetric in TPM 2.0) encryption functions
- External private key encryption using TPM device internal key (Storage Root Key - SRK)
- Often implemented as a surface mount component with SPI interface

TPM Implementations

- TPM can be implemented as:
  - Discrete chip - often with tamper resistance
  - Integrated TPM on SoC
  - Firmware TPM - must be protected by additional means such as a Trusted Execution Environment
  - Software - A software TPM can be protected by running in a virtual machine

TPM Summary

- Wide range of vendor discretion in hardware and software functions an implementation methods.
- There is nothing magic about a TPM hardware implementation
- Its presence does not in itself provide security
- Data encrypted with an SRK is unrecoverable if the TPM is reset or broken

Comparison of HABv4 to TPM

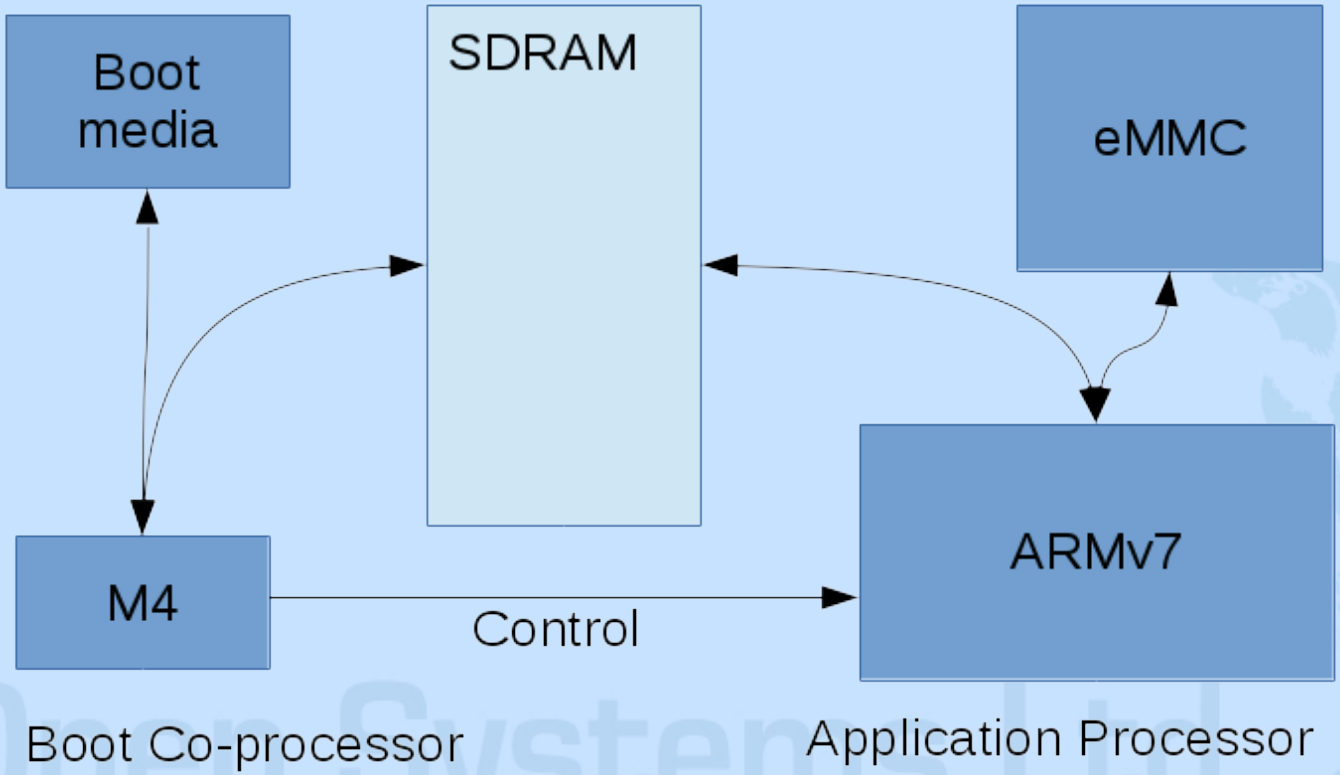
- NXP i.MX crypto engine can store decryption keys (the root of trust) that the developer generates
  - But it cannot generate keys by itself
- A TPM generates its SRK by itself, internally
  - But the SRK private key cannot be exported from the TPM, only the public key is exported
  - TPM does not store certificates or the root of trust, external persustent storage is required
- HABv4 basic secure boot can use the same private key for many device units
  - A TPM-based secure boot uses keys that are unique to each device
- NXP i.MX crypto engine OTP is not designed to store software metric hashes

Lecture Overview

Co-Processor Isolation of Boot Context

- The boot context is the SPL and main bootloader code, and optionally the OS kernel and initramfs
  - Isolation achieved by adding a physical co-processor for boot loading
  - The boot co-processor shares the same SDRAM as the main processor
  - The boot media is accessible to the co-processor but not to the main processor
  - TPM accessible only to co-processor
  - Examples: AMD Platform Security Processor (PSP), Qualcomm 8084
- Advantages: Complete isolation of boot media from application environment
- Disadvantages: A lot of extra silicon

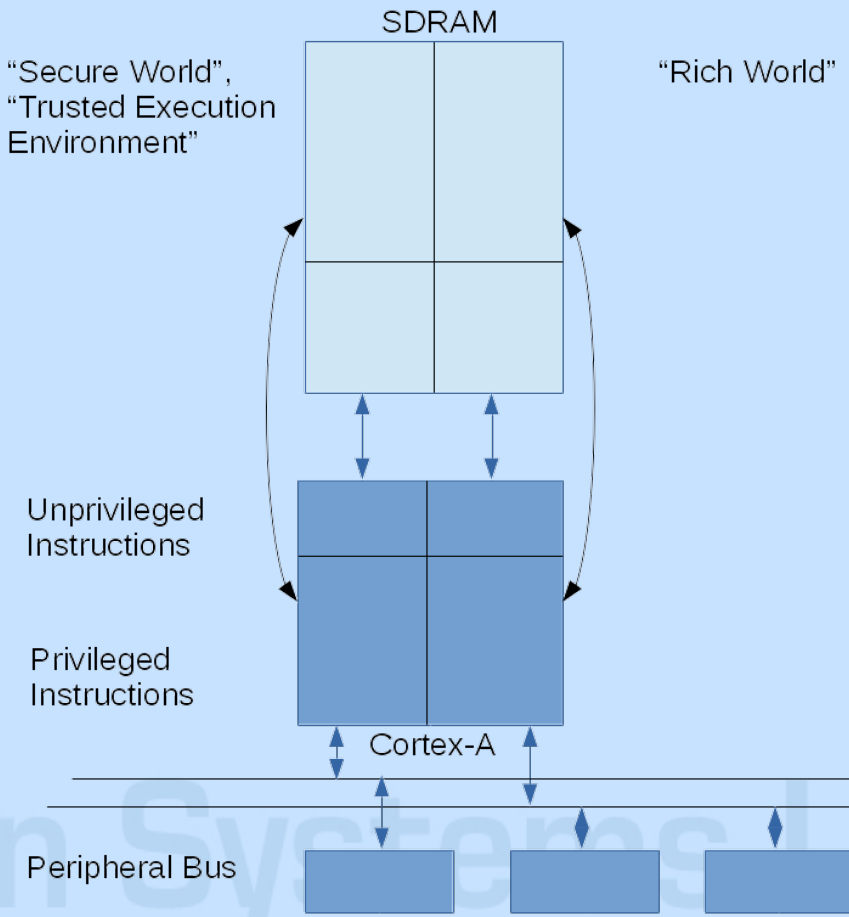
Co-processor Boot



ARM TrustZone Design

- A co-processor implemented as a processor mode with banked CPU registers
- Like ARM FIQ mode, but with extensions for bus and memory access
- Like Hyperthreading or hardware virtualization in Intel x86 architecture
- Separated bus access mode
- Separated SDRAM access mode - “33rd address bit”
- Available on Cortex-A architectures
- Enables creation of a “Trusted Execution Environment” (TEE)
- Implementation is highly SoC designer dependent
- Is not itself virtualizable with ARM hardware virtualization - only one per core

ARM TrustZone Design



ARM TrustZone Advantages

- Low silicon overhead compared with co-processor
- Powerful - can run a general operating system such as Linux - M4 cannot
- Flexible functionality - can be extended after deployment with “trustlets”
- Can have access to all buses and devices - unlike boot co-processor
- Can monitor the “Rich World” OS and media continuously

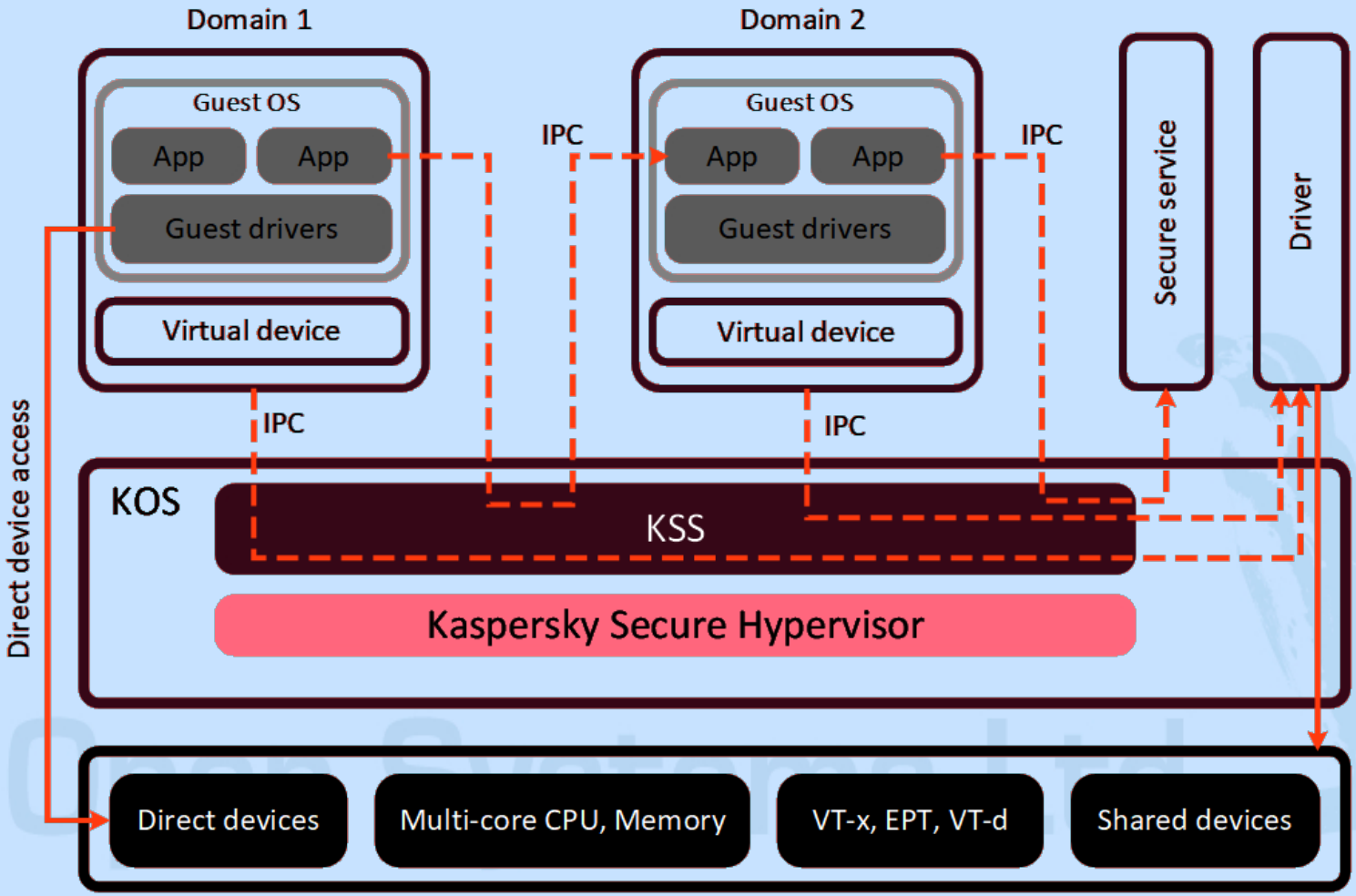
Secure Boot using ARM TrustZone

- Depends on SoC implementation of the TrustZone
- Boot ROM starts the CPU in TrustZone mode and boots the SPL in SRAM
- SPL copies the main bootloader, or a trusted OS kernel and filesystem into SDRAM
- The trusted OS copies the application kernel into SDRAM, sets the program counter and moves the CPU to non-trusted mode
- There are now two distinct OS kernels in SDRAM
  - ...and **two distinct OS's running on the same CPU core**

Boot Isolation using a Software Hypervisor

- The hypervisor code is **trusted** but constant (not flexible as in TEE)
- The hypervisor runs in privileged mode
- The application domains run in unprivileged mode
- To run in a hypervisor domain, standard OS's such as Linux must built to support paravirtualization
  - Major Linux distros provide dual mode kernels
- A Linux kernel running as a hypervisor domain (guest) runs in unprivileged mode
- Communication between application domains via hypervisor message queue only
- Hypervisor configuration determines domain peripheral access

Hypervisor Isolation



Comparison of ARM TrustZone with Hypervisor Isolation

- Hypervisors provide **software isolation**, TrustZone is **hardware isolation**
- Hypervisor is **CPU-only**, does not segregate bus access, so is open to **DMA attacks**
- Hypervisors can generally be ported to many different non-ARM architectures
- A hypervisor can be run as an ARM Rich World OS to provide a hybrid security model

Lecture Overview

Intel Software Guard Extensions (SGX)

- CPU Memory Encryption engine (MEE) for on-the-fly encryption and decryption of code and data memory "enclaves"
- Available in both privileged and non-privileged modes
- Prevents even privileged code outside the enclave from reading from the enclave
- Writes to persistent storage from enclaves are automatically encrypted
- Designed for user-space application self-defense against privileged attacks
- Code run in enclaves must be signed, to prevent running malware in an enclave
- Allows running an application securely even on a hacked OS
- Implementation of crypto functions on the CPU chip prevents inter-chip bus snoop attacks
- Requires a very large amount of silicon

SGX Use Case

- Proprietary, signed applications are distributed in encrypted form
- A wrapper program creates an enclave for the application code
- The wrapper program initiates decryption the proprietary application inside the enclave
- The application runs in the enclave - neither the OS not the wrapper can see it's code or data
- SGX is not supported in Linux yet, although a patch set available