Android Firmware Over-the-Air Updates

Die Frage nach einem modernen Update-Mechanismus und Androids Antwort

Matthias Schaff, inovex GmbH,
Dominik Helleberg, inovex GmbH

Im Embedded Bereich gibt es kaum ein Lastenheft ohne die Anforderung, die Firmware eines linuxoiden, eingebetteten Systems im Feld aktualisieren zu können. Die Umsetzung hierfür ist vielfältig: vom Vor-Ort-Termin eines Technikers, über das Versenden von CF-Cards bis hin zur Selbst-Flash-Anleitung für den Kunden ist alles dabei. Im Zeitalter des IoT, Industrie 4.0 und der damit Einzug haltende allgemeinen Vernetzung der Komponenten erscheinen diese Vorgehensweisen geradezu antiquiert. Ein moderner Update-Mechanismus arbeitet voll automatisch während weder Sicherheit noch Stabilität des Prozesses darunter leiden.

Im Folgenden wollen wir neben den Anforderungen an solch einen Prozess, kurz verschiedene Update-Architekturen besprechen, um abschließen den Android FOTA Prozess im Detail zu beleuchten.

Anforderungen

Die dominierende Anforderung an einen FOTA Mechanismus ist Stabilität. Darunter fallen Punkte wie das Überstehen von Stromausfällen während des Update Vorgangs, und der Umgang mit fehlerhaften Update-Paketen (bedingt durch Übertragungs- oder Speicherfehler). Neben der Stabilität ist Sicherheit von zentraler Bedeutung: nur echte, vom Hersteller passend signierte FOTA Updates sollen akzeptiert werden, und der Vorgang soll keine Einfallstore für Angreifer öffnen.

Bei der Bewertung verschiedener Update-Architekturen ist zu beachten, dass die Komponenten eines eingebetteten Systems mit unterschiedlichen Update-Risiken verbunden sind. Ebenso ist die zu erwartende Update-Häufigkeit von Komponente zu Komponente unterschiedlich. Je näher sich eine Komponente an der Hardware befindet, desto seltener wird sie upgedatet, aber desto höher ist das Risiko. Jede Update-Architektur positioniert sich hierbei unterschiedlich in der Abwägung zwischen Häufigkeit und Risiko. Welcher Fokus dabei der Richtige ist, lässt sich nicht pauschal sagen, sondern ist von Fall zu Fall individuell zu entscheiden.

Gängige Update Architekturen

Im Folgenden sollen drei Update-Architekturen genauer dargestellt und deren Vor- und Nachteile kurz erläutert werden. In allen Fällen ist der Ablauf ähnlich: das System, welches aktualisiert werden soll, initiiert den Updateprozess, nachdem es erfolgreich ein Update-Paket heruntergeladen und ggf. verifiziert hat. Dabei wird in einem vom Bootloader zugänglichen persistenten Speicher des aktuell laufenden Systems eine Boot-Markierung (boot-flag) gesetzt und anschließend ein Reboot initiiert. Der Bootloader überprüft das boot-flag und startet ggf. den Update-Vorgang. Erst nach einem erfolgreichen Boot des upgedateten Systems wird das boot-flag wieder zurückgesetzt.

Der Bootloader als Updater

Der Bootloader agiert als eigenständiger Updater. Moderne Bootloader können auf block- oder flashbasierte Dateisysteme zugreifen und eignen sich daher als Updater.

Jedoch mangelt es in einer so eingeschränkten Arbeitsumgebung normalerweise an notwendigen Werkzeugen, um einen stabilen Updateprozess durchzuführen. Grundsätzlich kann diese Funktionalität komplett im Bootloader integriert werden, jedoch ist dabei zu beachten, dass der Bootloader nicht ohne Grund so minimalistisch gehalten ist: er ist ein elementarer Teil eines Systems, und jedes Feature ist eine potentielle Fehlerquelle, zudem ist ein Bootloader Update risikoreich. Daher sollte man davon absehen weitere Komplexität in den Bootloader zu bringen, und stattdessen das eigentliche Update einem dafür besser geeignetem Systemteil überlassen.

OS-Switching Architektur
Das eingebettete System enthält zwei komplett funktionsfähige Betriebssysteme, inklusive aller Devicetreiber und jeweils einem Kernel. Es ist je nur eine Version des Betriebssystems gebootet, und diese kann damit das aktuell inaktive System aktualisieren. Das boot-flag zeigt dabei an, welches der beiden Systeme gebootet werden soll. Erst nach einem erfolgreichen Flashvorgang und dem erfolgreichen Abgleich der Prüfsumme wird das boot-flag getoggelt. Damit ist stets eine funktionsfähige Systemversion vorhanden. Im Fehlerfall kann jederzeit wieder auf die Vorgängerversion zurückgewechselt werden. Das System ist immer im normalen Betriebsmodus, die Ausfallzeiten durch das Update sind minimal und der Bedarf an Speicherplatz verdoppelt sich.

Recovery OS Architektur
Diese Architektur ähnelt der "OS-Switching" Architektur, mit dem Unterschied, dass nur ein System über den vollen Funktionsumfang verfügt, das Zweite dient nur dem Updatevorgang. Dazu beinhaltet es einen eigenen Kernel und alle Device Treiber, aber nur eine kleine Anzahl Userland Tools, die Funktionalitäten wie flashen und verifizieren abdecken. Meist wird das Recovery OS wird vom Haupt OS aktualisiert, welches ebenfalls die notwendigen Tools beinhaltet.

Der Android Updateprozess
Der Android Updateprozess basiert auf der zuletzt vorgestellten "Recovery-OS" Architektur. Dabei lässt sich der gesamte Prozess in vier Stufen aufteilen, die untereinander eng verzahnt sind und ineinander greifen. Das Ergebnis ist ein sehr robuster, sicherer und flexibler Updateprozess, der sich in der Praxis schon millionenfach bewährt hat.

Der Android Updateprozess ist beständig gegen Übertragungsfehler und Unterbrechungen und stellt die Integrität und Authentizität von Update-Paketen sicher. Zudem sind die Update-Pakete auf minimale Größe getrimmt, um die zu übertragende Datenmenge so klein wie möglich zu halten (Delta-Updates). Die genaue Umsetzung wollen wir im Folgenden betrachten.

Das Android Update-Paket
Das Paketformat für ein Android Update ist ein gewöhnliches, digital signiertes ZIP Archiv. Die Partitionen und Verzeichnisstrukturen des Android Systems finden sich auch im Update-Paket wieder: das "recovery"-Verzeichnis enthält die Updates des Recovery-OS, das "system"-Verzeichnis analog alles für das Android-OS.

Kernel und "root" Dateisystem des Android-OS werden als "boot.img" ausgeliefert. Da sich Recovery-OS und "root" Dateisystem des Android-OS nur minimal unterscheiden, wird das Recovery-OS als Binär-Diff aus dem boot.img generiert.

Das Verzeichnis “META-INF” enthält neben den Signaturinformationen, sowie dem Public-Key zur Verifikation, eine Datei namens “updater-script”. Dieses Script beschreibt in Androids “edify” Scriptspache jeden der einzelnen Schritte des Updates. “edify” wurde extra für Update Aufgaben geschaffen, um, unter Anderem, ein einfaches Mounten, Formatieren, Kopieren und Patchen von Partitionen bzw. Dateien zu ermöglichen.

Als letzter relevanter Teil des Update-Pakets ist die Binary “update-binary” zu nennen. Diese ist der Script Interpreter, welcher das edify Script ausführt. Dieser wird von Recovery-OS gestartet und führt das eigentliche Update aus. Durch diesen Aufbau ist das Update-Paket an keine bestimmte Recovery-OS Version gebunden.

Das Update-Paket ist ein Build Artefakt aus dem Android AOSP Build-System und wird vollkommen automatisch erzeugt. Für gerätespezifische Anpassungen verfügt das Build-System über entsprechende Schnittstellen für Plugins. Bei der Erstellung eines Update-Pakets kann einen Ausgangsversion mit angegeben werden. So kann das Build-System ein inkrementelles Update erzeugen, welches nur Unterschiede zur Ausgangsversion enthält. In diesem Fall wird pro Datei nur ein Binär-Diff zur entsprechenden Vorgängerversion als Patch im Update-Paket gespeichert, was gerade bei kleinen Bugfix Firmware Updates die Dateigrößen extrem reduziert.

Der Android Updateprozess in Schritten

Teil 1 Android-OS: Herunterladen, verifizieren, neu starten

Android macht keine Vorgaben wie das Update-Paket auf das System kommt. Im Normalfall überprüft eine App, bei bestehender Internetverbindung, ob ein neues Update-Paket auf einem zuvor konfiguriertem Update-Server verfügbar ist. Ist das der Fall, so wird es nach einer expliziten Bestätigung des Users heruntergeladen. Da es sich nur um ein Zip-Archiv handelt, kann es sowohl via USB als auch über eine SD-Karte auf das System gebracht werden.
Anschließend wird die digitale Signatur des Update-Pakets mittels `RecoverySystem.verifyPackage()` kontrolliert, bevor die Installation startet. Die Signatur wird gegen die OTA-Public-Keys[1], welche im Android-OS hinterlegt sind, überprüft. Folglich kann man nur ein gültiges Update-Paket erzeugen, wenn man über die entsprechenden Private-Keys verfügt.

Bei passender Signatur wird mittels `RecoverySystem.installPackage()` der Installationsvorgang gestartet. Dieser Aufruf erzeugt zunächst eine command - Datei[2] welche die vom Recovery-OS auszuführenden Aktionen enthält. In unserem Falle wäre der Inhalt beispielsweise “ --update-package=/data/update.zip”.

Im letzten Schritt des ersten Teils wird mit `PowerManager.reboot(PowerManager.REBOOT_RECOVERY)` das boot-flag gesetzt und ein Reboot ins Recovery-OS angestoßen. Das boot-flag selbst wird erst beim Kernel-Aufruf für den Reboot gesetzt.

Teil 2 Recovery-OS: verifizieren, entpacken, ausführen

Beim Bootvorgang wertet der Bootloader das boot-flag aus, und bootet entsprechend das Recovery-OS welches am Ende die “recovery” Binary startet. Diese verifiziert nun die Signatur des Update-Pakets erneut, nun jedoch gegen die Recovery-OS ge-

speicherten Public-Keys[3]. Bei Erfolg wird nun die update-binary aus dem Update-Paket entpackt und ausgeführt.

Teil 3 update-binary: das Update einspielen
Die update-binary beginnt nun mit der Interpretation des update-script[4] und führt die somit eigentliche Aktualisierung des Android-OS aus. Im Falle eines inkrementellen Updates wird die zu patchende Datei zunächst kopiert, dann gepatched und eine Prüfsumme gebildet. Nur wenn diese mit der im update-script angegebenen Prüfsummer übereinstimmt, wird die gepatchte Datei zurückgeschrieben. Da sich das Recovery-OS nicht selbst im laufenden Betrieb flashen kann, wird hier der Patch, welcher das Recovery-OS aus dem boot.img erzeugt sowie ein Shellscript[5], welches diesen Patch anwendet, auf die Systempartition des Android-OS geschrieben.

Nach erfolgreichem Update geht der Kontrollfluss zurück zur recovery Binary, welche das boot-flag wieder zurücksetzt und einen Reboot initiiert.

Teil 4 Android-OS: recovery updaten und aufräumen
Der Bootloader startet nun das aktualisierte Android-OS zum ersten Mal. Während des Init-Prozesses wird nun das im Recovery-OS kopierte Shellscript zur Installation des Recovery-OS ausgeführt. Dieses berechnet die Prüfsumme der Recovery-OS Partition. Sollte diese nicht dem neuen Recovery-OS entsprechen, so wird es nun aktualisiert. Abschließend wird das Update-Paket selbst gelöscht, und das System kann wieder seinen normalen Betrieb aufnehmen.

Quellen

[1] siehe /system/etc/security/otacerts.zip
[2] siehe /cache/recovery/command
[3] siehe /res/keys
[4] siehe im Update Paket unter META-INF/com/google/android/update-binary
[5] siehe /system/etc/install-recovery.sh

Autoren

Matthias Schaff arbeitet bei der inovex GmbH im Bereich Embedded Android Development am Standort Karlsruhe. Seine Schwerpunkte liegen im Bereich System Entwicklung, Build System und des Updateprozesses.

Dominik Helleberg leitet bei der inovex GmbH den Bereich “Mobile Development”. Neben der klassischen Applikationsentwicklung im Enterprise Kontext werden hier Embedded Lösungen auf Android Basis entwickelt.