Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 122

Am Puls der Zeit

Das freie Update-Tool RAUC verspricht, Embedded-Software robust und sicher auf den aktuellen Stand zu bringen. Die Konfiguration dazu spielt allerdings nicht immer auf Anhieb mit. Eva-Katharina Kunst, Jürgen Quade

Die Autoren

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

Software-Updates für vernetzte IoT-, Smart-Home- oder Industrie-4.0-Geräte sind komplex und alternativlos. Praktisch ist fehlerfreie Software nichts als schöne Utopie. Um bekannte Fehler durch unbekannte zu ersetzen, ohne das laufende System zu bricken, überprüft die Installationsroutine Herkunft sowie Integrität der neuen Gerätesoftware und – nach dem Neustart – deren Funktionalität. Das setzt eine geeignete Hard- und Softwareversionierung voraus und ein ausgeklügeltes kryptografisches Framework (PKI).

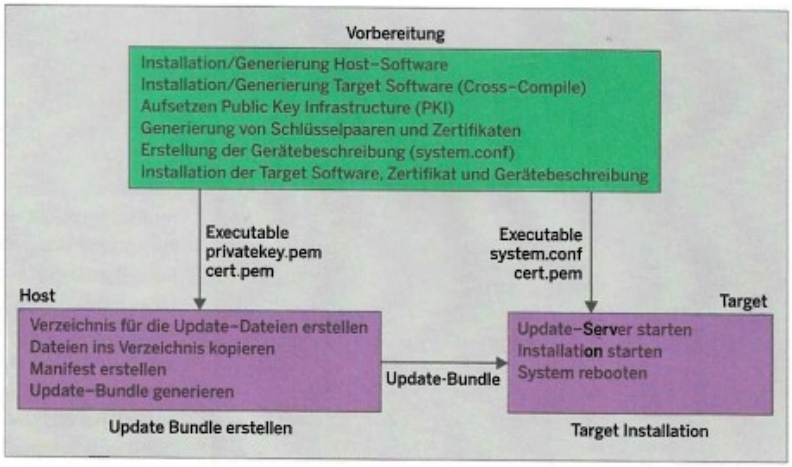
Neben der Installationsroutine auf dem zu aktualisierenden System, dem Target, braucht es zum Generieren der Update-Datei Softwareunterstützung auf dem Entwicklungsrechner. Die grundsätzlichen Anforderungen, Probleme und Architekturen eines solchen Aktualisierungs-Frameworks hat bereits die letzte Kern-Technik-Folge am Beispiel der Projekte RAUC und SWUpdate beleuchtet. Diesmal wenden wir uns der Praxis zu, installieren RAUC und testen es in einem überschaubaren Szenario.

Für das Update reicht ein einzelnes Stück Software nicht aus. Vielmehr benötigen wir sowohl Code für den Entwicklungsrechner, auf dem wir das Update-Bundle erstellen, als auch für das Target, auf dem wir das Update einspielen. Ersteres ist der Bundle-Generator, Letzteres der Update-Installer. Genau genommen umfasst RAUC auf dem Target noch mehr Komponenten, denn der eigentliche

Installer läuft als Dienst im Hintergrund (Service, Daemon) und erhält Aufträge über ein Command Line Interface (CLI). Bei RAUC ist letztlich alles ein Programm, das abhängig von den Aufrufparametern wechselweise die Funktionen Generator, Installer oder CLI ausübt.

Beim Einsatz von RAUC gehen Sie nach dem Schema aus der Abbildung 1 vor. Zunächst gilt es, die Software auf dem Entwicklungsrechner zu installieren und zu generieren, danach ist das Target an der Reihe. Im nächsten Schritt setzen Sie eine PKI zur Signatur der Updates auf und legen eine Beschreibung der Systemarchitektur des zu aktualisierenden Systems (`system.conf`) an. Gerätebeschreibung sowie Zertifikat landen schließlich auf dem Target.

Um das Update-Bundle zu produzieren, richten Sie einen Ordner ein und legen sämtliche für das Update benötigten Dateien dort ab. Zusätzlich erstellen Sie ein Manifest als Beschreibung des aktuellen Updates. Es definiert den künftigen Systemzustand, also welche im Bundle befindliche Komponente auf welcher Partition des Zielsystems installiert werden soll. Haben Sie alles beisammen, geht es im nächsten Schritt ans Generieren des Update-Bundles.



1 Der Weg zum Update von Embedded-Systemen mithilfe von RAUC.

Software marsch

Auf dem Zielsystem muss der Update-Server laufen. Befindet sich ein Update-Bundle auf dem Target, können Sie das Update selbst einspielen. Bei sauberer Integration in den Boot-Prozess sorgt RAUC dafür, dass anschließend in jedem Fall ein funktionstüchtiges System vorliegt. Funktioniert das neu installierte Update nicht, reaktiviert sich automatisch das vorherige System.

Listing 1 zeigt die Befehle, über die sich RAUC unter Ubuntu 20.04 sowohl herunterladen als auch generieren lässt. Das Kommando aus Zeile vier installiert RAUC auf dem Entwicklungsrechner. Ein erfolgreicher Generierungsvorgang setzt allerdings zusätzliche Softwarepakete voraus. Beispielsweise benötigt RAUC das Paket *libdbus-1-dev* für die Kommunikation via D-Bus und *libssl-dev* zur Verschlüsselung (Zeile drei).

Das durch den Aufruf in der letzten Zeile erzeugte Executable dient sowohl als Bundle-Generator als auch als Bundle-Installer – allerdings mit der Einschränkung, dass es für eine x86-Architektur generiert wurde und somit nur auf einem entsprechenden System läuft. In den meisten Setups dürfte es sich bei der Zielplattform um eine andere CPU-Architektur handeln, vermutlich ARM. Deswegen verwenden Sie zur Host-Target-Entwicklung einen Cross-Compiler und übersetzen RAUC damit. Installieren und Generieren funktionieren auf einem Raspberry Pi unter Pi OS analog.

So leicht das Installieren auf dem Host von der Hand geht, so viel Fingerspitzengefühl verlangt der Umgang mit der Software selbst. Für einen Test brauchen Sie neben dem Entwicklungsrechner ein Zielsystem. Gemäß dem Motto „wer keine Hardware hat, emuliert sich eine" unternehmen Sie am besten mithilfe des Emulators Qemu erste Gehversuche. Damit entfällt angenehmerweise Cross-Generieren von RAUC.

Qemu bildet die Basis für den in RAUC integrierten Softwaretest, den `./qemu-test` im Quellcodeverzeichnis von RAUC startet. Das Ergebnis des Testdurchlaufs legt RAUC in der Datei `test-suite.log` ab. Das per Qemu simulierte System verwendet ähnlich wie ein Container die Programme und die Daten des Host-Systems mit. Abgesehen von einem separaten Kernel, den sich `qemu-test` übers Netz holt, braucht es dadurch keine weitere Software.

Schlüsselfrage

Neben dem Erzeugen der Software selbst gehört es weiterhin zu den Vorbereitungen, eine PKI zu erstellen und das Zielsystem zu beschreiben. Für die PKI genügt im einfachsten Fall ein einzel-

Listing 1: Host-Installation

```
01 $ mkdir ~/update
02 $ cd ~/update
03 $ sudo apt install build-essential automake
   libtool libdbus-1-dev libglib2.0-dev libcurl3-dev
   libssl-dev libjson-glib-dev
04 $ git clone https://github.com/rauc/rauc.git
05 $ cd rauc
06 $ ./autogen.sh
07 $ ./configure
08 $ make
```

```
quade@ezs-cocka:~$ mkdir update
quade@ezs-cocka:~$ cd update/
quade@ezs-cocka:~/update$ mkdir ca
quade@ezs-cocka:~/update$ cd ca
quade@ezs-cocka:~/update/ca$ openssl req -x509 -newkey rsa:4096 \
>     -keyout key.pem -out cert.pem \
>     -days 3650 -nodes
Generating a RSA private key
.....................................................................
.....................................................................
.....................................................................
.....++++
.....................................................................
.....++++
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:NRW
Locality Name (eg, city) []:Krefeld
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Holy Moly
Organizational Unit Name (eg, section) []:IoT
Common Name (e.g. server FQDN or YOUR name) []:Linux Magazin Test Device
Email Address []:redaktion@linux-magazin.de
quade@ezs-cocka:~/update/ca$ ls
cert.pem  key.pem
quade@ezs-cocka:~/update/ca$
```

2 Achten Sie beim Schlüsselgenerieren auf Namensgleichheit.

nes Schlüsselpaar, das aus dem privaten Schlüssel und dem selbst unterschriebenen Zertifikat besteht.

Doch Vorsicht: Der Common Name (CN) im Zertifikat muss unbedingt mit dem Inhalt der Variablen `compatible` in der Gerätebeschreibungsdatei `system.conf` übereinstimmen **2**. Es empfiehlt sich, das Zertifikat in einem separaten Verzeichnis wie `~/update/ca/` per OpenSSL zu erzeugen (Listing 2).

Als Ziel dient wieder das Qemu-System, das Sie per `qemu-test` erzeugen **3**. Wie bei einem A/B-System üblich, gibt es ein aktives Root-Dateisystem sowie ein aktives Applikations-Filesystem (A) und das redundante Gegenstück (B).

Listing 2: Schlüsselpaar generieren

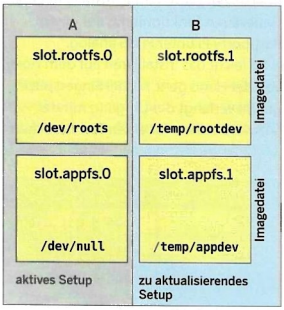
```
$ mkdir -p ~/update/ca
$ cd ~/update/ca
$ openssl req -x509 \
    -newkey rsa:4096 \
    -keyout key.pem \
    -out cert.pem \
    -days 3650 -nodes
# ACHTUNG: Common Name passend
# zu system.conf waehlen!
[...]
```

Da Sie beim Test das redundante System aktualisieren, lässt sich das aktive System recht einfach modellieren. Pragmatischerweise kommt die aktive Root-Partition (`/dev/root/`) zum Einsatz, die Applikationspartition bleibt leer (`/dev/null/`). Für das redundante System legt das Skript `qemu-test-init` beim Hochfahren per Truncate die zwei (leeren) Image-Dateien `rootdev` und `appdev` im Verzeichnis `/tmp` an.

Eine Beschreibung dieser Gerätestruktur, die RAUC interpretieren kann, zeigt Listing 3. Zusätzlich müssen in der Datei `system.conf` Informationen zur Hard- und Softwarerevision zur Verfügung stehen. Das simulierte Gerät heißt im Test `Linux Magazin Test Device`.

Außer der Datei `system.conf` legen Sie auf dem Target das Zertifikat schreibgeschützt ab, um darüber später die Authentizität und Integrität einer Update-Datei gewährleisten zu können. Danach können Sie Updates erstellen.

Auf dem Host-System mit dem privaten Schlüssel für die digitale Signatur kopieren Sie sämtliche für das Update notwendigen Dateien in das vorbereitete Verzeichnis `update/bundlefiles/`. Dort legen Sie außerdem unter dem Namen `manifest.raucm` das Manifest ab, das das Update im Detail beschreibt.



3 Partitionierung des Testsystems.

Für einen Test erstellen Sie mit den Kommandos aus Listing 4 den Ordner `bundlefiles/` und befüllen ihn mit dem Image `rootfs.img` für das Root-Dateisystem. Das Image selbst versorgen Sie mit einigen Pseudo-Files wie einer Datei, in die Sie durch `date` den aktuellen Zeitstempel schreiben. Das Applikationsdateisystem erzeugen Sie mithilfe schlichten Kopierens des Root-Filesystems. Später lassen sich die zugehörigen Slots auf dem Testsystem mithilfe dieser Image-Dateien aktualisieren.

Den Inhalt der Manifest-Datei im INI-Format zeigt Listing 5. Dort spezifizieren Sie in der Sektion `[update]` die Hardwarerevision in Form des Gerätenamens über die Variable `compatible` und vergeben

Listing 3: `system.conf`

```
[system]
compatible=Linux Magazin Test
Device
bootloader=grub
grubenv=/tmp/boot/grub/grubenv
statusfile=/tmp/rauc.status

[keyring]
path=/tmp/rauc/ca.cert.pem

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

[slot.rootfs.1]
device=/tmp/rootdev
bootname=B

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

[slot.appfs.1]
device=/tmp/appdev
parent=rootfs.1
```

über `version` eine Versionsnummer. Hier ist es wichtig, den Namen des zu aktualisierenden Systems (`Linux Magazin Test Device`) exakt so zu verwenden, wie er sich bereits im Zertifikat findet.

Weiter beschreibt das Manifest, mit welchen Dateien die Slots des Systems bespielt werden sollen. Im Beispiel steht `filename=rootfs.img` für den ersten und `filename=appfs.img` für den zweiten Slot, wobei die Angabe des Dateinamens genügt. Die für den Integritätsschutz notwendigen SHA256-Hashes erzeugt RAUC beim Erstellen des Update-Bundles ebenso selbstständig wie die Größenangabe der Datei.

```
quade@ezs-cocka:~/update$ rauc/rauc bundle -d --cert=ca/cert.pem --key=ca/key.pem bundlefiles/ 2022-04-29-update.rauc
rauc-Message: 19:36:54.898: Debug log domains: 'rauc'
(rauc:1535168): rauc-DEBUG: 19:36:54.903: Resolved bootname /dev/disk/by-uuid/05bc16f6-c339-4da3-99c0-28ede1409d9c to 3
(rauc:1535168): rauc-DEBUG: 19:36:54.903: bundle start
(rauc:1535168): rauc-DEBUG: 19:36:54.903: input directory: /home/quade/update/bundlefiles
(rauc:1535168): rauc-DEBUG: 19:36:54.903: output bundle: /home/quade/update/2022-04-29-update.rauc
(rauc:1535168): rauc-DEBUG: 19:36:56.187: Payload size: 8192 bytes.
Creating bundle in 'plain' format
rauc-Message: 19:36:56.198: No keyring given, skipping signature verification
(rauc:1535168): rauc-DEBUG: 19:36:56.198: Signature offset: 8192 bytes.
(rauc:1535168): rauc-DEBUG: 19:36:56.198: Bundle size: 18677 bytes.
quade@ezs-cocka:~/update$
```

4 **Im letzten Generierungsschritt erzeugen und signieren Sie das Update-Bundle.**

Listing 4: Pseudo-Image erstellen

```
$ mkdir -p ~/update/bundlefiles
$ cd ~/update/bundlefiles
$ truncate --size=64M rootfs.img
$ mkfs.ext4 rootfs.img
$ sudo mount rootfs.img /mnt
$ sudo touch /mnt/foo
$ sudo touch /mnt/bar
$ sudo touch /mnt/new_software
$ sudo sh -c "date > date"
$ sudo umount /mnt
$ cp rootfs.img appfs.img
```

Wie schon in der vorhergehenden Kern-Technik-Folge angerissen und in der RAUC-Dokumentation ausführlich erläutert, lassen sich im Manifest noch Handler und Hooks angeben. Als Handler bezeichnet man Skripte oder Programme, die sich schon auf dem Target befinden und die der Installationsprozess dort aufruft. Für Hooks gilt Gleiches, sie liegen jedoch dem Update-Bundle bei.

Nachdem Sie alles Notwendige im Verzeichnis zusammengestellt haben, starten Sie die Generierung des Bundles unter Angabe des kryptografischen Keys (Listing 6). Der Bundle-Generator erzeugt in einer Image-Datei ein Splash-Filesystem, kopiert sämtliche für das Update zusammengestellten Dateien hinein und signiert das Paket schließlich **4**.

Ortswechsel

Sobald Sie das Testsystem `Linux Magazin Test Device` via `qemu_start system` im Quellcodeverzeichnis starten, präpariert Qemu es seinerseits. Ist das erfolgreich

Listing 5: `manifest.raucm`

```
[update]
compatible=Linux Magazin Test Device
version=2022-04-29
[image.rootfs]
filename=rootfs.img
[image.appfs]
filename=appfs.img
```

Listing 6: Update-Bundle erstellen

```
$ cd ~/update/
$ rauc/rauc -d bundle --cert=ca/cert.pem --key=ca/key.pem bundlefiles/
2022-04-29-update.rauc
```

```
root@qemu-test:/home/quade/update# ./install.sh
rauc-Message: 19:21:50.283: Using central status file /tmp/rauc.status
input bundle: /tmp/2022-04-29-update.rauc
installing
  0% Installing
  0% Determining slot states
 20% Determining slot states done.
 20% Checking bundle
 20% Verifying signature
 40% Verifying signature done.
 40% Checking bundle done.
 40% Checking manifest contents
 60% Checking manifest contents done.
 60% Determining target install group
 80% Determining target install group done.
 80% Updating slots
 80% Checking slot rootfs.1
 85% Checking slot rootfs.1 done.
 85% Copying image to rootfs.1
 90% Copying image to rootfs.1 done.
 90% Checking slot appfs.1
 95% Checking slot appfs.1 done.
 95% Copying image to appfs.1
 99% Copying image to appfs.1 done.
 99% Updating slots done.
100% Installing done.
Installing `/tmp/2022-04-29-update.rauc` succeeded
root@qemu-test:/home/quade/update#
```

5 **Ein Installationsskript übernimmt das Einrichten des Updates.**

abgeschlossen, stößt das Hilfsprogramm RAUC als Service im Hintergrund an und bootet anschließend in eine Root-Shell.

Dank Virtio-Mount greift das Testsystem direkt auf die Dateien des Entwicklungsrechners zu, was das Problem des Transports der Konfigurationsdateien und des Update-Bundles schon beinahe löst. Beinahe, weil RAUC bezüglich Sicherheit erfreulich paranoid ist und erwartet, dass die Update-Datei dem User *root* gehört. Dazu muss das Update-Bundle in das Verzeichnis `/tmp/` kopiert sein.

Zudem ist der von `qemu-test-init` gestartete Update-Server mit der Konfiguration für den Pakettest und nicht mit der von Ihnen verwendeten Konfiguration angelaufen. Daher stoppen Sie ihn mit `killall rauc` und starten unter Angabe der passenden Gerätebeschreibung `system.conf` neu. Listing 7 zeigt das Skript `lm_install.sh`, das alle notwendigen Vorbereitungen übernimmt und dann das Update anstößt **5**.

RAUC wählt mithilfe des Bootloaders automatisch die redundante, inaktive Partition aus. Im Testsystem ist das die Partition respektive der Slot B. Auf Basis der Dateierweiterung der zum Slot gehörenden Update-Datei entscheidet RAUC, wie es das Update im Detail installiert. Bei einem Image (`.img`) kopiert es das Image per Dd in die redundante Partition.

Nach einem erfolgreichen Update weisen die beiden inaktiven, redundanten Partitionen einen neuen Inhalt auf. Um das zu überprüfen, hängen Sie die Partitionen per Mount-Kommando ein und sehen sich deren Inhalt an **6**.

RAUC aktualisiert die Konfiguration des Bootloaders, sodass dieser – gegebenenfalls mithilfe eines Watchdogs – das sichere Funktionieren des Systems unter allen Umständen garantieren kann. Das Qemu-Testsystem geht dabei von einem Grub-Bootloader aus und richtet aus diesem Grund beim Booten eine rudimentäre Boot-Partition ein.

Listing 7: `lm_install.sh`

```
#!/bin/bash
cd ~/update/
cp ca/cert.pem /tmp/rauc/ca.cert.pem
cp 2022-04-29-update.rauc /tmp/
killall rauc
rauc service --conf=system.conf 2>/tmp/service.log &
sleep 1
rauc install --keyring=ca/ca.crt --cert=ca/ca.crt /tmp/2022-04-29-
update.rauc
```

```
root@qemu-test:/home/quade/update# mount /tmp/rootdev /mnt
[  506.439591][  T855] ext4 filesystem being mounted at /mnt supports)
root@qemu-test:/home/quade/update# ls /mnt
bar  date  foo  lost+found
root@qemu-test:/home/quade/update# cat /mnt/date
Fr 29. Apr 15:43:58 CEST 2022
root@qemu-test:/home/quade/update# umount /mnt
```

6 **Wie der Zeitstempel zeigt, war das Update erfolgreich.**

Maschine baut Maschine

Im Industrieumfeld aktualisiert RAUC eingebettete Systeme, die häufig nicht auf einer x86-Architektur laufen. Deren Software lässt sich mithilfe eines System-Builders wie Yocto oder Buildroot generieren. Allerdings müssen Entwickler das Erstellen der Updates und Installieren auf dem Target noch in die eigenen Arbeitsprozesse integrieren. RAUC schlägt hier aus Sicherheitsgründen vor, mit unterschiedlichen Zertifikaten zu arbeiten. Das soll verhindern, aus Versehen für die Entwicklung generierte Updates auf Produktivgeräten auszurollen.

Fazit

Im Kontext von RAUC lauern Stolperfallen, die aus den Anforderungen hinsichtlich Sicherheit und Robustheit resultieren. Entsprechend gilt es, sich zunächst in das Werkzeug einzuarbeiten und damit zu experimentieren, bevor man es erfolgreich in eigene Projekte einbinden kann. Hinzu kommt, dass die Funktionen erheblich weiter reichen als in diesem Artikel angerissen. So unterstützt RAUC in der aktuellen Version das Streamen sowohl von Updates als auch Delta-Updates, um ressourcenschonend aufzutreten. Aktualisierungen mit sensiblen Informationen lassen sich verschlüsseln. Darüber hinaus ermöglicht RAUC ein Monitoring des Update-Vorgangs. (*csi*)

Dateien zum Artikel herunterladen unter
www.lm-online.de/dl/47348

Weitere Infos und interessante Links
www.lm-online.de/qr/47348