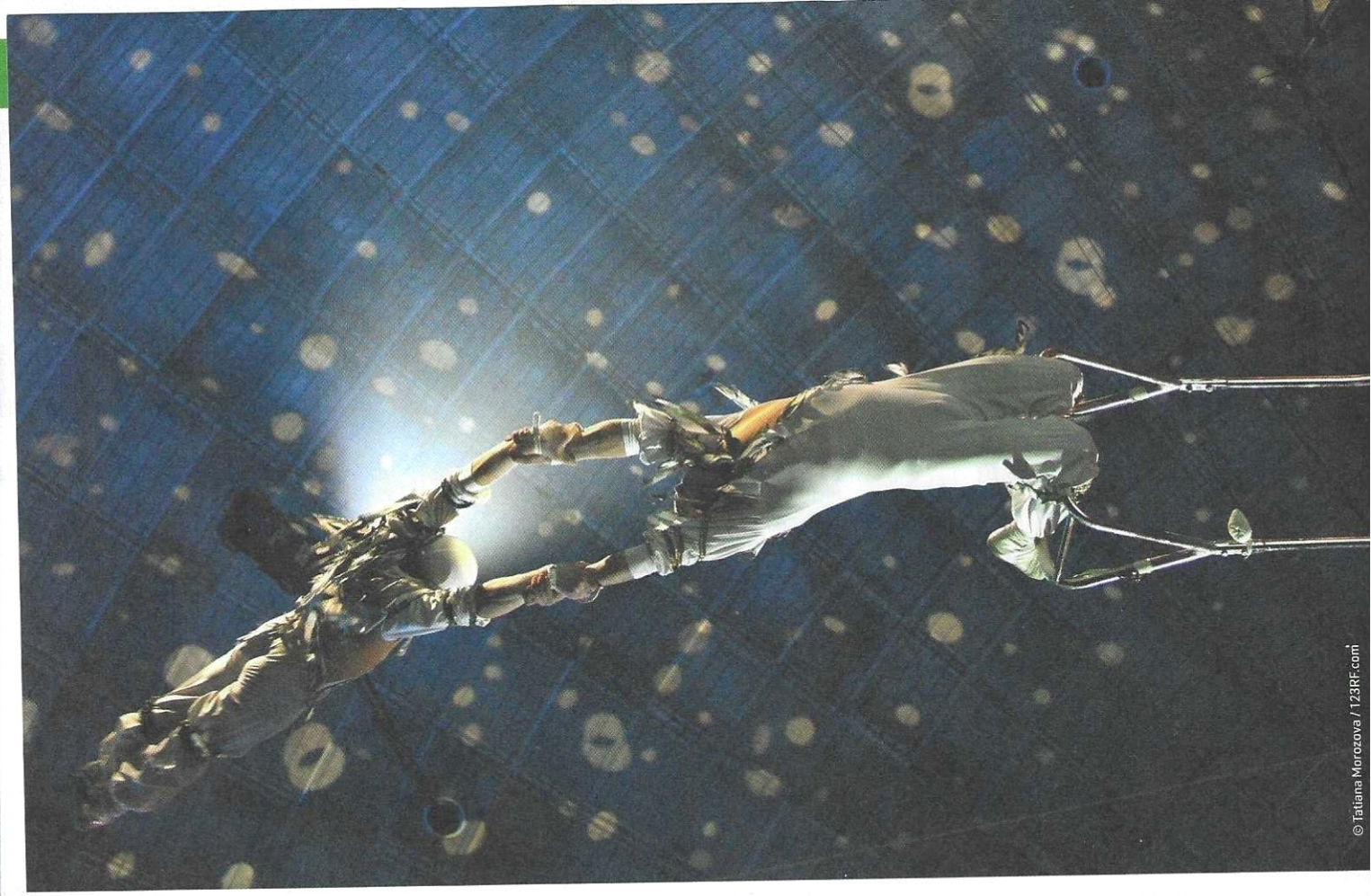


Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 121

Luftnummer

Eingebettete Systeme auf sichere und robuste Weise zu aktualisieren ist extrem anspruchsvoll. Software allein und deren komplexe Konfiguration reichen dazu nicht aus.

Eva-Katharina Kunst, Jürgen Quade

Bei PC, Tablet und Smartphone ist das regelmäßige *Over-the-Air*-Update (OTA) gelebter Alltag. Über diesen Weg stopfen Hersteller Sicherheitslücken im Kernel, in der übrigen Systemsoftware oder in den Applikationen. Manchmal kommt mit einem OTA-Update auch eine neue Funktion daher. Unglücklicherweise nutzen Hersteller heimliche Updates aber auch dazu, Funktionalitäten wieder zu beschneiden.

Smart-Home-Geräte, Internet-of-Things-Komponenten oder billige Elektroware dagegen lassen uns IT-sicherheitstechnisch häufig im Regen stehen. Hohe Stückzahlen haben Vorrang, die Gerätepflege ist für die Hersteller ein zu kalkulierender Kostenpunkt. Dabei kann man als Hersteller dank OTA reichlich Geld sparen. Während Tesla mit einem Knopfdruck über Nacht eine fehlerbereinigte Linux-Software auf seine Fahrzeugflotte ausrollt, rufen VW, BMW, Mercedes und Co. die Fahrzeuge kleinlaut einzeln in die Werkstatt, wo sie in solider Handarbeit Bits und Bytes auf Steuergeräte flashen. Kann man mal machen.

Flotten bricken

Am Beispiel, wie schwer sich die klassische Automobilbranche mit OTA tut, zeigt sich, dass die Aktualisierung von Firmware, Kernel, System und Applikationen keineswegs trivial ist. Um nicht gleich eine ganze Fahrzeugflotte zu bricken, muss sichergestellt sein, dass das Update zur vorhandenen Hard- und Basissoftware passt (Versionierung). Darüber hinaus gilt es zu gewährleisten, dass die Software nicht von einem windigen Hacker stammt, sondern vom Hersteller

Die Autoren

Eva-Katharina Kunst ist seit den Anfängen von Linux Fan von Open Source. Jürgen Quade, Professor an der Hochschule Niederrhein, führt auch für Unternehmen Schulungen zu den Themen Treiberprogrammierung und Embedded Linux durch.

selbst (Authentizität) und von keiner weiteren Instanz manipuliert wurde (Integrität). Schließlich sollte man sich dahingehend versichern, dass im Fall eines Abbruchs per Recovery ein funktionstüchtiger Zustand erhalten bleibt (Robustheit). Weitere Anforderungen betreffen das Monitoring und das Deployment, also das Ausrollen eines Updates.

Im Detail wird es noch komplizierter. Gerade eingebettete Systeme legen oft Systemsoftware im Flash-Speicher ab, der sich nur per mehr oder minder proprietärem Protokoll schreiben lässt. Außerdem muss für ein Update eventuell partitioniert und formatiert werden. Alternativ schreibt man Daten als Image auf eine Partition. Außerdem sind möglicherweise nicht ausreichend Ressourcen vorhanden, um System plus Update im Storage zu haben. Für die Authentifizierung und den Integritätscheck müssen Zertifikate erstellt, gepflegt und sicher sowohl beim Hersteller als auch im Gerät abgelegt werden. Und dann bleibt noch das Problem des Ausrollens selbst.

Alles öko

Die Out-of-the-Box-Lösung, um beispielsweise den Kernel eines eingebetteten Systems sicher zu erneuern, gibt es allerdings (noch) nicht. Hier konkurrieren mehrere Systeme um die Gunst der Entwicklerinnen und Entwickler. Mit RAUC und SWUpdate stehen quelloffene Lösungen zur Verfügung – übrigens eine Voraussetzung, um überhaupt ein Update sicher vornehmen zu können.

Beide Lösungen stammen interessanterweise aus Deutschland. Hinter SWUpdate steht die Firma DENX in Gröbenzell, bei RAUC ist es Pengutronix aus Hildesheim. Schon mal vorweggenommen: Updates im Allgemeinen und OTA-Updates im Speziellen stellen einen sehr komplexen Vorgang dar, der in betriebliche Abläufe eingebettet werden muss und Infrastruktur benötigt. Software wie SWUpdate und RAUC sind damit Basisbausteine eines von den Entwicklerinnen und Entwicklern zu realisierenden Ökosystems (einer Architektur).

Grundpfeiler eines solchen Ökosystems sind erstens das Erstellen eines Updates und zweitens dessen Installation. Da das Deployment weitestgehend unabhängig vom Update selbst erfolgt, lassen wir es zunächst beiseite. Zum Erstellen eines Updates muss neben den eigentlichen Update-Dateien für das eingebettete System, also beispielsweise neben einem neuen Kernel oder einem neuen Root-Dateisystem, auch eine Konfiguration existieren (Listing 1).

Diese Konfiguration – bei RAUC als Manifest bezeichnet – stellt insbesondere die Eignung (Hard- und Softwarerevision) für das zu aktualisierende Gerät und die Integrität sicher. Eine digitale Unterschrift gewährleistet zudem die Authentizität. Manifest plus die eigentlichen Installationskomponenten packt man zusammen in eine einzelne Datei, die Update-Datei beziehungsweise bei RAUC das sogenannte Bundle.

SWUpdate verwendet als Format für diese Aktualisierungsdatei ein CPIO-Archiv, das auf dem Target für die Installation ausgepackt wird. RAUC vermeidet das Auspacken auf dem Zielsystem und spart damit Speicherressourcen. Dank eines Squash-Filesystems gelingt der Zugriff auf die Daten direkt. Dass die Daten im SquashFS komprimiert sind, ist für den gedachten Einsatz ein glücklicher Nebeneffekt.

A/B versus A/R

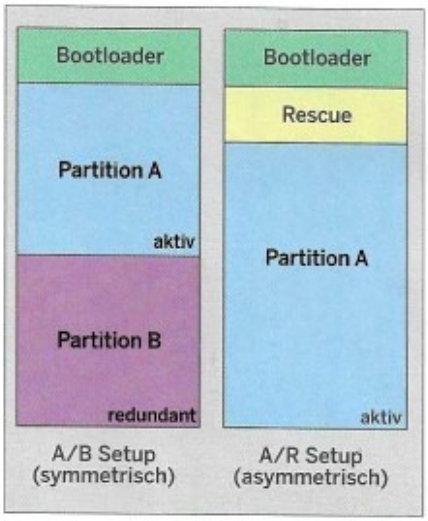
Im Ökosystem eingebetteter Systeme sprechen Entwickler bei Softwareaktualisierungen gern von symmetrischen A/B-Setups oder asymmetrischen A/Recovery-Setups 1. Beide Varianten wollen sicherstellen, dass ein System auch bei einem fehlerhaften Update nicht ausfällt. Der Begriff Setup steht in diesem Kontext als Synonym für Speicherbereiche, beispielsweise Partitionen.

Listing 1: `manifest.raucm`

```
[update]
compatible=HW-Rev 1.20
version=2022-02-20

[image.rootfs]
filename=rootfs.img

[image.appfs]
filename=appfs.img
```



1 Redundanz sorgt für Robustheit.

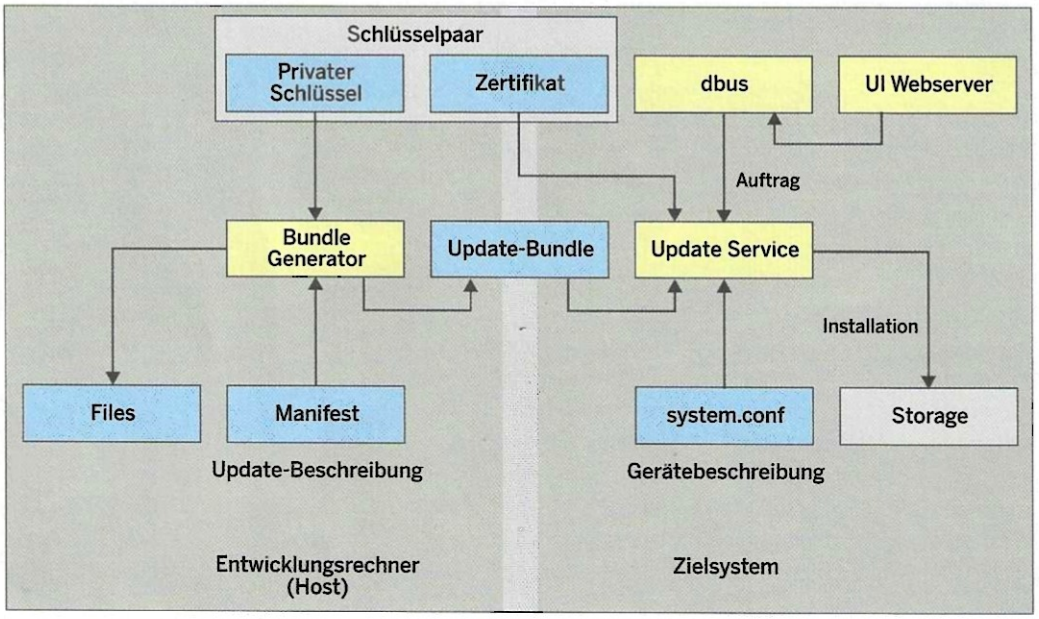
Wenn im Fall des A/B-Setups der Rechner mit der Partition A gebootet wurde, wird B aktualisiert und für den nächsten Neustart aktiviert. Schlägt nach dem nächsten Reboot beispielsweise ein Watchdog zu, weil das neue System doch Probleme bereitet, wählt der Bootloader beim nachfolgenden Start wieder das funktionierende A-Setup aus. Anderenfalls, wenn von B erfolgreich gebootet wurde, erfolgt das nächste Update auf der A-Partition.

Steht für ein A/B-Setup nicht genügend Flash-Speicher (secondary storage) zur Verfügung, bietet sich das A/R-Setup an. Das Recovery- oder Rescue-System benötigt erheblich weniger Ressourcen als ein redundantes Setup und wird als funktionstüchtiges System mit nur einer Aufgabe, der Softwareinstallation, typischerweise nicht aktualisiert.

Slot Machines

RAUC verwendet für die Beschreibung des Setups auf dem Zielsystem das antiquierte aus der Microsoft-Welt bekannte INI-Format, das in Sektionen gegliedert Schlüssel-Wert-Zuweisungen enthält (Listing 2). Die einzelnen Speicherbereiche werden hier als Slots bezeichnet, beispielsweise `slot.rootfs.0` oder `slot.rootfs.1`.

Slots repräsentieren ein Gerät, eine Partition als Teil eines physikalischen Laufwerks, ein logisches Laufwerk (Volume) oder aber auch einfach eine Datei.



2 Ein Ökosystem für ein sicheres und robustes Softwareupdate.

Ein Slot wird durch die Pfadangabe beschrieben (Schlüsselwort `device`), falls relevant durch den Dateisystemtyp (Schlüsselwort `type`) und durch eine Referenz zum Bootloader (Schlüsselwort `bootname`).

Da ein gerade in Verwendung befindlicher Slot nicht aktualisiert werden sollte, geht RAUC typischerweise von zwei Slots pro Systemteil aus: dem primären Slot (A) und dem redundanten Slot (B). Einer der beiden ist aktiv, der andere steht für ein Update bereit.

Die Konfiguration im Bundle, das Manifest also, ordnet schließlich die im Päckchen befindlichen Dateien den Slots in der Systemkonfiguration zu. Das erfolgt wieder anhand des Bereichsnamens (section name), beispielsweise `[image.rootfs]`. Anhand des Dateityps erkennt RAUC, wie es ein Update auszuführen hat. Bei einer Dateierweiterung mit Namen `ext4` beispielsweise wird der Inhalt der Datei schlichtweg auf die in der Systemkonfiguration angegebene Gerätedatei kopiert – zack, fertig. Handelt es sich dagegen um ein TAR-Archiv, wird zunächst ein Filesystem angelegt, das neu angelegte Dateisystem gemountet, der Inhalt aus dem Tarball dort ausgepackt und anschließend `umount` aufgerufen.

Neben den eingebauten Verfahren ermöglichen sowohl SWUpdate als auch RAUC über sogenannte Hooks und Handler das Einbinden proprietärer Update-Mechanismen. Es handelt sich um eigene Programme oder Skripte, die der Installer aufruft und mit den notwendigen Parametern versorgt. Der Unterschied zwischen Hook und Handler besteht darin, dass ein Handler unabhängig von der Update-Datei auf dem Zielsystem installiert ist und vom Installer aufgerufen wird. Der Hook wird zwar ebenfalls vom Installer aufgerufen, ist aber Teil der Update-Datei: Handler gleich systemspezifisch, Hook gleich Update-spezifisch. Beim Handler unterscheidet man im übrigen noch zwischen einem Pre- und einem Post-Install, die – ihrem Namen entsprechend – vor respektive nach der eigentlichen Installation aufgerufen werden.

Insbesondere RAUC, das den Schwerpunkt auf Robustheit legt, tauscht sich vor und nach der Installation mit dem Bootloader über die Auswahl der Bootpartition aus und informiert darüber, ob der Boot-Vorgang erfolgreich war oder nicht. Der Bootloader selbst muss dem Installer vor der Installation mitteilen, über welche Partition der Boot-Vorgang erfolgt ist, damit diese Partition nicht dummerweise überschrieben wird. RAUC unterstützt ab Werk die bekannten Bootloader wie U-Boot, Grub, UEFI oder Barebox. Kommen andere Bootloader zum Einsatz, fällt Handarbeit an.

Listing 2: Systembeschreibung des Targets

```
[system]
compatible=HW-Rev 1.20
bootloader=grub
grubenv=/boot/grub/grubenv

[keyring]
path=ca.cert.pem

[slot.rootfs.0]
device=/dev/mmcblk0p0
bootname=A

[slot.rootfs.1]
device=/dev/mmcblk0p1
bootname=B

[slot.appfs.0]
device=/dev/null
parent=rootfs.0

[slot.appfs.1]
device=/dev/mmcblk0p2
parent=rootfs.1
```

Vertrauen schaffen

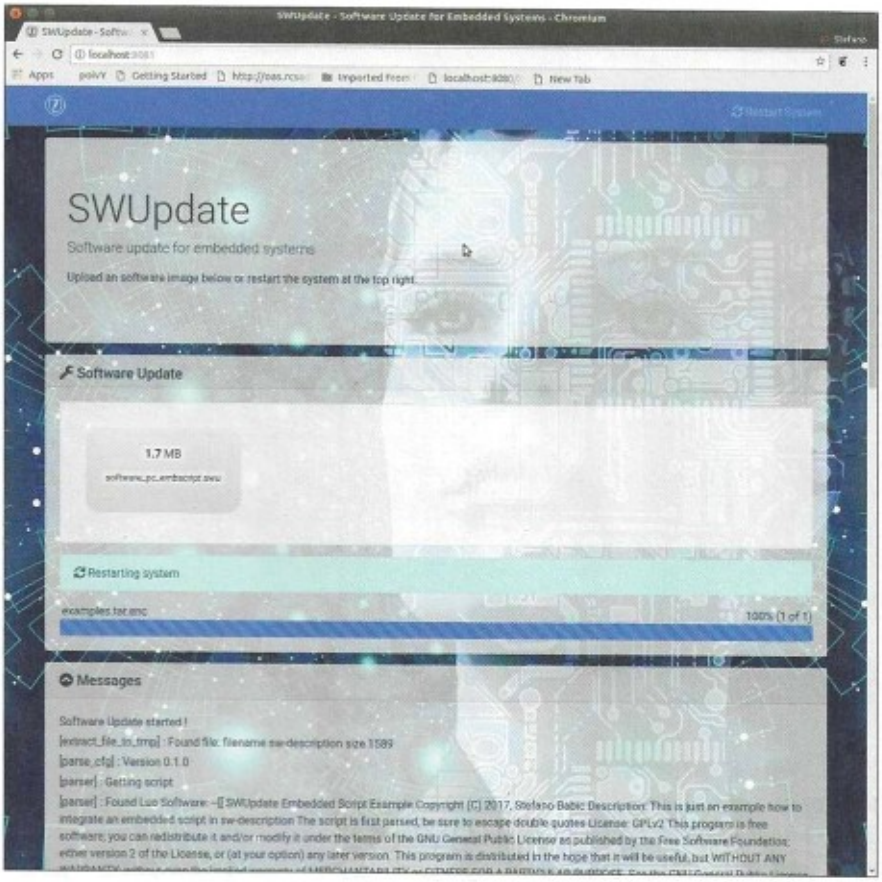
Bevor man überhaupt ein Update erstellen kann, muss es ein Schlüsselpaar geben, das die sichere Authentifizierung gewährleistet. Zum Erstellen bietet sich OpenSSL an, beziehungsweise für komplexere Setups dazu aufgesetzte Werkzeuge wie beispielsweise Easy-RSA.

Im einfachsten Fall genügt ein einzelnes Schlüsselpaar, wobei der öffentliche Schlüssel in ein Zertifikat verpackt wird. Wird das Zertifikat mithilfe des privaten Schlüssels digital unterschrieben, spricht

man von einem Self-signed Certificate. Das Zertifikat stellt mit dem integrierten öffentlichen Schlüssel die Unterschriftenprobe dar und muss daher auf das Zielsystem abgelegt und dort vor unlegitimierten Modifikationen geschützt werden. Hier stellt weniger die Vertraulichkeit des Zertifikats das Problem dar, sondern vielmehr die Integrität beziehungsweise der Schutz vor Manipulation: Wenn es Angreifern gelingt, das Zertifikat gegen eine andere Unterschriftenprobe auszutauschen, können sie unautorisierte Updates einspielen und damit das Gerät kontrollieren.

Im professionellen Umfeld ist eine richtige PKI (Public Key Infrastructure) sinnvoll, die aus einem Master-Schlüsselpaar, einer sogenannten Certification Authority und weiteren Intermediate-Schlüsselpaaren sowie zugehörigen Rückruflisten (Certification Revocation List, CRL) besteht. Ein solch hierarchisches Zertifikatsmodell hat den Vorteil, dass beim Abhandenkommen eines privaten Schlüssels nicht gleich alle damit signierten Updates hinfällig respektive kompromittiert sind. Nur der zum kompromittierten Schlüssel gehörende Teilbaum muss erneuert werden, was dank Master Key weiterhin klappt.

Dazu sind auf dem Gerät – ähnlich wie bei Secure Boot beim PC – mehrere Unterschriftenproben abgelegt, beispielsweise ein Hersteller- und ein Gerätezertifikat. Updates werden typischerweise mit dem Geräteschlüssel signiert.

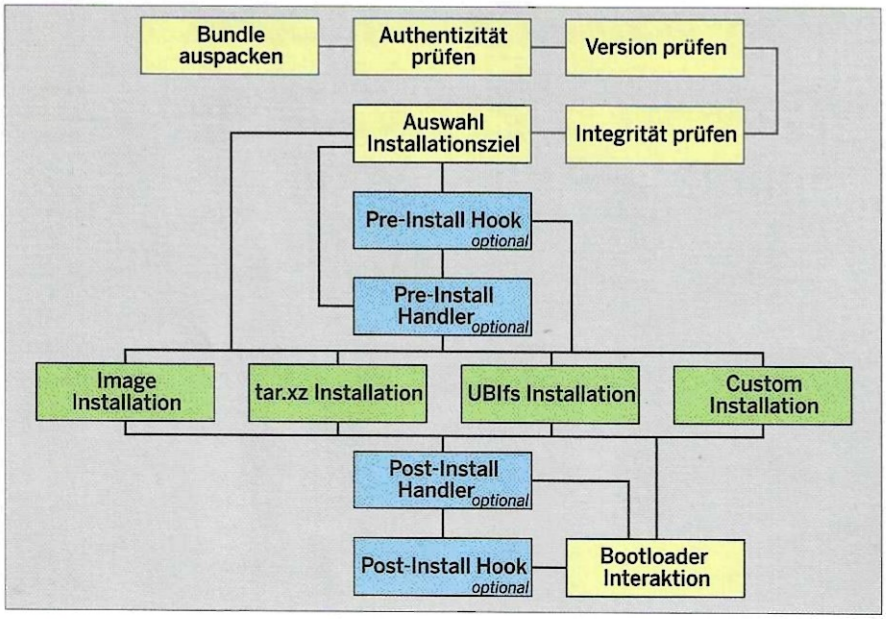


3 **Deployment via Webserver bei SWUpdate.**

Kommt der abhanden und wird damit das Gerätezertifikat als ungültig markiert und zurückgerufen, lässt sich mithilfe des Herstellerschlüssels ein neues Gerätezertifikat hinterlegen, mit dem die nachfolgenden Updates verifiziert werden.

Up to date

Die Prüfung der Integrität und Authentizität erfolgt auf dem Zielsystem klassischerweise über einen dort ständig im Hintergrund laufenden Rechenprozess



4 Es erfordert einigen Aufwand, ein Update sicher und robust einzuspielen.

(Service). Der erhält seine Aufträge per Interprozesskommunikation. Während SWUpdate ein eigenes Protokoll definiert, setzt RAUC auf D-Bus 2.

D-Bus ist ein vor allem in Linux-Desktop-Umgebungen eingesetztes, standardisiertes Nachrichtensystem, das einen Service als Nachrichten-Broker zur Verfügung stellt. Anwendungen verbinden sich mit dem Broker und tauschen mit dessen Hilfe Messages über einen definierten Namensraum aus, der Dateipfaden ähnelt. Der auf dem Zielsystem laufende Update-Prozess registriert sich beim Nachrichten-Broker und nimmt über diesen Weg Aufträge entgegen.

Über diese Interfaces wird nicht nur der Installationsprozess selbst aktiviert, sondern darüber lässt sich auch ein Monitoring betreiben und beispielsweise der Installationsfortschritt abfragen. Diese auf einem zentralen Service basierende Architektur trennt sauber Funktionalität und Oberfläche und offeriert die Möglichkeit zur individualisierten Integration in das Geräte-Ökosystem. So kann man mit wenig Aufwand ein schnörkelloses Command Line Interface realisieren oder eine aufwendige, grafisch anspruchsvolle Kommandozentrale 3.

Damit ein Update-Bundle eingespielt werden kann, transferiert (deployt) man es zunächst auf das Zielsystem. Der Update-Service wird mit der Installation der neuen Softwareversion beauftragt, extrahiert als Erstes das Manifest und prüft die Authentizität 4. Handelt es sich um ein autorisiertes Update, gleicht die Installationssoftware unter Zuhilfenahme der auf dem Gerät liegenden Konfiguration die Hard- beziehungsweise Softwarerevision ab und prüft die Integrität der im Bundle enthaltenen Komponenten.

Ist alles so weit im grünen Bereich, kann die eigentliche Installation beginnen. Zunächst wird ein potenziell vorhandener Pre-Install-Hook aufgerufen, dann erfolgt auf Basis der Systemkonfiguration und der aktiven Partition die Aktualisierung der im Bundle definierten Komponenten. Die Aktualisierung verwendet entweder eingebaute Installationsschemata oder den Installations-Hook beziehungsweise einen auf dem System vorhandenen Installationshandler.

Falls vorhandenen, ruft der Installer den Post-Install-Hook auf, ebenso – falls vorhanden – einen Post-Install-Handler. Im Anschluss markiert er für den Bootloader die erfolgreiche Aktualisierung. Der aktuelle Fortschritt lässt sich während der ganzen Zeit verfolgen. Häufig wird außerdem der fällige Reboot ausgeführt.

Robust booten

Der Bootloader entscheidet, welches System es zu booten gilt. Nach erfolgter Installation setzt er die entsprechende Partition als aktiv. Außerdem kann eine Partition nach dem Booten als „gut" oder „schlecht" markiert werden.

Der Bootloader selbst realisiert den robusten Bootvorgang. Typischerweise kommt dazu ein persistenter Zähler zum Einsatz, der mit der neuen Version auf einen Anfangswert (beispielsweise 2) gesetzt und dann mit jedem Bootvorgang dekrementiert wird. Nach einem erfolgreichen Booten erhält der Zähler wieder seinen ursprünglichen Wert. War das Booten jedoch nicht erfolgreich und hat der Zähler den Wert 0 erreicht, wählt der Bootloader das Alternativsystem aus.

Idealerweise unterstützt den Bootloader außerdem ein (Hardware-)Watchdog, der ein potenziell hängendes System per schmerzlosem Reset wieder zurück zum Boot-Vorgang befördert.

Alternativlos

Um den Update-Vorgang noch einmal aus anderer Perspektive aufzudröseln: Dabei lassen sich unterschiedliche Rollen differenzieren.

Der Hersteller sorgt für die eigentliche Infrastruktur, die für das Signieren der Updates und deren Deployment sorgt. Das Update lässt sich dabei aktiv auf das Zielsystem pushen. Alternativ legt der Hersteller das Update an prominenter Stelle ab, und das Zielsystem kümmert sich eigenständig um den Transfer. Die Entwicklung hinterlegt auf dem Gerät die Unterschriftenproben, erkennt potenzielle Schwachstellen auf Geräten, erstellt die Update-Bundles und signiert sie. Der Betreiber eines Geräts schließlich legitimiert die Zeit-Slots, in denen ein Update möglich ist.

Integriert man diese Aufgaben sauber in die eigenen Geschäftsprozesse, dann bleibt der zeitliche Aufwand dafür in vertretbarem Rahmen. Allerdings: Rein aus IT-sicherheitstechnischen Gründen sind Softwareupdates alternativlos. Soviel zur Theorie – die Praxis folgt dann in der nächsten Folge dieser Reihe. (*jlu*)

Weitere Infos und interessante Links
www.lm-online.de/qr/47347