Linux Secured Integrity
Verifying Boot Concept for Embedded Systems

Embedded Linux Conference 2013
Edinburgh/UK

Holger Dengler, Linutronix GmbH
holger.dengler@linutronix.de
Agenda

• Problem Statement
• „Classic“ Boot Process
• Verifying Boot Concepts
• Status Quo
• Conclusion
Embedded System Environments

Machine A

Control-Unit

Machine B

Control-Unit

Machine C

Control-Unit

Public Network (Intranet, Internet)

Control Center

Copyright 2013 by Linutronix GmbH
Embedded System Environments

Machine A
Control-Unit

Machine B
Control-Unit

Machine C
Control-Unit

Public Network
(Intranet, Internet)

Attack

Control Center

Copyright 2013 by Linutronix GmbH
Questions

• Kernel Originator?
  - Security
  - Safety (Warranty)

• Kernel Modification?
  - Detect Violation
  - → Notification, Reaction

• Dedicated Security Hardware?
Classic Boot

- CPU Initialization
- Bootloader
- Kernel Image
Classic Boot

- CPU Initialization
- Bootloader
- Kernel Image

1) load CPU Init
2) exec.
3) load Bootloader
4) exec. Linux Kernel

read-only
Verifying Boot

- CPU Initialization
- Bootloader with Certificate
- Kernel Images with Kernel Signature
- Verify Kernel Signature with Certificate

1) load
2) exec.
3) load
4) verify
5) exec.

Copyright 2013 by Linutronix GmbH
Verifying Boot

- CPU Initialization
- Bootloader with Certificate
- Kernel Images with Kernel Signature
- Verify Kernel Signature with Certificate
Extended Key Handling

1) add
2) verify

System Context

System Key-pair

Certificate

Signature (s)

Bootloader Keyring

3) add
4) verify

Signature (w)

Kernel

Working Key-pair
Kernel up & running. And now?

- Signed Modules
  - prevent Low-Level Hacking
- Linux Integrity Subsystem
  - prevent Filesystem Violation
- Linux Security Modules
  - restrict Resource Access
Status Quo

+ Phytec (phyCore, ARM Cortex-A8)
+ U-Boot 2013.07
+ some Changes (minimal DTS support)
+ RSA Key-pair (2048bit Key length)
+ some Configurations & Scripts

= Prototype with Kernel Signature Verification
Status Quo: Features

- Linux Kernel Verification during Boot
- Simple Key-chain
  - Public Key in U-Boot image
  - Signature in Kernel Image
- RSA 2048bit Key length
Status Quo: Sample Configuration

/dts-v1/;
/ {
    description = "Phytec Verified Boot";
    address-cells = <1>;
    images {
        kernel@1 {
            data = /incbin/("../..\out\arch\arm\boot\zImage");
            type = "kernel";
            arch = "arm";
            os = "linux";
            compression = "none";
            load = <0x80508000>;
            entry = <0x80508000>;
            kernel-version = <1>;
            signature@1 {
                algo = "sha1,rsa2048";
                key-name-hint = "lx-phy";
            }
        }
    }
    configurations {
        default = "conf@1";
        conf@1 {
            kernel = "kernel@1";
        }
    }
}
Conclusion

• No Dedicated Security Hardware required (Protection against Remote Attacks only)
• Adaptable
• Extensible
• Completely Reviewable
• No Secrets on System
• It's already there. Use it!
Questions?
Thanks for your attention!

Linutronix GmbH

Auf dem Berg 3, 88690 Uhldingen-Mühlhofen