Cutting malware off at the root

Preventing Linux rootkit threats through secure boot design using flash based SoC FPGAs. By **Tim Morin.**

The Linux OS is likely to become even more popular as 32bit computing becomes a commodity and projects like Yocto make it easier to create, develop and maintain Linux based systems for embedded applications.

One of the advantages of Linux is that it enables OEMs to become more like startups, where agile hardware development teams speed time to market by using an OS to abstract the underlying hardware details. However, despite its benefits, a Linux system can be vulnerable to rootkits unless its embedded processor is booted properly.

In general, rootkits try to access privileged (root) modes while hiding from system malware detection tools. The malware may also try to install itself into a persistent state by modifying the system's boot process. If successful, the infection is permanent or persists through power cycles. From there, the malware will do whatever its author wants; everything from logging key strokes to enabling unauthorised services. And if a system is infected, a complete OS reinstall may be required.

The problem starts at the embedded processor (see fig 1). The on chip ROM will, on power up, fetch the boot loader from an external nonvolatile memory. The application specific boot loader configures the processor during its startup for the specified application requirements. Clocks, caches, memory controllers and peripherals are all configured and once the processor has been initialised, the application is fetched from external non volatile memory and started.

"The only way to protect the boot process is to secure it with an entity that can be trusted to always behave in the expected manner."
Tim Morin

During the boot process, malware has the opportunity to attempt modifications to the boot loader of the embedded system. Fig 2 shows a simplified and typical block diagram of an embedded Linux system. UBOOT, the Linux kernel and the application layer are all stored in non volatile memory.

The only way to protect the boot process is to secure it with an entity that can be trusted to always behave in the expected manner. As a system element, this root of trust supports verification of system, software and data integrity and confidentiality, as well as the extension of trust to internal and external entities. It is the foundation upon which all further security layers are created and it is essential that its keys remain secret and the process it follows is immutable.

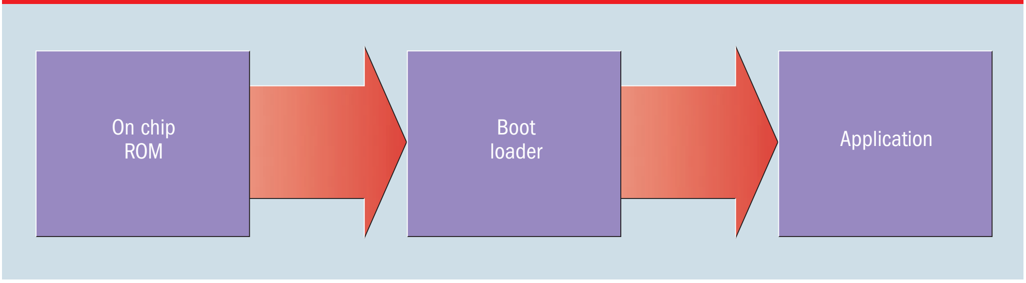
In embedded systems, the root of trust works in conjunction with other system elements to ensure the primary processor boots securely using only authorised code. Using cryptographic techniques, the trusted zone can be extended to cover all critical elements of the system or even to tie multiple trusted systems together over an intrinsically insecure network.

Looking to address these issues, Microsemi has launched a secure boot reference design that can add security to a processor that doesn't have it built in. In addition, it can prevent root kit installations if used properly.

While many newer processors come with special features to support secure boot, this is far from universal and often requires a multichip solution. A better solution is a flash based FPGA.

Microsemi's reference design is enabled by its SmartFusion2 SoC FPGAs or IGLOO2 FPGAs, which offer a number of advanced security features. These flash based solutions are inherently secure because once they are programmed, critical information never leaves the die. Amongst the security features is at-speed serial peripheral interface (SPI) flash memory emulation to enable a secure boot of an external processor. The devices also feature strong design security and differential power analysis (DPA) resistant anti tamper measures. When coupled with an embedded Linux CPU, these FPGAs can be used to prevent malware from attempting to modify the boot loader of the embedded system.

Fig 1: The embedded boot process



The reference design uses two SmartFusion2 starter kits stacked on top of each other. The SmartFusion SoC FPGA development board serves as the target processor, with the SmartFusion SoC FPGA Cortex-M3 processor acting as the target. Meanwhile, the SmartFusion2 SoC FPGA development board acts as the root of trust, implementing the secure boot functions. Signals are passed between the two boards using a GPIO connector that carries the reset signal and the SPI interface signals between the target processor and the root of trust.

Fig 3 provides an example of a flash based SoC that can be used for secure boot. The non volatile memory in the SmartFusion2 SoC stores UBOOT, while the FPGA fabric emulates an SPI flash memory at speed. The CPU is unaware that the SoC sits between it and the SPI flash. On power up, the SoC directs the SPI read request for UBOOT from the internal eNVM to the CPU. The CPU then goes through the normal UBOOT booting process and then fetches the remainder of the application image from the external SPI flash, during this latter stage the SoC is acting as a conduit to the external memory and passes the application image to the CPU.

The SoC and its internal eNVM can only be programmed with an authenticated encrypted bitstream constructed by the SoC's FPGA development environment. The format is proprietary and resistant to DPA side channel attacks, thanks to countermeasures licensed from Cryptography Research (now Rambus). In other words, the only way malware can attempt to write to flash is to have a bitstream designed for the SoC with the proper encryption and user defined keys.

To ensure secure multistage boot, it is essential the code be validated prior to delivery and execution. This ensures that no compromise has occurred that could subvert or damage the boot of each phase, and can be done using either symmetric or asymmetric key cryptographic techniques. Preferably, continual feedback to each prior stage is used to confirm that no tampering has occurred during boot load. Each phase can continue to execute if all anti-tamper monitors confirm a safe environment.

After power up, the FPGA holds the main MPU in reset until it completes its integrity self-tests. When ready, it releases the reset. The MPU can be configured to boot from the interface to the FPGA, which delivers the requested Phase 0 boot code to the MPU as it comes out of reset. Assuming the MPU does not inherently support secure boot, the challenge is to load code into the MPU with the confidence that it hasn't been tampered with.

If everything checks, the boot process continues by branching to the trusted code in the MPU's SRAM. This contains the code needed to initiate the next phase and could include an RSA or ECC public key. Once the code in the MPU SRAM is trusted, additional security measures can be employed.

Whether used as a self contained processing element or in conjunction with adjunct processors, the SoC FPGA improves security for embedded processing. While it is possible to construct an embedded processor module with specialty security devices that perform monitoring and static key storage, consolidation of system critical functions along with security features within an SoC FPGA provides much greater security, flexibility, and performance.

Author profile: Tim Morin is director of product marketing for Microsemi.

Fig 2: An embedded Linux system

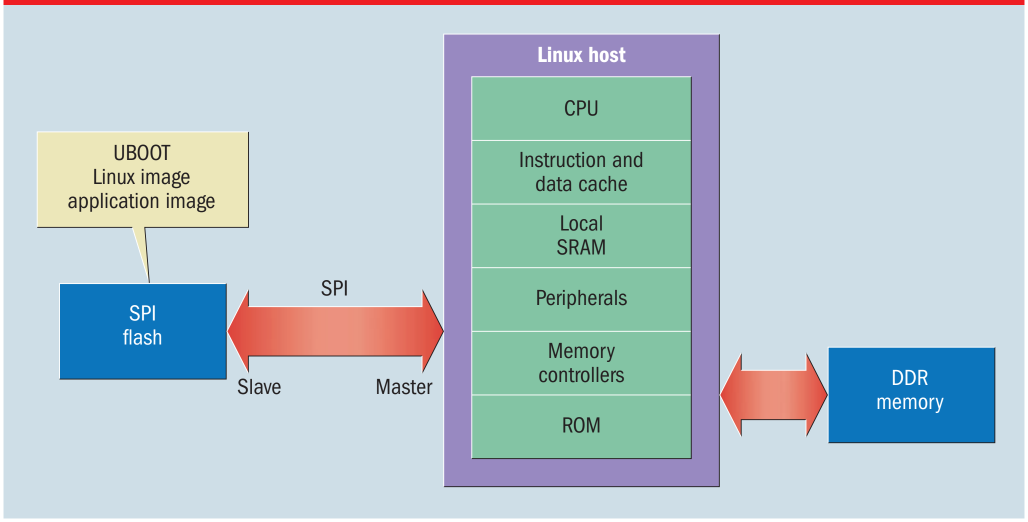
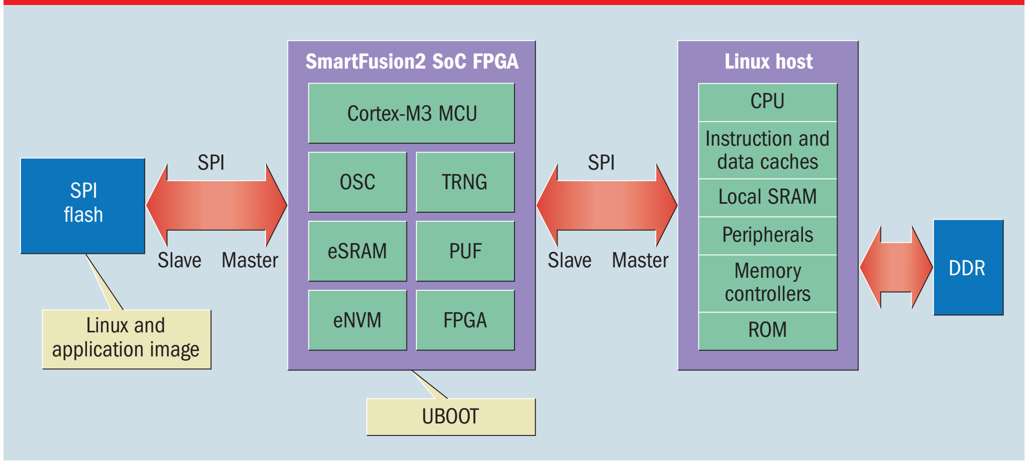


Fig 3: Using a SmartFusion2 SoC FPGA to store UBOOT


SPI flash
Linux and application image
SPI
Slave
Master
SmartFusion2 SoC FPGA
Cortex-M3 MCU
OSC
TRNG
eSRAM
PUF
eNVM
FPGA
UBOOT
SPI
Slave
Master
Linux host
CPU
Instruction and data caches
Local SRAM
Peripherals
Memory controllers
ROM
DDR