Trusted Secure Embedded Linux

From Hardware Root Of Trust To Mandatory Access Control

Hadi Nahari
MontaVista Software, Inc.
hnahari@mvista.com

Abstract

With the ever-increasing presence of Linux implementations in embedded devices (mobile handsets, set-top boxes, headless computing devices, medical equipments, etc.), there is a strong demand for defining the security requirements and augmenting, enhancing, and hardening the operating environment. Currently an estimated 70% of new semiconductor devices are Linux-enabled; such a high growth is accompanied by inevitable security risks, hence the requirement for hardware-based trusted and secure computing environment, enhanced with MAC (Mandatory Access Control) mechanisms for such devices in order to provide appropriate levels of protection. Due to stringent security requirements for resource-constrained embedded devices, establishing trust-chain on hardware root of trust, and deploying MAC mechanisms to balance performance and control are particularly challenging tasks.

This paper presents the status of MontaVista Software's efforts to implement such solutions based on ARM cores that provide separated computing environment, as well as SELinux (Security Enhanced Linux) to provide MAC for embedded devices. The focus will be on practical aspects of hardware integration as well as porting SELinux to resource-constrained devices.

1 Introduction

The defining line between embedded and non-embedded systems is becoming more and more blurred [2]; whether the system has an elaborate UI (User Interface), or if it is operating under a resource-constrained environment are not sufficiently delineating identifiers anymore.

The availability of more computing power at a low cost has also resulted in developers' and manufacturers' interest in adding more functionality to devices that were traditionally either not capable of, or not expected to have them (smart phones, hand-held computing devices, complex medical control systems, navigation gear, etc.). The difficulty in satisfying the security requirements of a software package is proportional to its design complexity. The default access control method in Linux, that is, DAC (Discretionary Access Control) is at best considered *insufficient* for any security-sensitive implementation, hence the increase in demand for MAC in recent years.

HAS (Hardware Assisted Security) has not yet been sufficiently standardized and from the industry adoption perspective, it is still considered to be in its infancy. Yet (or however), HAS is one more item in the toolbox of security architects who need to provide solutions for fundamental problems such as establishing TCB (Trusted Computing Base) using hardware-based root of trust, managing key material, and security governance of the system.

In the past years, there has been an increasing growth in adoption of Linux in various environments. This important phenomenon, which is unparalleled by other operating systems, has resulted in Linux becoming more and more the de-facto operating system for embedded devices. The recent acceleration of this adoption by said-devices is partially due to the highly modular architecture of Linux kernel, and partially due to its maturity, which results in lower COO (Cost Of Ownership.)

One must note, however, that this growth is accompanied by complex and elaborate software solutions build atop embedded Linux devices to address the ever increasing demands of the market, and hence the complex security requirements of such devices. This makes implementing a holistic security strategy for Linux more challenging, especially when one considers the highly varied selection of embedded devices adopting, or planning to adopt Linux; from smart phones, hand-held de-

vices, set top boxes, high-end televisions, medical devices, automotive control, navigation systems, assembly line control devices, missile guidance systems, to the other end of the spectrum such as CGL (Carrier Grade Linux) in the telecoms industry. Such environments each possess a very unique set of security characteristics and requirements, and therefore provide different challenges. Figure 1 shows a typical Linux-based mobile phone architecture.

In this article I will start by describing the most common security requirements in the embedded industry, and follow it by explaining the design principles and reviewing the available technologies that were initially considered viable, both for HAS and MAC, and will propose an architecture based on the selected technologies. Where applicable, I will point out whether the said-method or technology satisfies a subset of the embedded systems (mobile handsets and CGL).

The main focus of this article is to provide:

1. A high-level, architectural overview of a design that provides fundamental and necessary facilities to establish the trustworthiness of the operating system services (via connection to a hardware-based root trust).
2. A mechanism to establish *Effective Containment* (that is, a mechanism to prevent an exploited application from enabling attacks on another application possible) via the MAC offered by SELinux.

Security services provided by higher-level software constructs, such as middleware and frameworks alike are not the focus of this article and will not be discussed.

It is also noteworthy to mention that, where the required security related components exist in the underlying hardware architecture, the proposed design is viable for non-embedded systems as well.

2 Design Constraints

The following are the most commonly considered constraints when designing and developing software components for an embedded device:

1. Memory Footprint
2. Performance Trade-off

Memory footprint is important because a big majority of the embedded devices tend to have limited memory available at run-time. Any security solution for such devices must therefore have an acceptable and low memory footprint.

Performance trade-off is important because the computing power available in an embedded environment is typically low, due to hardware architecture characteristics, as well as issues pertaining power management in battery operated devices. Low power consumption is one of the fundamental reasons based on which ARM is the de-facto architecture for such devices.

3 Design Principles

1. Simplicity
2. Modularity

We have made the design as simple as possible. This is to ensure no unnecessary complexity is introduced into the system and also the overall security analysis of the system is easier to perform.

The proposed architecture is modular. This is to ensure that the design could also be implemented on hardware architectures that lack the security capabilities introduced, and also environments where the types of attacks, or the security assets of the system would not require the high degree of protection provided by this design.

4 Technology Overview

4.1 Secure Boot

Secure Boot (a.k.a. High Assurance Boot) is a technique for verifying and asserting the integrity of an executable image prior to passing the control to it. Assuming the verification mechanism is based on the digital signature of the image being verified, then the reliability of this verification is at best as good as the reliability of the protection mechanism provided in the device for the public key of the image signer.

The most important assumption here is that the code which performs the integrity verification process is itself

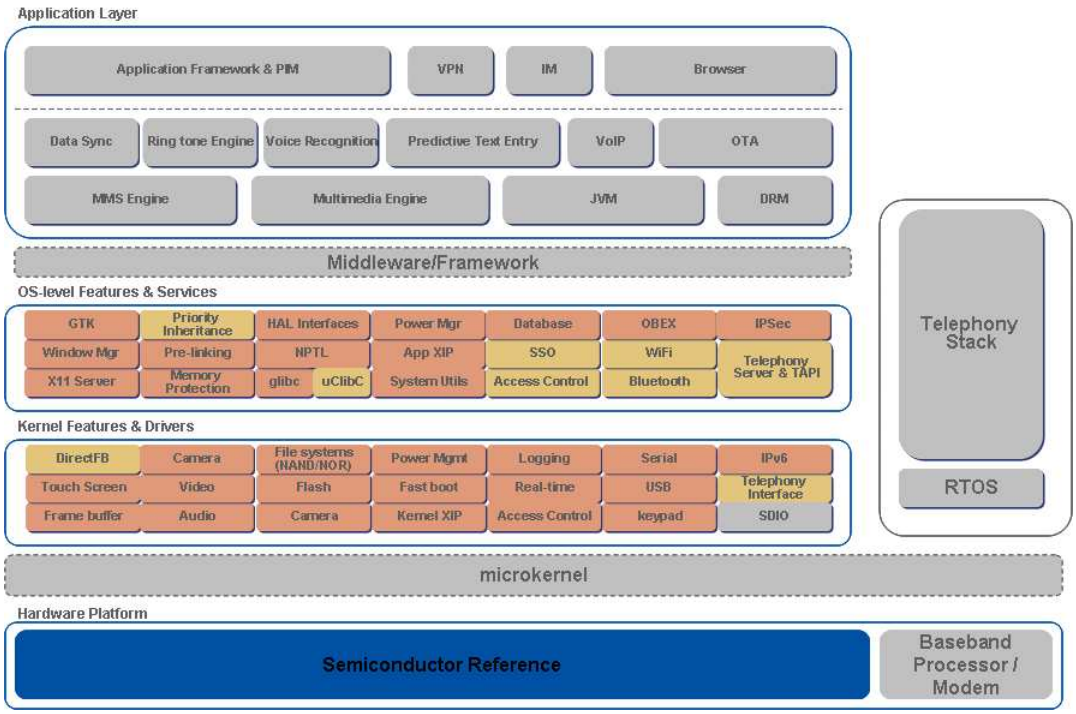

Figure 1: A Typical Linux-based Mobile Phone Architecture

trustworthy. To assert this assumption, the implementations typically put the public key material, as well as the verifier code into non-writable areas of memory, which in turn are protected via a form of hardware protection mechanism. Figure 2 shows a generic Secure Boot architecture.

This design enables the establishment of a chain of trust by ensuring that the trust, on each layer of the system, is based on, and is only based on, the trust on the layer(s) underneath it, all the way down to the hardware security component, which serves as the *Root Of Trust*. If verification fails to succeed at any given stage, the system might be put in a suspended-mode to block possible attacks.

One must note, however, that this architecture, though ensuring the integrity of the operating environment when a *hard boot* occurs, does not guarantee its integrity during the runtime; that is, in case of any *malicious* modification to the operating environment during the runtime, this architecture will not detect it until the next hard boot happens.

4.2 Effective Containment

The "Buffer Overflow" class of attacks is practically impossible to prevent in native environments with no type- or boundary-checking available at runtime. Executing native code in operating environments like Linux makes it specifically susceptible to this category of attacks. This therefore makes exploiting a buffer overflow attack of particular interest to hackers, and successfully mounting such an attack is considered a badge of honor. *Effective Containment*, in this context, is therefore referred to a class of techniques that *contain* (as opposed to *prevent*) such attacks for which there are no *practical* prevention mechanism available. This could be achieved via the use of various software and security technologies. Applying a MAC mechanism is one way to implement effective containment.

4.2.1 Embedded SELinux

One method to achieve a MAC is via implementing RBAC (Role-Based Access Control). NSA's SELinux, among other features such as MLS (Multi Level Security), provides Linux with MAC through RBAC.

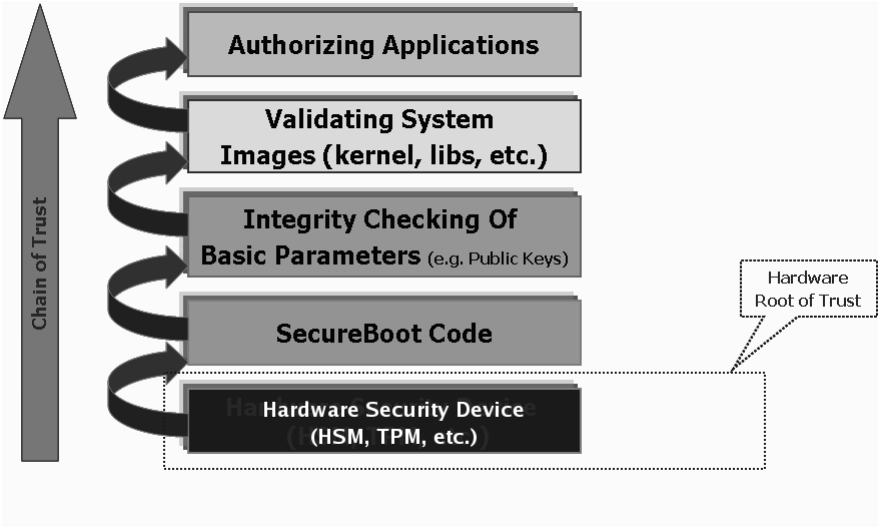

Figure 2: A Typical Secure Boot Design

SELinux was not originally designed for the ARM architecture, or for embedded devices. There have been, however, previous *and reasonably successful* attempts to port SELinux or parts of it into an embedded device and on an ARM architecture; the most notable of which being by Russell Coker [1].

By adding a MAC mechanism such as SELinux on top of Secure Boot, we will be able to address one of its fundamental shortcomings; providing a level of protection at runtime. Figure 3 shows an architecture, deploying Secure Boot and MAC mechanisms together.

In this design, not only have we accomplished augmenting the Secure Boot mechanism (by way of providing runtime containment), but have also enabled a way to expose hardware-security capabilities (e.g. TPM standard services) to the applications and processes during the system runtime.

As a side note, it is important to mention that RBAC is not the only mechanism to implement MAC. Other implementations exist which might also become suitable for embedded devices; "Tomoyo Linux" and Novell's "AppArmor" are both examples of such solutions that implement a technique called NBAC (Name-Based Access Control). LIDS (Linux Intrusion Detection System) is another example. At the time of writing this article, however, neither of these implementations seem to have been able to gain the traction in the industry, nor by manufacturers of embedded devices to be the default MAC for such devices. This state, however, may change in the future as the above-mentioned technologies mature.

4.2.2 Multi-core and Virtualization

In recent years, the industry has put more focus on adding more computing power to embedded devices, not only in adding more processing capabilities to each core, but also adding to the number of cores available in hardware architectures. One of the objectives of this expansion is have a dedicated hardware resource to each high-level task, and therefore achieving easier management of software design and implementation through hardware compartmentalization. This approach has resulted in recent growth in implementing virtualization technologies in embedded space. Various types of virtualization techniques exist (hardware- and software-based) which all provide multiple *guest domains*, each assuming total access to the underlying hardware, and conceptually having no awareness of the other guest domains. A fundamental element of any virtualization implementation is a layer called *hypervisor*, which is responsible for mediating the interactions among guest domains, and providing necessary life-cycle management for each guest domain.

As is in most cases, this technique has existed in large-scale enterprise systems prior to embedded space; however, again the implications of this technology in resource-constrained embedded devices are different

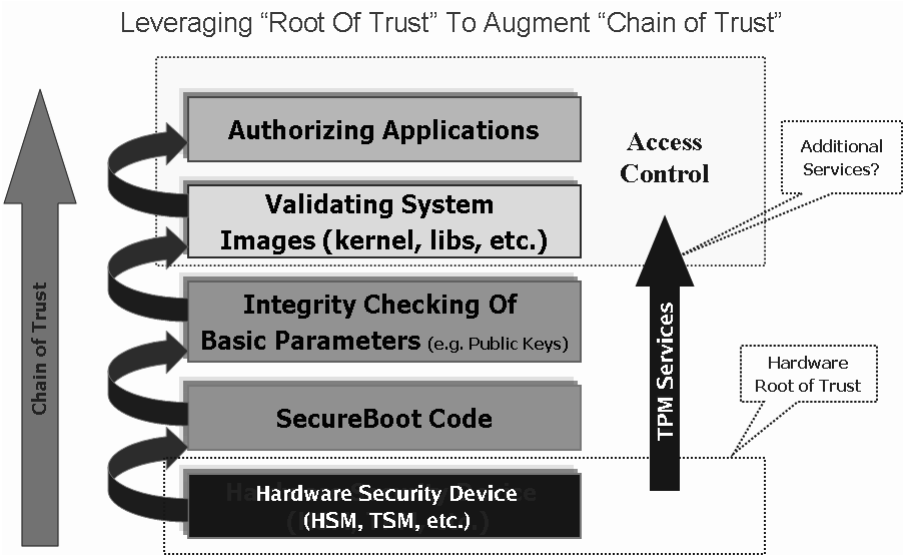

Figure 3: Augmenting Secure Boot with Access Control

than those of the enterprise systems. The use cases, however, remain similar.

Any modern design that attempts to provide a security solution for embedded space, therefore needs to assume it might be contained in a virtual, guest domain.

4.2.3 ARM TrustZone Technology

TrustZone is a security technology introduced by ARM Ltd. in its "ARM 1176" core. TrustZone is a technology to provide a hardware-based separation for execution environment, and divide it into two halves; secure and normal worlds. The security-sensitive applications are run executed in a separate memory space which is not accessible to "normal applications." TrustZone is the first ARM architecture to provide hardware-based security in its core. Although *and if done right* this could potentially provide the operating environment with an additional level of security, at its core this could be considered a hardware-based virtualization solution, and is conceptually not a new idea. Furthermore compartmentalized-security and separation of execution environments due to application's security requirements, along with "separation of concerns," have all been well-understood and known concepts in computer science and software engineering for decades [3].

The proposed design in this article considers the availability of ARM TrustZone technology on the underlying hardware architecture; however, no parts of this design *rely* on such capability. It is also important to mention that we only assume the TrustZone hardware availablity in the underlying architecture; analyzing the *proprietary* software stacks on top of TrustZone hardware that provide additional security capabilities to the applications, is outside the scope of this article.

5 Proposed Architecture

5.1 High-level Analysis

We propose the architecture shown in Figure 4 to address the security requirements discussed in this article.

This design is based on a hardware root of trust, and therefore could provide security as reliable as the method protecting this root. It implements a MAC mechanism based on embedded SELinux, and can do it in a virtualized environment. The proposed design deploys the security capabilities available in hardware (that is, TrustZone hardware) to enforce a separation mechanism among guest domains, via dedicating separate areas of memory to different processes that exist in each domain. The overall security management, such as secure IPC (Inter Process Communication) of such processes, is the responsibility of the VMM (Virtual Machine Monitor) which acts as the hypervisor in this design. This design enables a hardware-enforced separation among processes running in each guest domain,

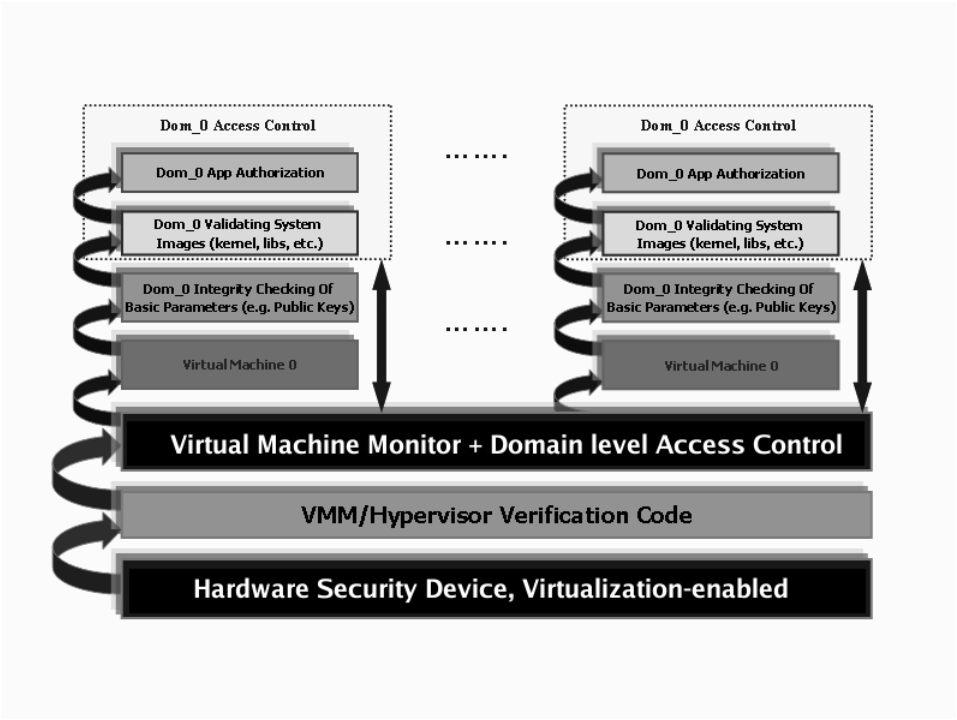

Figure 4: Implementing MAC and Virtualization, Based on Hardware Root Of Trust

with a fine-grained, mandatory access control mechanism provided by SELinux infrastructure. The initial verification of each guest domain happens prior to bringing it up. After each domain is on-line, ensuring its health from security perspective is also provided by the SELinux infrastructure.

It is presumed that due to differences in security requirements during the start-up and runtime, not all the embedded devices would require all the elements provided in this design. This, however, does not indicate a weakness in the architecture, as the main security aspects of the design could be implemented/enabled independently.

6 Conclusion

We proposed a design that deploys a hardware root of trust to provide secure execution of applications in a virtualized environment. We also augmented the design by adding a MAC mechanism to provide enhanced protection to applications and processes at runtime.

On an system which requires and implements all the capabilities provided in this architecture, a thorough analysis must be performed to ensure an appropriate security policy is in place to deploy the SELinux capabilities efficiently; an unnecessarily comprehensive and restrictive policy has a potential to hamper the overall runtime performance, and increase the memory footprint of the system.

The performance of the hypervisor is also the key in this design, as it is the layer that arbitrates the interactions among the processes which exist in separate guest domains. A fast and high-performing hypervisor is the quintessential key to the successful implementation of this design.

7 Legal Statement

This work represents the personal views of the author and is a technical analysis of an architecture; it does not necessarily represent the views of MontaVista Software, Inc.

Furthermore, the author is not an attorney and makes no judgment or recommendation on legal (and specifically GPL) ramifications of the proposed design; such issues are outside the scope of this article, and should be dealt with via consulting with a GPL attorney/law practitioner.

References

[1] Russell Coker. Porting nsa security enhanced linux to hand-held devices. In *Proceedings of the Linux Symposium*. Ottawa Linux Symposium, July 2003.

[2] William R. Hamburgen, Deborah A. Wallach, Marc A. Viredaz, Lawrence S. Brakmo, Carl A. Waldspurger, Joel F. Bartlett, Timothy Mann, and Keith I. Farkas. Itsy: Stretching the Bounds of Mobile Computing. *IEEE Computer*, 34(4):28–35, April 2001.

[3] Jorrit N. Herder, Herbert Bos, Andrew S. Tenenbaum. A Lightweight Method for Building Reliable Operating Systems Despite Unreliable Device Drivers. Technical report, Dept. of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, 2006.

Proceedings of the Linux Symposium

Volume Two

June 27th–30th, 2007
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Martin Bligh, *Google*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
John Feeney, *Red Hat, Inc.*
Len DiMaggio, *Red Hat, Inc.*
John Poelstra, *Red Hat, Inc.*