LINUTRONIX

Embedded Secure-Boot

Sicherheit und Systemintegrität spielen in Embedded-Systemen eine immer gewichtigere Rolle. Dabei erstrecken sich die Sicherheitsaspekte über die gesamte Systemarchitektur. Ein hohes Sicherheitsniveau kann mit dem offenem Betriebssystem Linux erreicht werden. Was dabei beachtet werden sollte, wird im Folgenden dargestellt.

Die Anforderungen an Embedded-Systeme wie z.B. Steuerungen steigen seit eingien Jahren nicht nur in Bezug auf Funktionalität. Das Thema Sicherheit und System-Integrität drängt sich unaufhaltsam ins Bewusstsein einer immer breiteren Öffentlichkeit.

Betrachtet man das Gesamt-Konzept eines Steuerungssystems, so fällt ins Auge, dass sich Sicherheits-Aspekte über die gesamte System-Architektur erstrecken. Die jeweiligen System-Bereiche wie z.B. Netzwerk, Laufzeitumgebung und das Datei-System stellen unterschiedliche Anforderungen in Bezug auf Sicherheit an das System-Konzept. Für diese Anforderungen bietet das offene Betriebssystem GNU/Linux eine Vielzahl an Funktionen, die – eine korrekte Umsetzung vorausgesetzt – in den jeweiligen Bereichen zu einem deutlichen Anstieg des Sicherheits-Niveaus führen.

Ein erheblicher Teil von Basis-Sicherheitsfunktionen wird erfreulicherweise mittlerweile als „Standard“ für ein Betriebssystem im Embedded-Umfeld vorausgesetzt. Heute muss niemand mehr von den Vorteilen eines Multi-User-Konzepts und der darauf basierenden Zugriffssteuerung auf Dateisystem-Ressourcen überzeugt werden.

Über solche Basis-Sicherheitsfeatures hinaus bietet der Linux-Kernel auch die Funktionalität, um die System-Integrität sicherzustellen. So bietet die Integrity Measurement Architec-

ture (IMA) im Linux-Kernel die Möglichkeit, die Integrität des Laufzeitsystems und seiner Ressourcen festzustellen, diese zu beurteilen und gegebenenfalls auf Verletzungen derselben zu reagieren.

Ein weiterer wichtiger Baustein zur Steigerung der System-Integrität ist die Signatur-Überprüfung beim Laden von Kernel Modulen, die der Linux-Kernel seit der Version 3.7 anbietet. Dem System-Integrator ist damit die Möglichkeit gegeben, das Laden nicht autorisierter (da von ihm nicht signierter) Kernel-Module komplett zu unterbinden. Selbst wenn bei einem Angriff auf das System der Angreifer Administrator-Rechte erlangen sollte, wird dieses Feature das Laden von kompromittierten Kernel-Modulen erfolgreich verhindern.

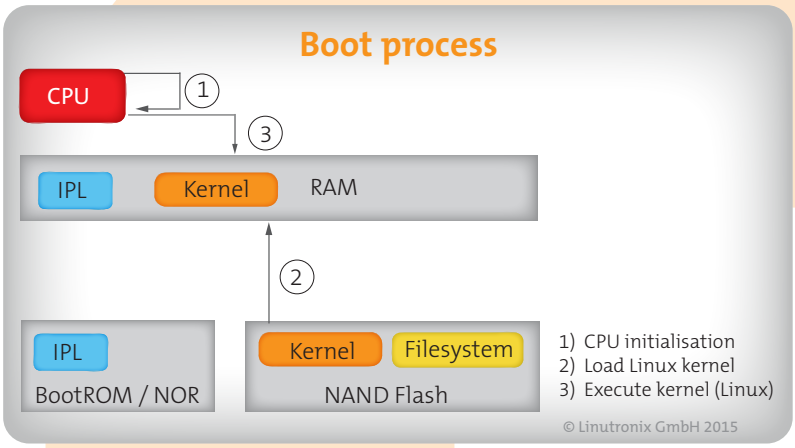

Bild 1: Der Boot-Vorgang ohne Integritätsprüfung erfolgt in 3 Schritten

Doch wie schützt man sich gegen eine mehrstufige Attacke, bei der im ersten Schritt der Boot-Kernel im Flash-Speicher durch einen kompromittierten Kernel ohne die oben genannten Sicherheits-Funktionen ersetzt wird und im zweiten Schritt dann nach einem Neustart das restliche System? Im folgenden wird ein Konzept zum Laden eines integren Kernels skizziert werden, das die Eigenschaft besitzt, eine unautorisierte Manipulation des Linux-Kernels von einem autorisierten Update zu unterscheiden.

Boot-Vorgang ohne Integritäts-Prüfung

Der Boot-Vorgang gliedert sich üblicherweise in drei Schritte, die – unabhängig vom verwendeten Bootloader – durchlaufen werden müssen.

- Im ersten Schritt wird die CPU und ihre Peripherie initialisiert.
- Im zweiten Schritt wird der Kernel aus einem Massenspeicher (z.B. NAND-Flash) in den Arbeitsspeicher geladen.
- Im dritten Schritt wird der Kernel zur Ausführung gebracht. Bild 1 stellt diese Schritte schematisch dar.

Die bisher erwähnten Sicherheits- und Integritäts-Funktionen - wie das Laden ausschließlich signierter Kernel-Module, sowie das Sicherstellen der Dateiintegrität - greifen erst ab dem dritten Schritt und schützen das Dateisystem im NAND-Flash gegen Manipulation. Der ebenfalls dort liegende Linux-Kernel sowie der Bootloader (IPL) im Boot-ROM/NOR sind nicht geschützt.

Das hier vorgestellte Konzept geht davon aus, dass der minimale Bootloader sehr selten aktualisiert werden muss. Wird das Konzept mit einem Boot-ROM umgesetzt, ist der Schutz durch die Wahl eines Read-Only-Memory implizit gegeben. Wird ein NOR verwendet, kann der schreibende Zugriffsschutz mit einfacher Hardware realisiert werden, etwa mit einem Schlüsselschalter im Write-Enable-Signal des NOR-Bausteins. Eine Modifikation des Bootloaders über einen Remote-Zugang kann damit ausgeschlossen werden. Im weiteren wird daher davon ausgegangen, dass der Inhalt des Boot-ROM bzw. des NOR sicher ist, ihm also vertraut werden kann. (man spricht hier dann auch von einem sicheren Anker).

Schutz vor unerlaubten Modifikationen

Der Schutz des Linux-Kernels soll im Folgenden beleuchtet werden. Ziel ist es, den Kernel vor unerlaubten Modifikationen zu schützen und gleichzeitig zu gewährleisten, dass er durch autorisierte Prozesse und Personen weiterhin aktualisiert werden kann.

Seit der Veröffentlichung von Auguste Kerckhoff's Grundsatz der modernen Kryptographie im Jahr 1983 ist bekannt, dass in einem System die Sicherheit nicht durch die Verwendung eines geheimen Verfahrens gewährleistet werden sollte, sondern durch die Verwendung sicherer, offener Verfahren in Kombination mit sicheren Schlüsseln. Das hier skizzierte Konzept setzt daher nicht auf „Security by Obscurity", sondern auf die Verwendung standardisierter kryptographischer Verfahren. Es empfiehlt sich darüber hinaus, bei der Umsetzung auf Open-Source-Software zurückzugreifen, weil sich

damit unter anderem die Möglichkeit einer Implementierungs-Überprüfung eröffnet.

Zum Einsatz kommen in diesem Konzept Signaturen und Zertifikate, welche mittels asymmetrischer, kryptografischer Verfahren (Public-Private-Key Verfahren) erstellt werden. Mit diesen Zertifikaten lassen sich sogenannte Vertrauens-Ketten realisieren.

Eine solche Vertrauens-Kette wird während des Bootvorgangs aufgebaut. Der Bootloader erhält ein Wurzel-Zertifikat (das **System-Zertifikat**), welches aus einem Schlüssel-Paar ($SysK_{priv}$ und $SysK_{pub}$) des Hardware-Herstellers generiert wird. Der System-Integrator, der für die Bereitstellung des Linux-Kernels und des dazu passenden Dateisystems verantwortlich zeichnet, erstellt mit einem eigenen Schlüssel-Paar ($WorkK_{priv}$ und $WorkK_{pub}$) ein eigenes Zertifikat (das **Arbeits-Zertifikat**), welches vom Hardware-Hersteller signiert wird. Wird nun vom Linux-Kernel eine Signatur mittels des Schlüssels $WorkK_{priv}$ erzeugt, so kann die Integrität dieses Linux-Kernels mit Hilfe des Arbeits-Zertifikats validiert werden. Die Vertrauenskette gliedert sich somit in zwei Schritte:

1. Das Arbeits-Zertifikat ist mit Hilfe des Systems-Zertifikats auf seine Vertrauenswürdigkeit überprüfbar.

2. Die Integrität des Linux-Kernels kann mittels des (vertrauenswürdigen) Arbeits-Zertifikats verifiziert werden.

Boot-Vorgang mit Integritätsprüfung

Diese Schritte lassen sich nun in den Boot-Vorgang integrieren. Sie fügen sich vor der Ausführung des Linux Kernels ein. Schritt 1 und Schritt 2 bleiben unverändert, zuerst wird die CPU initialisiert und der Linux-Kernel aus dem NAND-Flash geladen. Schritt 3 kommt neu hinzu. Hier wird das Arbeits-Zertifikat geladen und mittels des System-Zertifikats überprüft. Ist die Überprüfung erfolgreich, dann ist sichergestellt, dass dem Arbeitszertifikat für den weiteren Boot-Vorgang vertraut werden kann.

In Schritt 4 wird die Signatur des Linux-Kernels (welcher sich seit Schritt 2 bereits im Speicher befindet) geladen und mit Hilfe des Arbeitszertifikats überprüft. Nach einer erfolgreichen Verifikation der Signatur ist sichergestellt, dass der Linux-Kernel, so wie er vom System-Hersteller bereitgestellt und signiert wurde, im RAM liegt. Das Ausführen des eben verifizierten Linux-Kernels in Schritt 5 schließt den Vorgang ab.

Konnten alle Überprüfungen in Schritt 3 und 4 erfolgreich abgeschlossen werden, so ist die Integrität des Systems lückenlos nachgewiesen. Wurde eine Verletzung der Integrität festgestellt, kann der Boot-Vorgang abgebrochen werden. Ein manipuliertes System ist in jedem Fall von einem integren unterscheidbar.

Zu der Integritätsüberprüfung des Linux-Kernels bietet dieses Konzept jedoch noch weitere Vorteile. Durch das Auslesen und Prüfen des System- als auch der Arbeits-Zertifikate kann eine Post-Mortem-Analyse Hinweise auf eine Manipulation und deren Urheber geben. Hardware-Hersteller und System-Integrator haben die Möglichkeit, in einem Garantie- oder Gewährleistungsfall eine nicht autorisierte Modifikation des Systems nachweisen zu können.

Ein weiteres Merkmal dieses Konzepts ist, dass auf ausgelieferten Systemen nur öffentliche Schlüssel gespeichert sind. Es besteht zu keiner Zeit die Notwendigkeit, einen geheimen Schlüssel auf dem System abzulegen! Die öffentlichen Schlüssel können ausschließlich dazu genutzt werden,

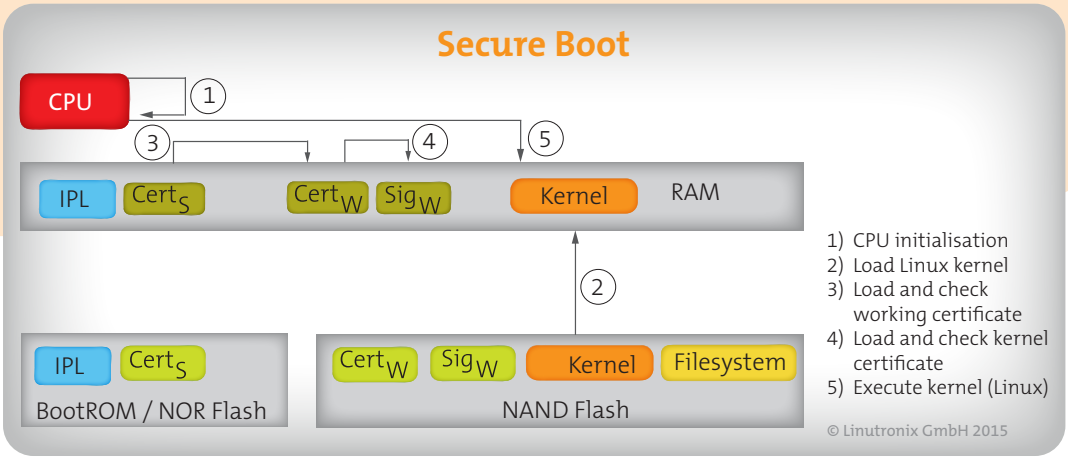

Bild 2: Beim Boot-Vorgang mit Integritätsprüfung kommen zwei Schritte hinzu: jeweils das Laden und Prüfen von Zertifikat und Signatur

die Integrität zu überprüfen. Mit ihnen lassen sich weder weitere Zertifikate noch Linux-Kernel Signaturen erzeugen.

Grenzen des Konzeptes

Die Ausführung zeigt, dass die Systemsicherheit deutlich erhöht werden kann, der Schutz jedoch auch seine Grenzen besitzt. So setzt dieses Konzept zwingend voraus, dass ein sakrosankter Bereich im System existiert, dem vertraut werden kann, um dort den Bootloader und das System-Zertifikat abzulegen. Einen solchen Bereich kann man mit einfachen Mitteln bereitstellen, solange ein Angreifer keinen physikalischen Zugang zum System hat. Muss man mit einem physikalischen Zuagng zum System rechnen, kann der Schutz dieses Bereichs ohne den Einsatz zusätzlicher Hardware nicht sichergestellt werden. In diesem Fall ist der Einsatz spezieller Hardware, wie z.B. eines Trusted Platform Modules (TPM) unausweichlich, um das System-Zertifikat vor Modifikationen zu schützen.

Konzept ohne Gehimniskrämerei

Das Konzept zeichnet sich unter anderem dadurch aus, dass sich spezielle Hardware zu einer Erhöhung des Systemschutzes nahtlos in den Ablauf des Konzepts integrieren lässt

gen Schritte, wie das Erzeugen von Schlüssel-Paaren, Zertifikaten und Signaturen des Linux-Kernels lassen sich in bestehende Prozess-Abläufe nahtlos integrieren. Das Konzept kommt durchgehend ohne Geheimniskrämerei aus. Kein einziger Teilschritt des Konzepts unterliegt einer Geheimhaltung, durch reines Reverse-Engineering geht keine Gefährdung des Integritäts-Schutzes aus.

Es besteht also eine echte, funktionierende Alternative zu „Security by Obscurity“.

Update, Security und System

Das hier gezeigte sichere Bootverfahren steht einem Updateprozess im Feld nicht im Weg. Ein Update mit signierten Modulen ist, die richtigen Werkzeuge vorausgesetzt, einfach und effizient zu erstellen.

Linutronix bietet Ihnen hierzu die benötigten Tools an. Vom Buildsystem ELBE über ein modernes Updatekonzept (im Bootloader bzw. Linuxkernel) bis hin zu einem exakt auf Ihre Bedürfnisse abgestimmten Security-Konzept für Ihr gesamtes System. Das kann von „einfachen“ Mitteln wie einem Laufzeitsystem im read-only Bereich bis hin zu Systemen mit Zertifikaten, IMA, EVM oder secure policy (MAC, ...) reichen. Bild 3 zeigt einen möglichen Ansatz, wie so ein System aufgebaut sein könnte.

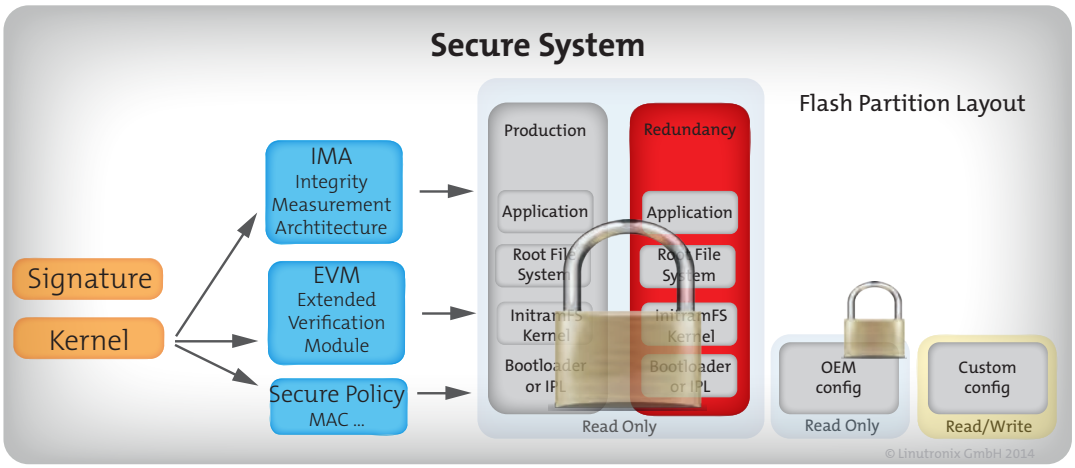

Bild3: Beispiel-Layout eines sicheren Systems

Zusammenfassend wird deutlich, dass mit dem hier vorgestellten Konzept nicht nur ein hohes Maß an Sicherheit gegen Manipulationen erreicht werden kann, es ist auch an die unterschiedlichsten Anforderungen und Bedürfnisse anpassbar. Alle notwendi-

Haben wir Ihr Interesse geweckt? Wollen Sie mehr wissen? Rufen Sie einfach an, oder senden Sie uns eine Email.

07/2015_V1.0

LINUTRONIX GMBH
Auf dem Berg 3 | D-88690 Uhldingen - Mühlhofen
Telefon +49 7556 4521 890 | Fax +49 7556 919 886
info@linutronix.de | www.linutronix.de

LINUTRONIX
LINUX FOR INDUSTRY