A practical approach to securing embedded & IoT platforms

Marc Canel
Security Systems
Agenda

- What can we learn from mobile security and apply to IoT?
- Building on proven security principles & Secure Partitioning Manager
- What can be done to make the IoT developer’s job easier?
- Summary
October 21 ‘16 – IoT Botnet attack

- “….100,000 malicious endpoints. We are able to confirm that a significant volume of attack traffic originated from Mirai-based botnets.” (Dyn, Attack analysis)
How can we make IoT security as robust as mobile?

How do we design-in robust end-to-end security?
How much security to fit your needs?

Cost/effort to attack

Security Subsystem

Secure Element

Hardware isolated TEE/SPM*

TLS/SSL

Value to attacker

SW & HW Attacks
  • Physical access to device
    – JTAG, Bus, IO Pins,
  • Time, money & equipment.

Software Attacks & lightweight hardware attacks
  • Buffer overflows
  • Interrupts
  • Malware

Communication Attacks
  • Man In The Middle
  • Weak RNG
  • Code vulnerabilities

Cost/effort to secure

*Trusted Execution Environment
/ Secure Partitioning Manager
What can we learn from mobile & apply to IoT?

- Lifecycle Security
- Communications Security
- Device Security
Keep it agile - enable OTA updates

- All code contains bugs and vulnerabilities
  - Vulnerabilities need to be patched before they are exploited

- IoT needs to take the Mobile platform approach & push out regular updates that fix security vulnerabilities as well as improve functionality

- For microcontroller based systems standards such as LWM2M can provide a common protocol

- For management of security domains and trusted code, standards such as GlobalPlatform TMF and IETF's Open Trust Protocol (OTrP) are emerging
Security is best with layers of hardware isolation

Normal World
IoT developer writes Apps
On top of his/her chosen RTOS

Secure World
= Trusted code (mostly libs)
Provided by MDK, IoT platform or ISV
+ Trusted hardware

Security subsystem
Highly evaluated code developed by security specialists & built in by silicon vendor

TrustZone

Control interface

Security Subsystem

Security is best with layers of hardware isolation

Normal world
- Non secure app
- Non secure RTOS

Secure world (TCB)
- Secure app/libs
- Secure RTOS
Platform security = TEE + Security subsystem

- Applications
  - Platform security
    - Code encryption
    - Lifecycle management
    - Debug authentication
    - SW updates validation
      - Data protection (off-line, runtime)
      - Rollback protection
    - Loaded SW validation
      - Cryptography
    - RNG
      - Persistent trusted storage
    - Trusted Execution Environment isolation

Security subsystem provides HW based security “toolbox”

ARM TrustZone provides isolation between normal world code and the Trusted Code Base
Building on proven security practices

- Mobile security architecture has evolved to provide robust protection against common attacks
  - Uses principles of hardware ”Compartmentalization” and “Least privilege”
  - Use a hardware root of trust & trusted boot
  - Ensure system is updatable

- IoT architecture can re-use proven security practices

- However, ease of use (of security functions) is much more important for IoT
ARM TrustZone based TEE on mobile systems

A reminder of the architecture
MCU architecture becoming similar to mobile

Normal World Code  Trusted Software

Unprivileged
- Apps/User
- Comms Stack

Privileged
- RTOS
- Device Drivers

Hardware Interfaces

ARM Cortex-M v8-M  Microcontroller  TRNG  Unique ID  Subsystem

Initial ROT & Security subsystem

Physical IP

TLS/Crypto Libs

CMSIS API

TrustZone based Partitioning Manager

Initial ROT & Security subsystem

TrustZone enabled MCU
IoT security needs to be easy / designed in

**Situation**
- Most IoT developers are not security experts
- Little to no knowledge of hardware
- Prior experience in mobile app development
- Time to market & functionality beat security

**Strategy**
- Ease of use requirements on tools & IoT platform providers
- Hide complexity of hardware based security
- Provide built-in security functions
- Use standard methods and building blocks
Dev tools need to hide security complexity

- Trusted software can be provided as libs
- IoT developer just needs to make function calls to secure world
- Development can start on models
Function calls to trusted code can be standardized

- RTOS running in Non-Secure state: RTOS functionality available to Non-Secure and Secure software
  - Full-featured RTOS for Non-Secure Application
    - Supports function calls to Secure state
    - Callback events from Secure state

- Secure state provides data and firmware protection
  - CMSIS* provides common API

*Cortex Microcontroller System Interface Standard
https://github.com/ARM-software/CMSIS_5
ARM is creating a SPM as a new Open Source Software project to provide partitions for critical code
- A framework for multiple vendors to work within
- Available as a common piece of middleware for the industry
- ARM mbed uVisor is an example implementation of SPM

SPM works with a system’s available HW capabilities to provide mutual separation of security functions
- Exposed code never sees critical keys/secrets
- Vulnerabilities on exposed side can’t affect critical side
- Critical side can reliably recover device to clean state
Consider using an IoT platform

- If you want to develop an IoT product but are not a security specialist then an IoT platform that comes with OTA updates, trusted libs & support of TrustZone hardware security features is a sensible option
Summary

- By careful selection of IoT platform and chip the developer can make use of a proven mobile security architecture

- IoT needs to be made easier for IoT developers who are not security experts
  - The dev tools must be easy to use
  - Trusted code should be supplied by the IoT platform, tools vendor or ISV

- ARM is working on open source enablement software such as Secure Partitioning Manager, ARM Trusted Firmware & standardised function calls (CMSIS) that the industry can adopt to have some common architecture and methods
Questions to ask your dev team

1. Are OTA updates enabled?
2. Do we have a threat model/risk assessment?
3. How do we protect data in flight?
4. How are keys provisioned and protected?
5. Do we use a HW-ROT?
6. Do we use any shared private keys or credentials?
7. Who is doing the security evaluation/penetration testing?
8. How do we handle reported security vulnerabilities?