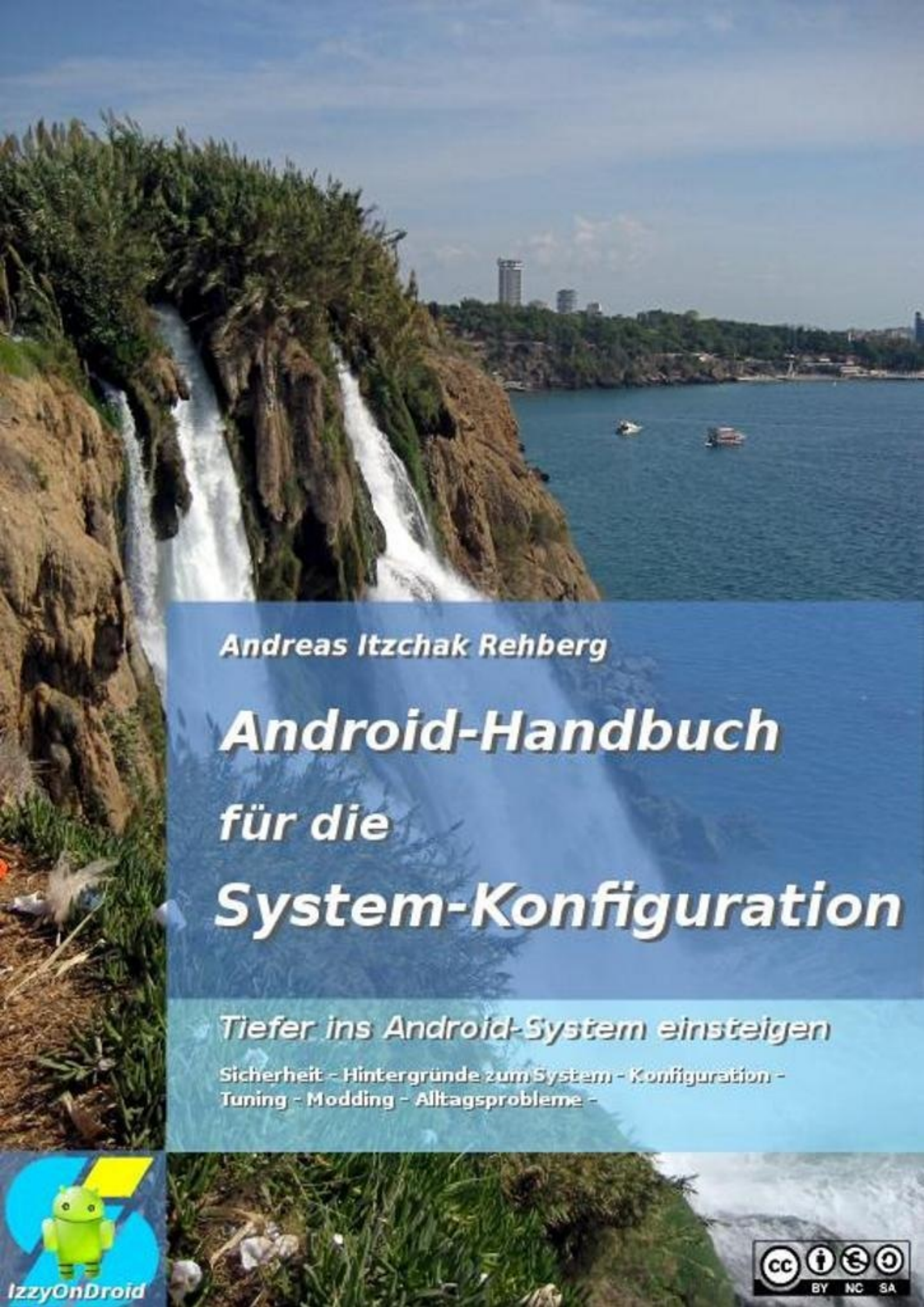


Andreas Itzchak Rehberg

Android-Handbuch für die System-Konfiguration

Tiefer ins Android-System einsteigen

Sicherheit - Hintergründe zum System - Konfiguration - Tuning - Modding - Alltagsprobleme -

IzzyOnDroid

cc BY NC SA

Android Handbuch für die Systemkonfiguration

Kleiner Führer durch das Android-System und seine Einstellungen

von
Andreas Itzchak Rehberg

Cover-Design: Izzy
(Foto: Wasserfall in Antalya, © Izzy 2014)

Inhalt

6. Leistungsaufnahme verschiedener Komponenten

EINLEITUNG

Vorwort

Das Android Handbuch für Anwender erfreut sich nach wie vor großer Beliebtheit, wie Ranking und Bewertungen bei Amazon zur nunmehr vierten Auflage (innerhalb von nur zweieinhalb Jahren) zeigen. Daher folgte ihm als zweiter Band das eBook Android Handbuch für Reiselustige, welches ebenfalls begeistert angenommen und mit Rufen nach „mehr“ belohnt wurde. Was blieb mir da anderes übrig, als für eine Fortsetzung zu sorgen?

Was diese zum Thema hat, lässt sich auch bereits am Namen des Buches erkennen: Das Android System, seine Einstellungen und seine Verwaltung. Was steht dafür an Bordmitteln bereit, und wo macht der Griff zu Apps von Drittanbietern Sinn? Was ist überhaupt möglich, und wo sind die einzelnen Einstellungen versteckt? Viele dieser Themen wurden bereits im ersten Buch angesprochen, aber meist nur kurz. Den meisten mag das ausgereicht haben; doch wer gern tiefer einsteigen und mehr Details wissen möchte, findet beides in diesem eBook. Auch Praxis-Tipps und Hintergründe sollen nicht fehlen.

Natürlich ist es dennoch unmöglich, auf jedes Detail und jede App bis in den letzten Punkt einzugehen – schon gar nicht, ohne dass es langweilig wird oder man den Faden verliert. Wie aus meinen vorigen Büchern gewohnt, möchte ich auch in diesem Buch das nötige Wissen wieder auf kurzweilige Art übermitteln. Ein eBook darf beim Blättern schließlich nicht stauben! Daher wird, wie gehabt, an vielen Stellen auf die thematisch sortierten App-Übersichten bei *IzzyOnDroid* Bezug genommen – wo sich weitere Details, Artikel, App-Auflistungen sowie -Vorstellungen und weitere Informationen finden.

An einigen Stellen wird auf eine „Franzis-Edition“ dieses Buches hingewiesen, die „weitere Details“ erhält – was den Einen oder die Andere Leser(in) verwundern könnte. Der Hintergrund ist jedoch leicht erklärt: Dieses zusätzliche Material entstammt einem Buch, an dem die Rechte für die deutsche Version beim Verlag liegen (*Android Forensics and mobile Security* von Andrew Hoog). Die genannten Inhalte wurden mir unter der Bedingung zur Verfügung gestellt, dass entsprechend detaillierte Ausführungen diesbezüglich exklusiv der Verlgas-Version vorbehalten bleiben. Deal ist Deal 😉

Und noch eins muss ich loswerden: Viele der hier kurz vorgestellten (oder auch nur genannten) Apps habe ich selbst nie getestet – dafür fehlt einfach die Zeit: Würde ich jede App erst selbst testen, wäre ich noch immer mit dem ersten Buch beschäftigt. Und wäre das dann fertig, könnte ich gleich wieder von vorn beginnen – einfach weil die Informationen bis dahin bereits veraltet wären. Ein Grund mehr, auf die Erfahrungen der Community zurückzugreifen.

Doch nun: Viel Spaß bei der Lektüre!

Hinweise zur Nutzung des Buches

Wie bereits im Vorwort aufgeführt, handelt es sich bei diesem Buch um die zweite Fortsetzung meines *inoffiziellen Android-Handbuchs*. Ausgewählte Themen sollen hier vertieft werden. Für andere Dinge setze ich teilweise Kenntnisse voraus, die im „ersten Band“ behandelt wurden.

Sofern der verwendete eBook-Reader Farben und StyleSheets korrekt unterstützt, lassen sich verschiedene Arten von Links an ihrer Textfarbe erkennen: Rote gehen „nach draußen“ (öffnen also wahrscheinlich Deinen Web-Browser), während grüne auf Begriffserklärungen im Anhang verweisen, und auch blaue „drinnen bleiben“ (also Querverweise innerhalb dieses eBooks sind). Desweiteren sind Ausführungen, die root voraussetzen, entsprechend farblich hinterlegt.

Ein weiterer Hinweis gilt den Screenshots: Diese sind oft der Playstore-Seite der jeweiligen App entnommen, und daher häufig auf Englisch. Das muss allerdings nicht zwangsläufig heißen, dass selbige App dann nicht auch auf Deutsch läuft – meist ist das der Fall. Es belegt also lediglich, dass sie auch Englisch kann.

SICHERHEIT

Leider haben wir im Deutschen dafür wohl nur das eine Wort – wo andere Sprachen, wie etwa das Englische, durchaus zu unterscheiden wissen. So hat jeder gewiss schon den Satz gehört: „Safety first!“. An was denkt man dabei zuerst? Wahrscheinlich an Werbung für Präservative. Gutes Beispiel: Diese „Gummi-Tütchen“ sollen u. a. vor unerwünschter Schwangerschaft oder der Ansteckung mit AIDS schützen. Doch verrät das gut sichtbare „Leucht-Kondom“ im stockdunklen Schlafzimmer der Frau im Bett noch lange nicht, ob der nackte Typ da neben ihr wirklich der eigene Partner – oder vielleicht ein wildfremder Mann ist. Das wäre dann nämlich Security…

Also gilt es bei dem Wort „Sicherheit“ sehr wohl zu unterscheiden, was man denn nun eigentlich meint. Und was ich hier meine, wenn ich diese Begriffe in die Tastatur hacke, soll folgende Grafik veranschaulichen – die damit gleich nebenbei die Struktur dieses Teils abbildet:

Der Begriff „Sicherheit“ hat verschiedene Bedeutungen

Safety

Schrieb ich da gerade „Safety first“? Nun, dann soll das auch für dieses Buch gelten. Mit „Safety“ meine ich: Schutz vor Datenverlust. Schützt nicht vor dem Verlust des Androiden selbst: Genügend „Kleingeld“ vorausgesetzt, lässt sich dieser noch relativ einfach ersetzen. Doch die persönlichen Daten, die gibt es nicht im Laden: Termine, Adressen, Fotos … Wo bekommt man die wieder her, wenn irgend so ein Depp die aus *Spaceballs* bekannte Ansage provoziert hat: „Vielen Dank, dass Sie den Selbstzerstörungsmechanismus aktiviert haben … 3…2…1 … Tschüss!“?

Lokale Backups

Ah, aus dem Backup? Werfen wir zunächst einen Blick in „Murphys Gesetze“. Da heißt es unter anderem:

> ***Backup:*** *Etwas, das man nicht braucht, wenn man es hat – oder nicht hat, wenn man es braucht. Hat man es, wenn man es braucht, ist es entweder nicht*

lesbar – oder zumindest der Sektor mit der wichtigsten Datei kaputt.

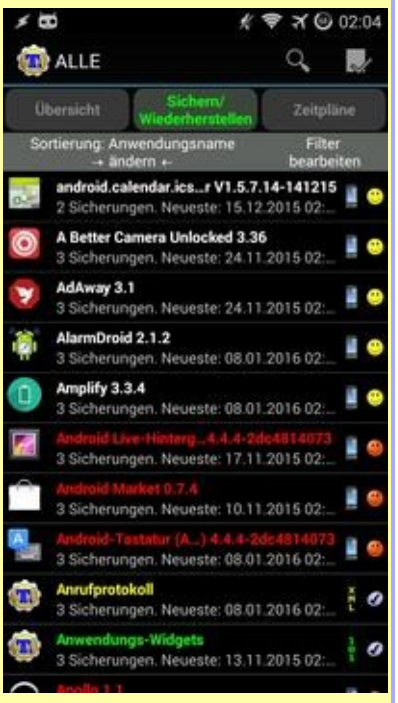
Damit einem dieses Schicksal erspart bleibt, kann man mit Netz, doppeltem Boden und Sicherheitsleine arbeiten: Ein Backup auf der SD-Karte, eine Kopie davon auf dem PC, und eine weitere außerhalb der eigenen vier Wände. Vielleicht nicht unbedingt im Bank-Tresor, aber auf einem fremden Server sollte es dann auch verschlüsselt sein. Nicht nur, um es vor unbefugten (Ein-) Blicken, sondern auch vor etwaiger Fremd-Modifikation zu schützen. Wie weit man das ganze jetzt treibt, bleibt natürlich jedem selbst überlassen.

Die nötigen Werkzeuge werde ich jedoch hier nennen. Ich muss dabei zugeben, dass ich in Sachen Datenschutz vielleicht ein wenig paranoid bin: So bevorzuge ich Lösungen, die ohne Netzwerk-Permissions auskommen. Sollen die Daten dann doch irgendwo außerhalb meines Androiden abgelegt werden, lasse ich das lieber von einer separaten App erledigen. Dieser Sachverhalt wird sich hier wohl ein wenig widerspiegeln.

Natürlich werde ich an dieser Stelle nicht auf alle verfügbaren Apps eingehen – dafür sind es einfach zu viele. Eine entsprechende Übersicht findet sich jedoch bei *IzzyOnDroid*.

Komplettsicherungen

Titanium Backup



Fragt man nach einer guten Backup-App, taucht zweifellos Titanium Backup bei den Antworten auf. Aus meiner Sicht ist dies die wirklich beste Lösung, sie hat nur einen Haken: Man benötigt dafür root-Rechte. Bei genauerem Nachdenken ist das logisch – wie sonst will man wirklich *überall* rankommen? Und *Titanium Backup* kann wirklich alles sichern, System-Apps eingeschlossen. Letztere lassen sich damit sogar „einfrieren“ oder, für ganz Harte, vollständig entfernen: So wird man die ganzen „Zwangsbeglückungen“ los, die einem Hersteller (und, bei Geräten mit Branding, auch die Provider) auf’s Auge drücken – die aber kaum jemand braucht.

Die Erläuterung aller Features dieser App würde beinahe für ein eigenes Buch ausreichen, also muss ich mich auf das Wesentlichste beschränken. Ein vollständiges Backup einschließlich aller Apps und ihrer Daten bedarf eigentlich keiner gesonderten Erwähnung – das kann man mit Fug und Recht erwarten (sonst wäre es ja kein Komplett-Backup; die Sicherung erfolgt übrigens wahlweise auf SD-Karte oder auch in die Dropbox). Gleiches gilt für die Wiederherstellung – denn was nützt

ein Backup ohne diese Möglichkeit? Aber die zahlreichen Kleinigkeiten sind es, die *Titanium Backup* so einzigartig machen.

So lässt sich festlegen, wie viele „Generationen“ man denn aufheben möchte – damit man bei Bedarf auch auf eine ältere Sicherung zurückgreifen kann. Eingangs erwähntem „Murphy“ zufolge stellt man einen Fehler nämlich erst dann fest, wenn er schon ins Backup übernommen wurde. Oder der Market-Doktor: Taucht eine App im Playstore nicht mehr unter den eigenen Apps auf? Kein Thema: Der Market-Doktor repariert das. Hat ein Update eine App unbrauchbar gemacht? Na, dann holen wir uns aus einer “vorigen Generation” eine ältere Version zurück. Versehentlich in einer App zu viele Daten gelöscht? Die sind im Backup ebenfalls enthalten. Das Gerät wird immer langsamer? Dann lassen wir doch mal den Dalvik-Cache ein wenig aufräumen.

Und da wir ja alle ein wenig vergesslich sind – aber dennoch im Notfall gern ein aktuelles Backup hätten: Mit *Titanium Backup* lassen sich auch automatische Backups erstellen.

Für weitere Details etwa bezüglich Dropbox-Support, der Nutzung von Labeln, dem Wiederherstellen von Backups nach einem Systemupdate, oder gar dem Übertragen der Datensicherung auf ein neues Gerät, empfiehlt sich ein Besuch auf der Homepage. Dort steht u. a. auch ein hilfreiches Wiki bereit.

Nandroid Backup

Eine weitere Art des Komplett-Backups führt über die sogenannte Recovery-Konsole – allerdings in der erweiterten Version, wie sie Custom Recoveries bereitstellen. Am bekanntesten ist in diesem Umfeld ClockworkMod (kurz: CWM; siehe Screenshot) von Koushik Dutta (besser bekannt als „Koush“). Für dieses findet sich im Netz sogar ein (zwar leicht veralteter, aber dennoch sehr nützlicher) englisch-sprachiger Guide.

Anders als bei *Titanium Backup* werden hier keine einzelnen Dateien gesichert, sondern ein Image der gesamten Partitionen angelegt – wobei neuere Versionen von einigen Partitionen auch einfach Tar-Archive erzeugen. Für Windows-User lässt sich dies vielleicht mit Norton Ghost vergleichen – eingefleischten Linux-/Unix-Nutzern reichen als Beschreibung die beiden Buchstaben `dd`. Aus jeder Partition wird somit eine Datei auf der SD-Karte, die sich dann natürlich auch zur weiteren Sicherheit auf den Computer kopieren lässt.

Sinnvollerweise erstellt man in regelmäßigen Abständen – und insbesondere vor größeren Änderungen – eine solche Komplettsicherung, beispielsweise vor der Installation eines neuen ROM-Images. Geht dann etwas schief, lässt sich der

„lauffähige Zustand“ recht einfach wieder herstellen – indem man die erstellten Images über das Recovery-Menü wieder herstellt.

Und nun bringen wir die beiden genannten Kandidaten zusammen: Interessanterweise kann *Titanium Backup* nämlich auch einzelne Daten (etwa eine App inklusive ihrer Einstellungen) aus Nandroid-Backups wiederherstellen. Somit lassen sich ggf. kleinere „Fehler“ wieder korrigieren – wenn man etwa vor dem Einspielen eines neuen Custom-ROMs die Daten einer App zu sichern vergessen hat. Oder sich erinnert, dass ein älteres Nandroid-Backup noch etwas enthält, was man auf dem neuen Gerät nicht mehr hat – wobei übrigens alternativ auch der AppExtractor helfen kann.

Wer so ein Nandroid-Backup für eine interessante Sache hält – es aber als zu umständlich empfindet, dafür jedes Mal ins Recovery-Menü zu booten (da ist man ja offline! Man könnte eine wichtige Facebook-Meldung verpassen!) – der sollte einmal bei den XDA-Developers vorbei schauen. Dort hat nämlich jemand ein Tool gebastelt, welches Nandroid-Backups im laufenden Betrieb erstellt. Also so ganz ohne Booten und ohne Facebook-Ausfall. Ähnliches leistet übrigens auch die Companion-App ROM Manager (siehe ClockworkMod im Modding-Bereich dieses Buches).

Komplett-Backups ohne root

Vom Androiden aus sind diese eigentlich unmöglich: Das Berechtigungs-System lässt es bekanntlich nicht zu, dass eine App auf die Daten einer anderen direkt zugreift, ohne dass letztere diese explizit freigegeben hat – was in der Regel ja nicht der Fall ist. Wie aber will man ein solches auf anderem Wege erreichen?

Mit Android 4.0 (auch als *Ice Cream Sandwich* bekannt) wurde, nahezu undokumentiert, eine solche Möglichkeit geschaffen: Der auf dem Gerät laufende ADB-Daemon hat hier zusätzliche Rechte erhalten. Nutzen lassen sich diese für ein Backup allerdings nur, wenn auf einem per USB-Kabel angeschlossenen Computer auch ein entsprechender Client zur Verfügung steht. Dies trifft zu, sofern man das SDK (oder zumindest die notwendigen Teile davon) dort installiert hat. Dann nämlich kann man den Befehl `adb backup` an der Kommandozeile absetzen:

```
adb backup [-f <file>] [-apk|-noapk] [-shared|-noshared] [-all] [-
system|nosystem] [<packages…>]
```

Die erwähnten Optionen stehen dabei für folgendes:

- `-f` – Pfad und Name der auf dem Computer abzulegenden `*.ab` Datei. Bei dieser handelt es sich um eine komprimierte Archiv-Datei, welche die Apps und Daten des Android-Geräts enthält.

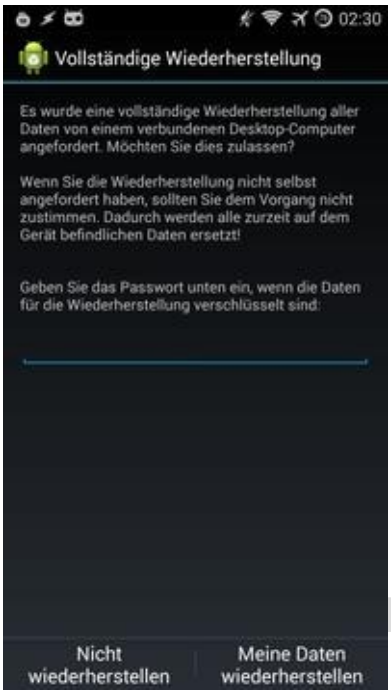
- `-apk|-noapk` – gibt an, ob die `*.apk` Dateien (Apps) mit verarbeitet werden sollen. Default ist `-noapk`.
- `-shared|-noshared` – Aktivieren/Deaktivieren des Backups von Inhalten des “externen Speichers” (“shared storage”, SD-Karte). Default ist hier `-noshared`
- `-all` – gibt an, dass das gesamte System gesichert werden soll (Komplett-Backup). Alternativ lassen sich mit dem Filter `<packages>` auch einzelne (oder mehrere) Apps sichern.
- `-system|-nosystem` – spezifiziert, ob System-Apps (und ihre Daten) im Backup enthalten sein sollen. Default ist `-system`.
- `<packages>` – hier lassen sich einzelne Pakete (Apps) zur Sicherung angeben, wenn kein Komplett-Backup erfolgen soll. Bei Verwendung der Option `-all` wird dies nicht benötigt.

Ein Beispiel-Aufruf könnte wie folgt aussehen:

```
adb backup -apk -shared -all -f ~/backup/Android_2016-01-10.ab
```

Oder, für Windows-Anwender:

```
adb backup -apk -shared -all -f C:\backup20160110.ab
```



Auf dem Gerät löst dieser Befehl sodann einen Hinweis aus, wie er im Screenshot zu sehen ist. Der Cursor zeigt bereits an: Hier wird eine Benutzer-Eingabe erwartet. Und zwar soll das Passwort angegeben werden, mit dem das Backup sodann verschlüsselt wird; wir haben ja bereits gelernt: Das Wort „Sicherheit“ hat zwei Bedeutungen. Durch das eigentliche Backup existiert eine Kopie, die vor Datenverlust schützen soll – und durch die Verschlüsselung wird Unbefugten der Zugang zu diesen Daten erschwert.

Analog geht es in der Gegenrichtung zu, wenn ein Backup wieder hergestellt werden soll:

```
adb restore ~/backup/Android_2016-01-10.ab
```

Oder, für Windows-Anwender:

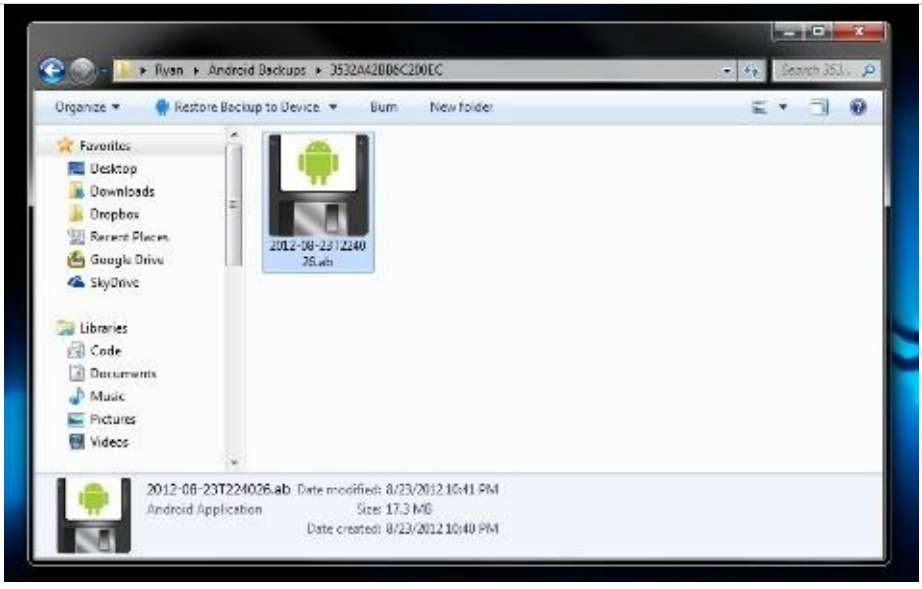
```
adb restore C:\backup20160110.ab
```

Auch hier erfolgt wieder die Abfrage des Passwortes: Das Backup wurde ja verschlüsselt, und muss somit zur Wiederherstellung auch entschlüsselt werden (man kann das Passwort natürlich beim Backup und sodann auch beim Restore

einfach weglassen, will man auf die Verschlüsselung verzichten). Wer beim Wiederherstellungs-Befehl genauer hinschaut, hat es sicher bereits gesehen: Dieser kennt als einzigen Parameter die Datei des wiederherzustellenden Backups. Einzelne Daten lassen sich also scheinbar nicht aus einem vollständigen Backup direkt auf das Gerät zurücktransferrieren.

Aber geht das wirklich nur über die Kommandozeile? Und von Hand? Die XDA-Developer haben sich da natürlich bereits etwas einfallen lassen. Und so berichtet ein dortiger Artikel vom Ultimate Backup Tool. Natürlich für Windows. Mittlerweile aber auch für den Mac, wo es jedoch ein installiertes SDK voraussetzt. Da es sich hier um ein Shell-Script handelt, sollte Linux-Fans eine Anpassung nicht zu schwer fallen.

Was aber tun diejenigen, die viel lieber die Maus schubsen würden? Die greifen zum Droid Explorer. Ryan (dem ich an dieser Stelle für den Hinweis auf diese Möglichkeit sowie auch für seine Ausführungen danken möchte) hat nämlich sein Tool, welches sich sonst hauptsächlich an Wurzelmenschen richtet, um dieses Features auch für nicht-gerootete Geräte erweitert. Weitere Tools, die dies unterstützen, sind übrigens auch bei den Power-User-Tools zur Verwaltung des Androiden vom PC beschrieben. Auch mein eigenes kleines Tool namens Adebar möchte ich an dieser Stelle nennen. Zwar arbeitet auch dieses an der Kommandozeile, vereinfacht aber vieles.



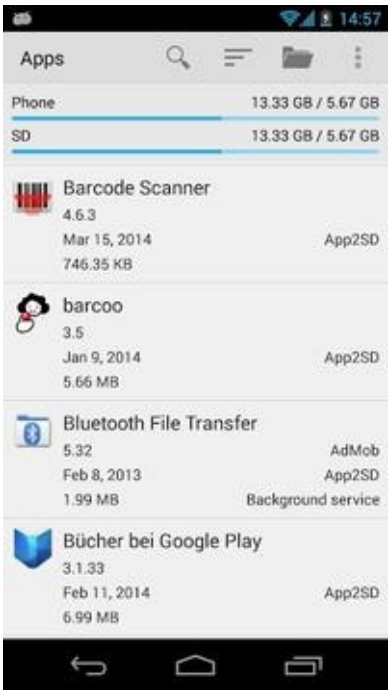
Droid Explorer beim Backup

Eine echte Alternative zu finden, die sich direkt auf dem Androiden ausführen lässt, und dabei ebenfalls ohne *root* auskommt, scheint unmöglich. Nicht einmal wirklich nah heran kommt man da, wenn es etwas gutes sein soll. Die Bewertungen sind bestenfalls ziemlich durchwachsen, wie auch die Erfahrungen der Benutzer. Erschwerend kommen die Zugriffsrechte hinzu: Natürlich soll alles gesichert werden, auch Kontakte und Termine. Aber doch nicht das Internet …

MyBackup Pro kommt in den Kommentaren noch recht gut weg (anders als die zugehörige Testversion). Die App soll ebenfalls ein komplettes Backup auf der SD-Karte anlegen, und natürlich von dort wiederherstellen können. Explizit erwähnt wird auch die Möglichkeit, auf diese Weise die Daten auf einem anderen Gerät wieder einzuspielen.

Da die App aber auch gern die Backups auf den Online-Servern des Anbieters speichern möchte, ist ein entsprechender Vertrauensvorschuss nötig: Die Kombination aus Zugriff auf Internet *und* persönliche Daten ist immer ein wenig heikel. Der Preis für diese App liegt mit knapp 4 Euro (eine gratis Testversion ist ebenfalls im Playstore verfügbar) leicht unter dem von *Titanium Backup*. Anwendern mit root-Zugriff würde ich dennoch eher letzteres empfehlen.

Apps sichern

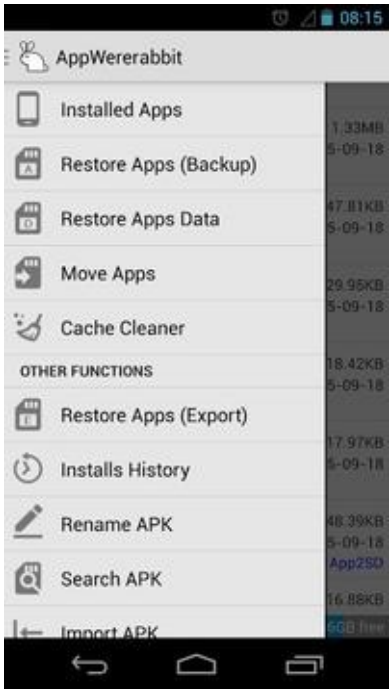


Dafür sieht es in diesem Bereich auch für Anwender ohne *root* recht gut aus – denn hier gibt es das AppMonster. Dieses schlägt in der Pro-Version mit knapp drei Euro zu Buche – hat mir aber mehr als einmal den Tag gerettet.

Die Hauptfunktionalität dieser App besteht darin, bei jeder Neuinstallation automatisch eine Kopie der APK-Datei auf der Karte abzulegen – was sich natürlich auch manuell erledigen lässt. Erweist sich ein Update im Nachhinein als nachteilig, kann man so mit Leichtigkeit auf eine vorige Version zurückgreifen. Gleiches gilt nach einem Factory-Reset: Auch wenn die Lieblings-App aus dem Playstore verschwunden sein sollte, das Monsterchen hat eine Kopie aufgehoben.

Da kann man es fast als „nette Dreingabe“ betrachten, dass so einige Zusatzinformationen in der Appliste angezeigt werden: Etwa ob eine App mit

Werbemodul (z. B. AdMob) daherkommt – was dann auch die Berechtigung für den Internet-Zugriff erklären könnte. Oder ob App2SD unterstützt wird. Des Weiteren hat man auch direkt Zugriff auf die Playstore-Seite der jeweiligen App – oder kann sie über Facebook, Twitter, oder auch per Mail weiterempfehlen. Oder ganz anders herum: Sie schnell deinstallieren. Verfügt man auf dem Gerät über root Rechte, kann sich das Monsterchen auch um die Daten der Apps kümmern.



Fairerweise sei hier noch auf die Einschränkungen der kostenlosen Version hingewiesen: Keine automatische Sicherung der Neuinstallationen, kein Batch-Restore, und auch die erweiterten Informationen (wie App2SD) werden hier nicht angezeigt. Backups lassen sich aber manuell erstellen und auch wiederherstellen. Aus eigener Erfahrung kann ich den Kauf der Vollversion jedoch wärmstens empfehlen!

Was *Titanium Backup* im Komplett-Bereich, ist *AppMonster* hier: Mir ist keine App bekannt, die dem Monsterchen das Wasser reichen könnte. Alles andere kommt allenfalls „recht nah heran“ – wie etwa das rechts abgebildete AppWhereRabbit, welches eine Toolbox zur App-Verwaltung darstellt. Vergleichbar mit der Gratis-Version vom *AppMonster* lassen sich Apps manuell sichern sowie auf die SD-Karte verschieben, ebenso ist ein APK-Installer mit an Board, und auf gerooteten Geräten können die App-Daten mit gesichert werden. Darüber hinaus bietet *AppWhereRabbit* Tools zur Cache-Bereinigung, und einige weitere Dinge.

Keine der anderen mir bekannten Apps in diesem Sektor unterstützt – wie die beiden genannten Kandidaten – das simultane Speichern mehrerer Versionen (sodass man bei Bedarf zu einer älteren Version zurückkehren kann). Dies halte ich jedoch hier für essentiell: Für alles andere habe ich ja, Dank *Titanium*, mein Vollbackup. Anwender ohne *root* sehen das vielleicht ein wenig anders: Für diesen Fall gibt es ja die bereits genannte Übersicht.

Ein spezieller Backup-und-Restore-Wunsch tritt meist dann ein, wenn das Amazon-Päckchen mit dem lang ersehnten neuen Androiden eingetroffen ist. Dann wird ausgepackt, herum gespielt, gerootet, die Konfiguration durchgeschaut, das Eine oder Andere ausprobiert … Alles selbstverständlich, während der „Alte“ noch in Benutzung ist.

Und dann ist es soweit: „Da war doch noch was …“ Genau: Wie kommen jetzt die Apps von A nach B? Klar, mit *Titanium Backup*. Also auf dem „Alten“ alles sichern, von der Karte auf den Rechner kopieren, vom Rechner auf die Karte des „Neuen“ … Besonders umständlich wird es dann, wenn beide Geräte noch länger parallel laufen müssen.

Aber warum schreibe ich das alles so dramatisch, wenn ich nicht mal wieder etwas anbieten möchte? Es muss also offensichtlich eine einfachere Lösung geben. Die trägt den Namen Helium Backup, und kann auch für normale App-Backups (inklusive der App-Daten) verwendet werden. Ohne root bedarf es dafür einer Aktivierung via USB vom PC (jeweils nach jedem Gerätestart erneut). Erwirbt man für knapp vier Euro die Premium-Version, lassen sich Apps und deren Daten auch direkt zwischen zwei Geräten austauschen (synchronisieren).

Warum soll man sich dabei auf den „Umzug“ beschränken? Da drängen sich förmlich weitere Anwendungsgebiete auf: Unterwegs in der S-Bahn mit einer App auf dem Smartphone gewerkelt – und bei Ankunft zu Hause den aktuellen Stand auf’s Tablet übertragen, wo es dann bequemer weitergehen kann. Oder morgens noch schnell den Spielstand von Tablet auf das Smartphone schieben – damit man damit in der U-Bahn weiter daddelt, ohne dass der Spielstand leidet.

Kontakte, SMS & Co.

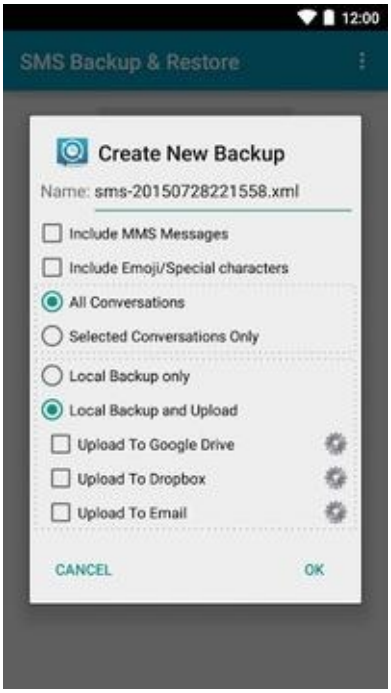
Für die Kontakte ist die Frage nach einem lokalen Backup schnell beantwortet: Das ist bereits von Haus aus dabei, auch wenn es gern übersehen wird. Einfach einmal die Kontakte-App starten, die Menü-Taste drücken, und Export auswählen – und schon landet die gesamte Kontaktliste mit allen Details, einschließlich Fotos, als VCard-Archiv auf der SD-Karte. Dieses Format versteht natürlich auch die Import-Funktion derselben App – ebenso aber nahezu jede Anwendung auf dem PC, die sich um Adressen kümmert.

Ebenfalls bereits im Auslieferungszustand an Bord: Die Synchronisation mit Google. Wer kein Problem damit hat, seine persönlichen Daten im Netz abzulegen, kann auf diese Weise Kontakte und Termine sichern. Google scheint

das als selbstverständlich anzusehen – denn wenn man die zugehörigen Einträge in der Konfiguration nicht selbst findet und abschaltet, ist dies automatisch aktiv. Aber der Synchronisation ist ein eigenes Kapitel gewidmet, daher soll hier die Erwähnung genügen.

Bei den eigenständigen Backup-Apps fange ich auch hier mit denen an, die möglichst vieles abdecken. Für diejenigen, welche Windows auf ihrem PC verwenden, ist dabei sicherlich MyPhoneExplorer die erste Wahl. Mit dem entsprechenden Pendant auf dem PC synchronisiert man Kontakte, Kalender und Notizen mit den jeweiligen Gegenstücken aus der Windows-Welt. SMS sowie Anruflisten lassen sich sichern, Dateien und Verzeichnisse synchronisieren, und mehr. Das Ganze funktioniert wahlweise über WiFi, Bluetooth, oder das USB-Kabel.

Kalender, Kontakte, SMS, Lesezeichen und Anruflisten hingegen sichert Mobile Backup 3 auf die SD-Karte. Bei den angeforderten Berechtigungen (u. a. Internet und IDs) sollte man sich jedoch überlegen, ob man diese App wirklich ohne passenden Schutz installieren möchte – zumindest, wenn man bzgl. der privaten Daten auch nur ein wenig paranoid ist; eine ohne diese Permissions auskommende Kaufversion ist leider nicht erhältlich.



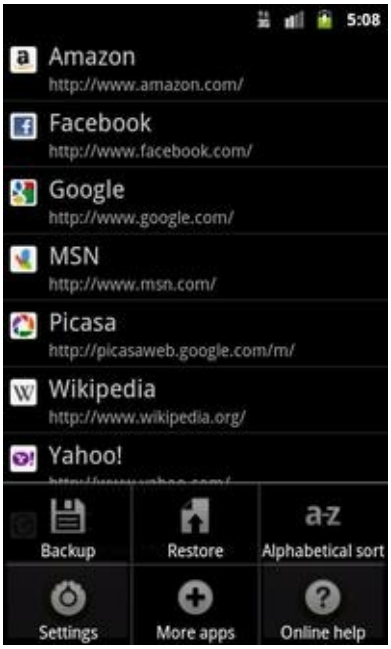
Doch auch für Einzelaufgaben stehen Apps bereit. So kümmert sich beispielsweise SMS Backup & Restore um – der Name sagt es schon – die Sicherung und Wiederherstellung der Kurznachrichten. Erstellte Sicherungen lassen sich sogar am Androiden sichten. Da für den Export das XML-Format gewählt wurde, steht auch einer Weiterverbeitung am PC nichts im Wege. Vor der Erstellung eines Backups lässt sich auswählen, ob man alles oder nur ausgewählte Konversationen sichern möchte. Auch ein Versand per Mail (hier wird die erzeugte Backup-Datei an die gewählte Mail-App übergeben) ist möglich.

Die Internet-Berechtigung wird nur in der Gratis-Version (für Werbung) benötigt. Durch Zahlung eines Lösegelds in Höhe von knapp anderthalb Euro erhält man eine von der Werbung befreite Vollversion, die diese Berechtigung auch nicht mehr anfordert.

Tipp am Rande: Einige Kommentare erwähnen explizit den mit dieser App erfolgreich vollzogenen „Umzug“ der Kurznachrichten zu einem neuen Telefon, sowie die Wiederherstellung nach einer Neuinstallation.

Für die Anrufprotokolle käme Call Logs Backup & Restore in Frage. Der Name klingt so nach …? Ja, genau: Gleicher Entwickler wie beim gerade genannten *SMS Backup*. Und auch hier wieder Topp Bewertung. Sieht von den Screenshots her auch wieder genau so aus – und ersetzt man “SMS” durch “Anrufprotokoll”, gleichen sich auch die Features weitgehend. Ebenso gibt es wieder eine Bezahlversion, mit der man die “Internet-Berechtigung” loswerden könnte. Aus dem gleichen Hause stammt dann auch APN Backup & Restore, welches sich um die APNs (Zugangspunkte zum Internet) kümmert.

Womit wir bei den Kalendern angekommen wären, die ja – im Gegensatz zum Adressbuch – von Haus aus keine Export-Funktion (außer dem Google-Sync) haben. Hier schafft die App iCal Import/Export Abhilfe, indem sie sie im iCal-Format exportiert. Neben dem lokalen Speicher auf der SD-Karte bietet die App für den Austausch WebDAV, FTP und weitere Möglichkeiten an, was wiederum die Internet-Berechtigung erforderlich macht (wer damit Bauchschmerzen hat: Bei *F-Droid* gibt es alternativ den Calendar ICS adapter, der ohne selbige auskommt). Natürlich ist auch ein Import aus diesem Format möglich, sogar über das Web: So lassen sich beliebige andere Kalender (z. B. Feiertage) importieren. Empfehlung: Sollte das jemand für Feiertage nutzen wollen, besser einen separaten Kalender dafür anlegen – dann wird man selbige später leichter wieder los.



Die integrierte Lösung für die Kontakte habe ich bereits im ersten Satz genannt (die alternativen Apps können da auch nicht mehr) – wenden wir uns also zu guter Letzt noch den Lesezeichen zu. Für eine einfache Sicherung und Wiederherstellung steht hier Bookmark Sort & Backup zur Verfügung, welches die Daten im HTML-Format der Lesezeichen von Firefox bzw. MSIE speichert (womit sich für die Ersteinrichtung der Lesezeichen auf dem Androiden u. U. ein Export aus selbigen Anwendungen anbietet). Da es sich dabei um ein textbasiertes Datenformat handelt, lässt sich das Backup auch mit einem Blick in die Datei überprüfen – und sollte die Wiederherstellung einmal fehlschlagen, sind die Daten damit noch lange nicht futsch: Datei im Browser öffnen, jeden Link einzeln anklicken und erneut als Lesezeichen hinzufügen funktioniert sodann als Notlösung. Darüber hinaus ist auch eine Sortierfunktion mit an Bord. Diese lässt sich im “Automatik-Modus” betreiben, um alle Lesezeichen alphabetisch zu sortieren – oder auch manuell, wobei man Lesezeichen einzeln in ihrer Position verschieben kann.

Die Anwender sind, soweit man das den Playstore-Kommentaren entnehmen kann, durchweg zufrieden bis begeistert. Warum die App allerdings “nach dem Booten starten” möchte, ist nicht so ganz nachvollziehbar. Die Internet-Berechtigung dient dem Werbebezug, und kann gegen einen knappen Euro mit der Vollversion „beseitigt“ werden. Leider scheint die Entwicklung jedoch eingestellt: Die aktuellste Version stammt vom September 2012 – und funktioniert nicht mit aktuellen Geräten, welche *Chrome* als Browser verwenden. Auf diesen bietet ggf. NEO Bookmark eine Alternative.

Backup in die Cloud

Das ist ja der letzte Schrei und furchtbar modern: Alles ab in die Wolke. Schon Reinhard Mey wusste (und weiß): „Über den Wolken muss die Freiheit wohl grenzenlos sein …“ Da können wir unsere Backups ja gleich einmal hinterher schieben (um uns später zu wundern, wo sie teilweise wieder herab regnen).

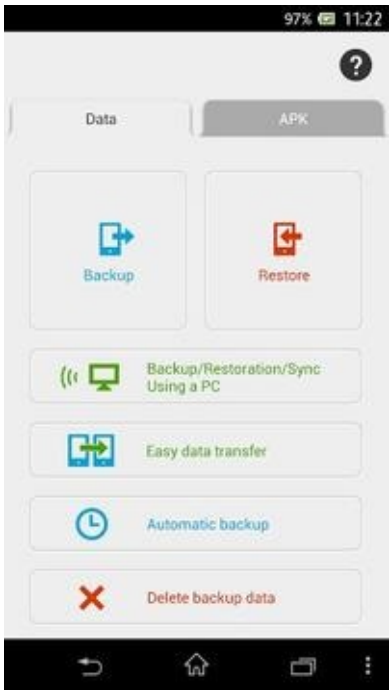
Bordmittel

Für die Wolke gibt es sie tatsächlich – jedoch sind sie extrem eingeschränkt, und oftmals auch recht unzuverlässig. So finden sich in der Gerätekonfiguration unter Datenschutz die Optionen *Meine Einstellungen sichern* sowie *Autom. Wiederherstellung*, die sich beide auf das *Google Backup* beziehen. Details dazu sind in diesem Buch bei der entsprechenden Einstellung ausgeführt. Hier nur soviel: Es werden nur einige Apps sowie deren Einstellungen unterstützt, sowie einige Android-Einstellungen (etwa Lesezeichen und WLAN-Netzwerke). Daher an dieser Stelle lediglich die „Erwähnung der Vollständigkeit halber“. Und apropos vollständig: Mit Android 6.0 soll sich in dieser Richtung etwas tun. Erfahrungen stehen allerdings noch aus.

Komplettsicherungen

Um so ziemlich alles (außer App-Daten) kümmert sich JS Backup: Kontakte, Texte, Kalendereinträge, Anruflisten, Lesezeichen, Systemeinstellungen, Fotos, Videos und mehr schiebt es wahlweise in die Cloud (Dropbox, SugarSync, Google Drive), auf den eigenen PC, bzw. die SD-Karte. Außerdem lässt sich die App sogar nutzen, um mit der „Easy Data Transfer“ Funktionalität alles auf ein anderes Android-Gerät zu übertragen. So ganz nebenbei möchte es dabei auch „nach dem Booten starten“ (für automatische Backups, siehe Screenshot) sowie SMS empfangen (huch? Wozu das?). Ähnliches leistet auch Pleex Mobile Backup – mit Ausnahme des „nach dem Booten Startens“ sowie der damit verbundenen automatischen Synchronisierung (dafür jedoch zusätzlich mit SMS Verschicken).

Da gibt es noch eine ganze Reihe weiterer Kandidaten, die diese Aufgabe mehr oder weniger vollständig erfüllen wollen – zu finden in der bereits bei den lokalen Backups genannten Übersicht. So richtig überzeugen konnte mich (abgesehen von *JS Backup*) keine dieser Lösungen – was aber nicht zuletzt auch an meiner diesbezüglich etwas konservativen Einstellung liegen mag.



Kontakte, SMS & Co.

Kontakte und Kalender? Na, diese Frage beantwortet sich bei Android eigentlich fast von selbst: Wenn man dem nicht explizit einen Riegel vorschiebt, landen diese Daten automatisch auf den Google-Servern – und werden bei bestehender Datenverbindung permanent synchron gehalten (außer, wenn der entsprechende Google-Dienst mal wieder verrückt spielt). Vergleichbare Alternativen, abgesehen von den obigen, sind mir nur mit Providerbindung in Übersee bekannt. Sofern man sich nicht selbst etwas gebaut hat – doch dazu später mehr.

Für SMS und MMS scheint es jedoch mit SMS Backup + einen Kandidaten zu geben, der durchaus einen näheren Blick wert sein könnte: Von über 50.000 Anwendern mit durchschnittlich viereinhalb Sternen bewertet, das will schon etwas heißen! Neben den genannten Kurz- und Multimedia-Nachrichten kümmert sich die App auch noch um die Anruflisten, und speichert alles (bei Nutzung unterschiedlicher Labels) im eigenen Google Mail-Account bzw. Kalender. Eine Wiederherstellung ist, mit Ausnahme von MMS, ebenfalls möglich. Übrigens möchte auch diese App einmal wieder nach dem Booten starten – mit einer automatischen Synchronisation dürfte also zu rechnen sein.

Nebeneffekt: Wenn SMS, MMS und Anruflisten automatisch im Google Mail-Account bzw. Kalender landen, kann man diese natürlich auch von jedem beliebigen PC ohne Zugriff auf das Telefon lesen. Also Vorsicht, wem man die Zugangsdaten gibt – denn diese Person sieht dann sehr wohl, wann man mit wem telefoniert hat.

Wer keine MMS zu verwalten hat, und die Anrufliste ignorieren kann, der mag vielleicht alternativ zu Gschickt greifen. Neben der Online-Sicherung kommen bei dieser App einige Zusatz-Funktionen dazu, wie etwa die Nutzung von (oft kostengünstigeren) Online-SMS-Services (genannt sind hier “sms-kaufen” und “innosend”), Statistiken über den SMS-Versand, ein SMS-Editor mit Smileys,

Zeichenspar-Funktion, Sprechblasen sowie anpassbarer Schriftgröße, Empfangsberichte, Text-to-Speech (TTS), und Nachrichten-Threads.

Die erweiterte Vollversion schlägt mit etwa drei Euro zu Buche. Diese bietet unter anderem Unterstützung für “lange” SMS mit bis zu 800 Zeichen, Textbausteine, sowie mehrere Identitäten (Absender-Nummern).

Für die zusätzlichen Web-SMS-Dienste sind natürlich separate Accounts notwendig, gleiches gilt in Bezug auf die Online-Sicherung für die Website von *Gschickt* selbst. Dass die App die Berechtigung zum Versenden von Kurznachrichten fordert, sieht sicherlich jeder ein. Auch der Zugriff auf das Adressbuch ist nachvollziehbar (wenn auch etwas kritisch im Zusammenhang mit der ebenfalls erklärbaren Berechtigung zum Internet-Zugriff). Nachdenklich stimmt hingegen, dass sie zusätzlich Telefonnummern anrufen möchte. Eine Erklärung dazu sucht man in der App-Beschreibung vergeblich; auch auf der Website wurde ich diesbezüglich nicht fündig.

Synchronisation

Die Grenzen zwischen einem Online-Backup und der Synchronisation sind fließend. Kalender und Kontakte werden unter Android beispielsweise automatisch mit den entsprechenden Google-Accounts synchronisiert – manch einer sieht das als Backup an. Im Prinzip kann man sagen: Die Synchronisation ist eine Form des Backups – ein Backup aber nicht zwangsweise eine Synchronisation. Darüber hinaus gibt es Synchronisation und Synchronisation: Man kann, wie soeben anhand der Google-Dienste aufgezeigt, die Daten „mit der Cloud“ synchronisieren – oder aber mit einem eigenen Rechner.

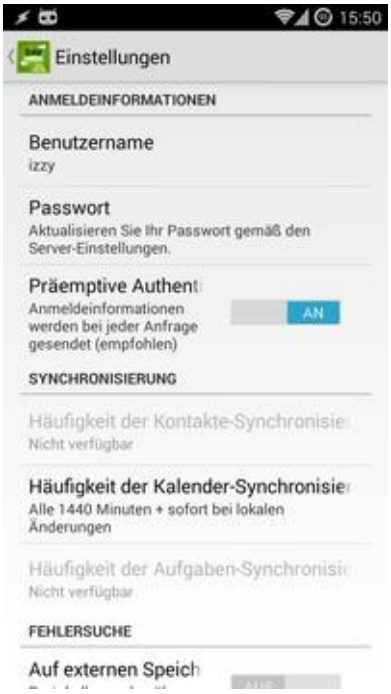
Beim Synchronisieren arbeitet man in der Regel mit zwei „gleichwertigen Kopien“ (im Unterschied zum Backup, bei dem die Daten meist komprimiert abgelegt werden). Die Daten liegen also auf der „Gegenseite“ genau so vor, wie sie es lokal tun. Und oftmals arbeitet man mit den Daten auch auf beiden Seiten: Fügt etwa Termine sowohl auf dem Androiden, als auch vom PC aus auf dem entsprechenden Server hinzu.

Unterscheiden kann man unter anderem nach der Gegenseite der Synchronisation: Landen die Daten auf „fremden Servern“ (also wieder in der Cloud) – oder auf einem eigenen Rechner? Ein nicht unwesentliches Kriterium, das ich im Folgenden auch besonders berücksichtigen werde – nicht jeder gibt schließlich seine Daten freiwillig heraus.

Kontakte und die Cloud

Zuallererst wären da die beliebten sozialen Netzwerke, die sich mit in die Kontaktliste integrieren lassen. Passende Apps gibt es etwa für *Facebook*, *Xing* und *LinkedIn* – wobei die Synchronisation von Kontakten häufig eher nur eine von vielen Funktionen ist. Meist sind zahlreiche weitere Features für die Interaktion mit dem entsprechenden Netzwerk ebenfalls integriert. Für Details möchte ich an dieser Stelle auf die entsprechende Übersicht verweisen.

Kontakte, Kalender und mehr mit eigener Hardware synchronisieren



Mein Favorit in diesem Bereich ist DAVDroid, welches sowohl mit Kontakten als auch Kalendern und Tasks per WebDAV umgehen kann. Das funktioniert wunderbar mit meiner ownCloud Installation – ist jedoch mit nahezu jedem anderen Service kompatibel, der sich über das CalDAV bzw. CardDAV Format ansprechen lässt (z. B. *Baïkal*, *iCloud*, *OS X Server*, *Zarafa*, *Zimbra* …). Für mich hat dies *Google Calendar* und *Google Contacts* vollständig ersetzt – einschließlich des Web-Interfaces.

Wie am Screenshot ersichtlich, lässt sich das Intervall für eine automatische Synchronisierung (so gewünscht) für jede Quelle separat einstellen, ebenso ist eine manuelle Synchronisation möglich. Im „laufenden Betrieb“ synchronisiert *DAVDroid* dann zu diesem Intervall – aber auch immer dann, wenn man auf dem Androiden einen Kontakt/Termin/Task hinzufügt bzw. verändert.

Zwei kleine Haken gilt es jedoch zu erwähnen: Für Kontakte und Kalender muss jeweils ein eigenes „Konto“ auf dem Android-Gerät eingerichtet werden (weshalb im Screenshot auch der Kontakt-Eintrag ausgegraut ist) – und zum Hinzufügen eines zusätzlichen Kalenders/Adressbuches muss man den entsprechenden Account löschen und neu anlegen. Zumindest letzteres steht jedoch bereits auf der Wunschliste für zukünftige Versionen.

Alternativen finden sich natürlich wieder in der zugehörigen Übersicht – etwa BirdieSync für die direkte Synchronisation mit Thunderbird ganz ohne Cloud, oder SyncRoid für gleiches mit Outlook.

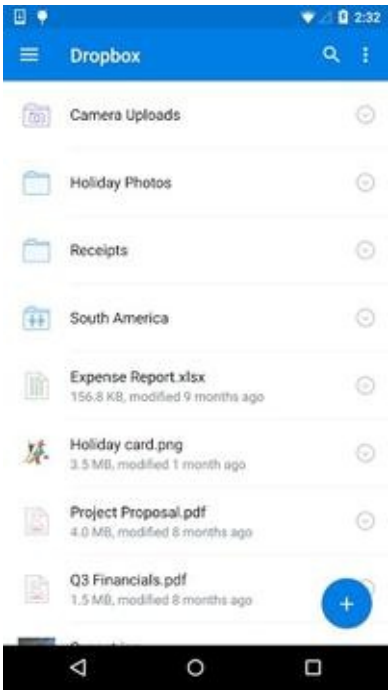
Dateien und Verzeichnisse in der Cloud

Unbewusst hat Reinhard Mey die Cloud (eine Übersicht dazu wieder bei

IzzyOnDroid) bereits 1971 in seinem Lied „Über den Wolken“ besungen. Hier einmal eine zeitgenössische Interpretation des Refrains:

Über den Wolken *[total abgehoben und völlig überbewertet]*
Muss die Freiheit wohl grenzenlos sein. *[die Daten werden über die halbe Welt verstreut!]*
Aller Kummer, alle Sorgen, sagt man, *[genau weiß es keiner]*
Blieben darunter verborgen, und dann, *[und niemand schaut da wirklich durch]*
Würde, was uns groß und wichtig erscheint *[Megabytes, Gigabytes]*
Plötzlich nichtig und klein. *[angesichts der verfügbaren Terra- und Petabytes]*

Bei dem Wort „Cloud“ denkt der Android-Jünger wahrscheinlich zuerst an Google, dann an die Dropbox. Danach folgt wahrscheinlich erst einmal eine ganze Weile – gar nichts. Also fangen wir mit den gängigen Produkten an:



Eine der größten und bekanntesten „Wolken“ ist sicher **Dropbox** – mit Client-Software für Linux, Mac OS und Windows auf dem heimischen Rechner, sowie Android, Blackberry, iPad und iPhone auf den mobilen Geräten, wird hier sicher auch die größte Bandbreite von Geräten unterstützt. Darüber hinaus gibt es für Entwickler die Möglichkeit, durch die Verwendung der bereitgestellten API die *Dropbox* direkt in ihre Applikation einzubinden – was bei vielen Apps auch bereits passiert ist. Um welche Apps es sich dabei handeln könnte, kann man im Dropbox App-Verzeichnis für alle beteiligten Plattformen (auch separat gefiltert) nachschlagen. Wobei diese Liste keinesfalls vollständig ist…

Aber schauen wir uns doch ausgewählte Clients einmal kurz an. Da wäre natürlich zuerst die offizielle Dropbox-App, die genau wie der Dienst einfach nur *Dropbox* heißt. Sie kümmert sich in erster Linie um die Synchronisation von Dateien und Verzeichnissen, wozu sie einen „Spiegel“ der im Service bereitgestellten „Online-Festplatte“ auf dem Gerät bereithält. Wird dort eine Datei abgelegt oder verändert, bemerkt der Service das, und veranlasst eine Online-Aktualisierung – gegebenenfalls wartet sie damit auch, bis eine Netzverbindung besteht. Umgekehrt erfolgen auch Aktualisierungen aus der Cloud, falls eine Datei von einem anderen Client modifiziert wurde. So sind alle Beteiligten stets auf dem aktuellen Stand.

Dropbox-Ordner lassen sich für Freunde und Verwandte freigeben. Oder EMail-Anhänge direkt in der Dropbox abspeichern. Oder Dokumente direkt in der Dropbox bearbeiten. Viereinhalb Sterne bei weit über einer Million Bewertungen zeugen nicht nur von reger Nutzung, sondern auch von vielen zufriedenen Anwendern.

Für einen derart beliebten Dienst stehen natürlich auch zahlreiche Apps von Drittanbietern zur Verfügung – die für manche Belange durchaus besser geeignet sind als das „Original“. Hervorheben möchte ich FolderSync, welches ich selbst einsetze. Neben *Dropbox* unterstützt diese App nämlich eine ganze Reihe weiterer Cloud-Dienste (etwa *Google Drive*, *Box* und *OneDrive*) – aber ebenso eigene Resourcen per WebDAV (etwa *ownCloud*), Samba (der Windows-PC, aber auch Linux), FTP (z. B. der eigene Web-Server) und SSH. Konfigurierte Verzeichnisse lassen sich manuell oder automatisch (ggf. auch auf das eigene WLAN beschränkt) synchronisieren. Erwirbt man die Kauf-Version, steht selbst einer Automatisierung mit Tasker nichts im Weg.

In Sachen Verschlüsselung sah es anfangs ein wenig Mau aus, wollte man dabei zu verschiedenen Systemen kompatibel sein (also etwa vom Androiden *und* vom PC unter Windows *und* Linux auf die Daten zugreifen, oder den selben Datenbestand bei verschiedenen Services unterbringen). Besserung versprach jedoch recht bald die App BoxCryptor, die sich u. a. für *Dropbox* und *Google Drive*, aber auch für *WebDAV* eignet, und die Daten mit AES-256 verschlüsselt auf dem jeweiligen Server ablegt. Die Verschlüsselung erfolgt dabei auf dem Gerät des Anwenders – sodass der jeweilige Cloud-Anbieter die Daten nicht einsehen kann. Für den PC sind Desktop-Varianten für Linux, Mac und Windows verfügbar, welche sich ohne großartige Installation verwenden lassen („portable Apps“).

Natürlich findet auch *Google Drive* unter Android gute Unterstützung – das Gegenteil wäre verwunderlich: Die „offizielle App“ ist i. d. R. bereits auf dem Gerät vorinstalliert, andere Apps wie obiges *FolderSync* haben den Zugriff integriert. Zahlreiche weitere Cloud-Dienste buhlen mit unterschiedlichem Erfolg um die Gunst der Anwender, welcher „die Qual der Wahl“ hat. Eingangs genannte Übersicht hilft hoffentlich bei der Entscheidungsfindung.

Dateien und Verzeichnisse auf eigener Hardware

Zugegeben: Nicht jeder kann sich gleich einen fetten Serverpark in den Keller stellen. Aber hey, auch MicroSoft hat einmal ganz klein in einer Garage angefangen. Und wenn wir gerade vom Windows-Hersteller sprechen: Den kann man doch auch freigeben. Ich meine, den Speicherplatz auf dem Windows-Rechner. Auf Linux und Mac OS Rechnern heißt es dazu: „Tanze **Samba** mit mir …“, denn hier gibt es den freien Samba-Server – bei den meisten Linuxen von Haus aus im Repository enthalten, und auch für Mac OS zu haben. Auch NAS-

Systeme bieten dies an, verbrauchen im Vergleich zum herkömmlichen PC kaum Strom – und sind mittlerweile auch recht erschwinglich. So lässt sich Speicherplatz im heimischen Netzwerk zur Verfügung stellen – und darauf können dann auch unsere Androiden zugreifen.

Beispielsweise mittels AndSMB, einem SMB-Client für Android. Diese App erlaubt Zugriffe auf Windows-Freigaben ebenso wie auf Samba-Freigaben eines Linux- oder Mac OS-Rechners, handelt es sich doch um das gleiche Übertragungs-Protokoll. Ein Dateibrowser ist in der App enthalten, und ermöglicht sowohl den Up- als auch den Download. Automatisiert man das Ganze nun, was mit dieser App möglich ist, hat man eine Synchronisation von Ordnern – und somit eine eigene minimale Cloud.

Alternativ könnte Sweet Home zum Einsatz kommen. Die Beschreibung der Pro-Version dieser App bringt es auf den Punkt: Keine Kabelsuche, auch muss man den Androiden nicht mehr umständlich zerlegen, um die SD-Karte zu entnehmen: Sobald man zu Hause ankommt, und sich der Androide ins heimische WLAN verbindet, werden die Daten automatisch synchronisiert. Die Gratis-Version punktet (laut Beschreibung) mit häufigen Updates (sowie dem unvermeidlichen „Nagging“), während die Bezahlversion gut getestet und stabil ist. Benutzer der freien Version sind also Beta-Tester …

Da Samba bzw. Windows-Freigaben in der Regel jedoch auf das jeweilige lokale Netzwerk beschränkt sind, ist man mit einer derartigen Mini-Wolke örtlich recht limitiert. Für viele Fälle mag dies ausreichend sein – und was nicht über fremde Netze geht, ist auch vor „Schnüfflern in der Mitte“ relativ sicherer. Wird jedoch ein Zugriff auch von unterwegs gewünscht, greift man am Besten zu einem Multi-Talent wie oben genanntem *FolderSync*, das sich auch auf Protokolle wie **FTP[S]** und **SFTP/SSH** versteht. Und auf der Gegenseite natürlich zu einem passenden Server: SFTP/SSH sind unter Linux von Haus aus dabei, ein FTP-Server lässt sich auf allen Systemen relativ einfach aufsetzen – und verfügt man über einen eigenen Root-Server für z. B. seine Webseiten, sollte einer der beiden Dienste (SSH-Server und/oder FTP-Server) ohnehin bereits vorhanden sein.

Eine passende App auf unserem Androiden wäre dann beispielsweise AndFTP. Topp bewertet, bringt diese gar einen Manager für die auf dem Server abgelegten Dateien mit. Sie versteht sich auf die Protokolle FTP, FTPS, SCP sowie SFTP, und kann mehrere Serverkonfigurationen verwalten. Neben Upload, Download und Synchronisation ist auch das Share-Menü sowie direkte Unterstützung für die Galerie eingebunden. Für SSH Verbindungen (SCP) können überdies RSA/DSA Schlüssel verwendet werden. SCP und Ordner-Synchronisation sind allerdings nur in der Pro-Version verfügbar.

Security

Wie eingangs erwähnt, soll es nun um den Schutz vor Fremdzugriffen gehen. Diesbezüglich möchte ich hier mehrere Ebenen unterscheiden:

- Verschlüsselung: Besonders interessant, wenn Daten auf fremden Servern (Stichwort: Cloud) abgelegt werden. Da es auf selbigen nicht mehr in der Macht des Daten-Eigentümers liegt, den „physikalischen Zugriff“ abzusichern, sichert man auf diese Weise „logischen Zugriff“ ab: Selbst wenn ein „Angreifer“ an die Dateien gelangt, muss er damit noch lange nicht an die Daten (Inhalte) kommen.
- Zugriffsrechte: Worauf darf eine App zugreifen? Unschön, wenn da jemand im Hintergrund ungefragt und ungewünscht sämtliche Kontaktdaten „irgendwo hin“ verschickt. Oder andere böse Sachen mit unseren Daten anstellt.
- Malware und Viren: Sollten eigentlich mit dem vorigen Punkt bereits abgearbeitet sein, werden aber dennoch explizit separat behandelt.

Wer sich genauer informieren möchte, welche Schutzmaßnahmen Android selbst anwendet, findet in der Computerwoche einen interessanten Artikel aus dem Jahre 2011, der sich dem Thema eingehend widmet.

Verschlüsselung

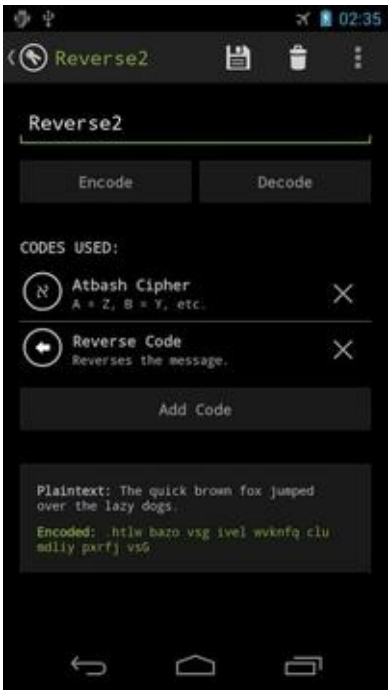
Texte Verschlüsseln

(Eine Übersicht mit Apps zum Thema Textverschlüsselung findet sich auf meiner Website.)

Das tat schon Gaius Julius Caesar – weshalb einer der einfachsten Codes auch nach ihm benannt wurde: die Caesar-Verschlüsselung. Die bekannteste Variante aus dieser Reihe trägt auch den Namen „ROT-13“ – was nicht etwa darauf hinweisen soll, dass Caesar damit die Cleopatra 13 Mal zum Erröten brachte (tat er bestimmt nicht: Nach Sueton verwendete Caesar nämlich ROT-4). Vielmehr heißt es: ROTiere jeden Buchstaben 13 Mal – aus „A“ wird also „N“. So wird aus dem Wort „Text“ das etwas schwerer lesbare „Grkg“. Zur Entzifferung verschiebt man die Buchstaben einfach zurück (aus „N“ wird „A“).

Von einer sicheren Kodierung kann hier also beileibe nicht die Rede sein – doch veranschaulicht dies wunderbar, wie Verschlüsselung prinzipiell funktioniert. Genutzt wird es trotzdem – weniger zu Sicherheitszwecken, als vielmehr um z. B. unbeabsichtiges Lesen von Spoilern zu vermeiden. Zumindest Geo-Cachern dürfte diese Verschlüsselungsart auch vertraut vorkommen.

So wird die App Caesar’s Cipher von ihren Benutzern als als hilfreiches und nützliches Tool beim Geo-Caching beschrieben (hier werden Hints kodiert und dekodiert). Oder, wie es ein englischer Kommentar formuliert: *Very useful. Now my teacher has now idea what the gibberish on my paper says.* Hoffentlich war da nicht der Geschichtslehrer gemeint …



Caesars Code hat natürlich einen großen Haken: War er zu Zeiten des genannten Römers noch relativ sicher, benötigt man heutzutage für das „Knacken“ des Codes wahrscheinlich weniger als eine Sekunde – bedenkt man, dass ein Computer lediglich 26 (bei reinen Großbuchstaben) bzw. maximal 256 Varianten (8-Bit ASCII) dafür durchspielen muss. So schützt ein reiner Caesar-Code allenfalls vor dem „schnellen Lesen“, nicht aber vor dem Lesen an und für sich.

Doch was galt bereits im Altertum als besonders kompliziert? Da war doch was mit einem Gordischen Knoten, an dessen Auflösung sich viele schlaue und starke Männer vergeblich versuchten. Nach diesem ist offenbar die App Gordian Secret Code Tool benannt. Diese nutzt prinzipiell ähnliche Methoden – erlaubt jedoch, sie zu kombinieren. Zugegeben: Ein Caesar-4 mit anschließendem Caesar-9 ergäbe auch nur wieder ROT13. Kombiniert man ihn jedoch beispielsweise mit Vigenère, wird die Sache schon komplexer. Ob der potentielle „Knacki“ deswegen gleich ins Schwitzen kommt, steht natürlich auf einem anderen Blatt.

Bleiben wir zunächst noch bei den Legenden – machen jedoch einen größeren Zeitsprung über ein paar Jahrtausende in Richtung Gegenwart. Damals war es vielen ein Rätsel, was das deutsche Militär da so funkte. Auf Griechisch heißt Rätsel „αίνιγμα“, mit lateinischen Buchstaben schreibt man es Enigma – die Rede ist natürlich von der bekannten Rotor-Schlüsselmaschine. Dieses Original benutzte auch eine Kombination, und zwar waren hier mehrere Walzen hintereinander geschaltet, die nach jedem Buchstaben auch noch wie ein Kilometerzähler weiterbewegt wurden. Dies bedeutet, dass sich der Schlüssel nach jedem Buchstaben ändert: Hätte Caesar aus einem *OTTO* vielleicht ein *GLLG* gemacht, wäre bei einer Enigma eher etwas wie *PQWS* dabei herausgekommen. Daran hatten die Alliierten im zweiten Weltkrieg eine ganze Weile zu knabbern.

Wer dieses Feeling einmal nachempfinden möchte, bekommt mit der App Pocket Enigma Machine (derzeit nur im Amazon Market und für ca. anderthalb Euro zu haben) Gelegenheit dazu: Der Entwickler hat hier Bild- und Tonaufnahmen des Originals verwendet, damit sich das Ganze möglichst „echt“ anfühlt.

Nah dran waren wir ja nun bereits, doch nun wollen wir uns endgültig der Gegenwart zuwenden. Spricht man heute von einer „sicheren Verschlüsselung“, so fallen Begriffe wie RSA (asymmetrisch) oder DES (symmetrisch). Den Nachfolger des letztgenannten (AES) macht sich Crypt Haze zu Nutze, um Mails und Kurznachrichten zu ver- und entschlüsseln (wozu es natürlich die entsprechenden Zugriffs-Rechte auf Kontakte und SMS anfordert – allerdings auf die allgegenwärtige Internet-Berechtigung verzichtet). Auch die App Encrypt It setzt auf eine 256Bit AES Verschlüsselung, und verbindet diese zur Erhöhung der Sicherheit noch mit einem Salt. Diese App verlangt keinerlei Permissions.

Was aber verwendet der sicherheitsbewusste Mensch von Heute, wenn er Nachrichten verschlüsselt oder signiert übertragen möchte? Das Zauberwort heißt GPG. Das „G“ steht hier für „GNU“, und „PG“ für den „Privacy Guard“ – also den Beschützer der Privatsphäre. Hier werden Schlüsselpaare verwendet, bestehend aus einem „privaten“ und einem „öffentlichen“ Schlüssel. Die Namen sagen es bereits: Den einen hütet man wie den eigenen Augapfel, den anderen gibt man frei heraus. Dank dieses öffentlichen Schlüssels ist nun jeder Anwender von GPG in der Lage, eine Nachricht so zu verschlüsseln, dass sie nur vom Besitzer des zugehörigen privaten Schlüssels wieder dekodiert werden kann. Umgekehrt lassen sich mit dem privaten Schlüssel Nachrichten signieren. Die Echtheit dieser Signatur kann jeder mit dem frei verfügbaren öffentlichen Schlüssel überprüfen.

Soweit die Theorie und das grobe Prinzip, kommen wir zur Praxis. Die bekannteste App in diesem Segment ist sicherlich APG. Als Open-Source Software stellt das Projekt sicher, dass es keine “Hintertürchen” geben kann (die würden schnell entdeckt). Durch die Verwendung von Schlüsselpaaren (wie soeben beschrieben) wird darüber hinaus auch eine relativ hohe Sicherheit gewährleistet – zumindest, so lange der private Schlüssel nicht kompromittiert wurde.

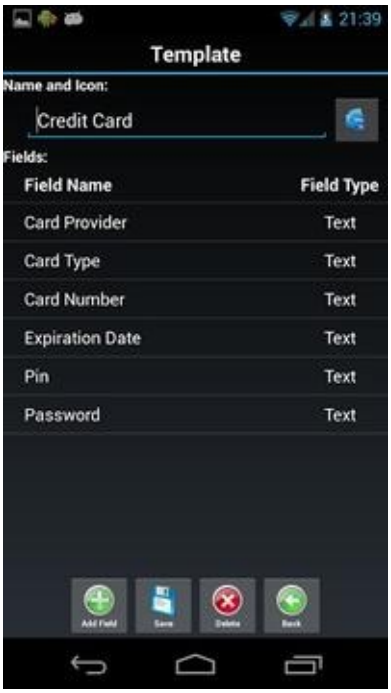
Mit der App lassen sich PGP/GPG Schlüsselpaare verwalten sowie Mails und Dateien verschlüsseln, signieren, entschlüsseln und die Signaturen überprüfen. Unterstützt wird GMail, aber auch in K9 lässt sich die App integrieren; für andere Apps wird eine API bereitgestellt. Seit neuestem werden auch Schlüsselserver unterstützt (daher auch die Internet-Berechtigung). Viereinhalb Sterne bei über 3.000 Bewertungen bestätigen die Qualität dieser App aus Anwendersicht.

Passwort-Safes

Ein ganz spezieller Fall von „kurzen Texten“, die es um jeden Preis abzusichern gilt, stellen sicher Passwörter und PIN-Codes dar – eine passende Übersicht verfügbarer Apps findet sich hier. Eine App, der ich solche Daten anvertraue, sollte möglichst mein Handy auch nicht verlassen – sonst bestünde ja immer die Möglichkeit, dass sie diese sensiblen Daten (von mir unbemerkt) an Dritte weiterreicht. Zum Glück sehen das viele Entwickler ähnlich – anderen ist hingegen beispielsweise wichtiger, auch ein „Backup“ in der Cloud ablegen zu können. Das wäre aus meiner Sicht allerdings, wenn überhaupt, Aufgabe einer anderen App aus anderem Hause – ich bin halt ein wenig paranoid. Daher werde

ich mich hier auf Apps beschränken, die über keine eigene „Internet-Berechtigung“ verfügen.

Da wären zum Beispiel die PINs von EC- und Kreditkarten. Der Eine oder die Andere kennt vielleicht noch diese kleinen „Geheimkärtchen“, auf denen man sich diese in der Regel vierstelligen Zahlencodes „verschlüsselt“ notieren kann – und genau dieses Prinzip greift PIN’r auf: Die PINs werden in einer Tabelle voller Zufallszahlen versteckt. Nur wer das Geheimwort kennt, findet die richtige Kombination wieder. Dumm für den Schnüffler: Jedes beliebige „Geheimwort“ liefert passende Zahlencodes, mit „Passwort erraten“ ist es also auch nicht ganz so einfach.



Doch oftmals geht es um mehr als nur PINs: Zugangsdaten für Foren und Websites, Kreditkartendaten, Mailkonten, Lizenzschlüssel für Software… Zahlreiche Daten sind sensitiv genug für einen Passwort-Safe. Da bieten sich Apps wie beispielsweise Password Safe an. Für jeden Anwendungsbereich lassen sich hier Vorlagen (Templates) definieren: Für ein Mailkonto benötigt man in der Regel drei bis vier Felder (Name, Mail-Adresse, Passwort sowie ggf. zugehörige Webseite) – für weitere Mailkonten sind es die gleichen vier Felder. Wie es im Fall von Kreditkarten aussehen könnte, zeigt der Screenshot. Einige Templates bringt die App gleich mit, weitere kann man selbst erstellen (bzw. die vorhandenen entsprechend anpassen). Jeder Vorlage lässt sich darüber hinaus auch ein Icon zuordnen, was die Erkennung vereinfacht. Bei Erfassung der Daten wählt man dann einfach das jeweilige Formular (also die Vorlage) aus, und füllt die Daten in die Felder; selbst jetzt lassen sich bei Bedarf noch zusätzliche Felder hinzufügen.

Da so eine Datensammlung schon einmal etwas größer ausfallen kann, ermöglicht *Password Safe* auch eine Einteilung in Kategorien; die Darstellung erfolgt hier in einer Baumstruktur, in der man sich leicht zurechtfinden kann. Davon kann man sich in einer Gratis-Version der App zunächst überzeugen – die erfassten Daten lassen sich in die für ca. zwei Euro erhältliche Vollversion übernehmen, die dann auch einen Export/Import im CSV-Format ermöglicht.

Und bevor ich es zu erwähnen vergesse: Damit die Daten vor fremden Blicken geschützt sind, werden sie mit 128Bit AES verschlüsselt gespeichert. Wem das noch nicht ausreicht, der kann einen Blick auf Passman werfen. Hier gibt es zusätzlich eine „Selbstzerstörung“: Nach einer konfigurierbaren Anzahl an Fehlversuchen bei der Eingabe des Master-Passwortes werden sämtliche

erfassten Daten automatisch gelöscht. Ein etwaiger Handy-Dieb hat somit wenig Chancen – und man selbst hat ja hoffentlich ein Backup der Datenbank an anderer Stelle (z. B. auf dem PC) abgelegt.

Apropos PC: KeePass bietet eine Passwort-Verwaltung, die sich sowohl auf dem Androiden als auch auf dem PC (Linux, MacOS, Windows) und sogar unter iOS, Blackberry sowie PalmOS nutzen lässt – völlig gratis und sogar Open Source.

Dateien und Verzeichnisse

Da fällt einem sicher als erstes TrueCrypt ein – wobei man unter Android allenfalls *TrueCrypt*-Container auf dem Androiden ablegen kann, um so einen "portablen, sicheren Speicher" bei der Hand zu haben (wie man das für besonders große Container umsetzt, beschreibt dieser Artikel).

Vom Verhalten am Nächsten kommt dem der LUKS Manager, der AES-verschlüsselte Container verwaltet. Diese können wie andere Dateisysteme eingebunden werden, und stehen dann ebenso anderen Apps zur Verfügung. *LUKS* kümmert sich dabei sowohl um die Erstellung/Entfernung als auch das Einbinden/Aushängen der entsprechenden Container.

Die App ist zwar gratis – doch sind die Hürden für den "einfachen Anwender" ein wenig hoch. So werden nicht nur root inkl. der App *BusyBox* vorausgesetzt – auch ein passender Kernel, der DM_CRYPT und "Loopback" unterstützt, muss vorhanden sein (beides Dinge, die auf die meisten Standard-Kernels/Standard-ROMs nicht zutreffen). Womit *LUKS* für die meisten Android-User leider als Option ausfällt. Hinzu kommt, dass die App seit 2012 keine Updates mehr erfahren hat.

Alternativ gibt es EDS. Und diese App arbeitet mit TrueCrypt-kompatiblen Containern. Auch ist von den bei *LUKS* genannten Hürden hier an keiner Stelle zu lesen. Sogar Dropbox-Support ist direkt aus der App (Vollversion) heraus möglich, sodass man seine verschlüsselten Container in der Cloud ablegen kann – das mindert die Bedenken gegen „Schnüffler im Netz", da diese ja in den Container nicht hineinschauen können.

EDS gibt es in einer Gratisversion (ohne Dropbox-Support und mit minimalen Rechte-Anforderungen, u. a. kein Netzwerkzugriff), sowie in einer Vollversion für derzeit ca. fünf Euro. Beide Varianten sind sehr gut bewertet.

Für Anwender ohne root wäre SecretVault eine Alternative. Anders als bei *LUKS* handelt es sich hier nicht um Open Source – wodurch die auch in der Kaufversion geforderte Berechtigung zum Internet-Zugriff manch einem ein wenig Bauchschmerzen bereiten dürfte. Selbst in der Pro-Version lässt sich nur ein einziger Container verwenden, der mit der App selbst im Basisverzeichnis der SD-Karte erstellt wird. Wurde dieser Container eingebunden, steht er transparent als Verzeichnis im System zur Verfügung – bis die App selbst beendet oder der Container aus der App heraus geschlossen wurde.

Detailliertere Informationen, eine Reihe weiterer Screenshots sowie eine kurze Beschreibung des Umgangs mit *SecretVault* finden sich im Blog der App. Auch diese App wurde seit Frühjahr 2013 nicht mehr aktualisiert.

Damit wäre der Vorrat an „transparenten Container-Apps“ allerdings leider bereits erschöpft. Es verbliebe aber noch die Möglichkeit, Dateien und Verzeichnisse separat zu verschlüsseln. Der Favorit hierfür dürfte Droid Crypt heißen, und verschlüsselt wahlweise einzelne Dateien oder ganze Ordner (auf Wunsch gleich rekursiv) mittels AES. Auch diese App vermag es, sich als Mittler zwischen den Apps und den verschlüsselten Daten zu betätigen – und so eine gewisse Transparenz zu ermöglichen. Der Anwender muss also nicht erst die benötigten Daten von Hand entschlüsseln, um sie mit einer App zu bearbeiten (und nicht vergessen, sie hinterher wieder zu verschlüsseln). Wer mag, kann zusätzlich eine Datenkompression vornehmen lassen: Auf diese Weise spart man gleichzeitig Speicherplatz.

Interessant ist auch die Wahl des Passwortes. Oder auch nicht – denn ein solches ist nicht zwingend erforderlich. Stattdessen lässt sich auch ein “Schwenkmuster” verwenden.

Die Testversion von *Droid Crypt* verlangt noch den Zugriff auf’s Internet für Werbeeinblendungen; nach 60 Tagen Testzeitraum ist sie darüber hinaus auf die Ver- und Entschlüsselung von einzelnen Dateien beschränkt. Beides gilt natürlich nicht für die Vollversion, die man bereits für ca. zwei Euro erwerben kann. Eine kurze Liste weiterer möglicher Kandidaten für die Ver-/Entschlüsselung von Dateien und Verzeichnissen findet sich bei *IzzyOnDroid*.

Ab Android 3.0 (aka *Honeycomb*) auf Tablets bzw. 4.0 (aka *Ice Cream Sandwich*) auch für Smartphones (für Besitzer des Samsung Galaxy S2 bereits ab Android 2.3 *Gingerbread*) sind all diese Umwege für viele Anwender unnötig

geworden: Hier ist eine Verschlüsselung des gesamten Gerätes von Haus aus mit dabei. Zum Einsatz kommt dafür dm-crypt mit 128-Bit AES-Verschlüsselung. Der Anwender muss dann bei jedem Bootvorgang das zugehörige Kennwort zur Entschlüsselung des Dateisystems eingeben – ab diesem Zeitpunkt stehen die Daten dann allerdings unverschlüsselt zur Verfügung.

Apps mit Passwort absichern

Viele Gründe kann es geben, warum man so etwas tun möchte: Etwa wenn man sein Handy „mal eben“ einem Kollegen „nur für einen kurzen Anruf“ übergibt. Oder es einem Freund „für ein paar Tage“ borgt, während dessen Telefon sich „in Reparatur“ befindet. Darüber hinaus kann es sich als sinnvoll erweisen, seinem (vorpubertären?) Kinde die Nutzung des Smartphones zwar prinzipiell nahezubringen – aber dennoch nicht zuletzt aus Sicherheitsgründen etwas einzuschränken.

Im letztgenannten Fall bietet sich eine App wie NetSpark Parental Control an, die einige nützliche Einstellungen bietet. So lässt sich die Nutzung von Apps zeitlich limitieren (damit sich der „Zögling“ nicht ausschließlich mit selbigen beschäftigt), oder auch gänzlich unterbinden. Letzteres gilt auch für „Online-Einkäufe“. Ebenso lässt sich die Installation bestimmter Apps (etwa solcher mit In-App-Einkäufen, oder Chat-Apps) sperren. Ein Internet Filter (mit White- und Blacklisten) sorgt darüber hinaus für ein gewisses Maß an „Safe Browsing“. Über neue Installationen, versuchte „Regel-Verstöße“ und Weiteres können sich Eltern von der App benachrichtigen lassen. Soll das Gerät von mehreren Personen verwendet werden, ist dafür die Verwendung verschiedener Profile möglich.

Alles in Allem ein guter Rundum-Schutz für den Nachwuchs. Wer eine genauere Überwachung wünscht – etwa um auch zu wissen, wo der Zögling gerade steckt, eine Warnung bei Verlassen/Betreten bestimmter geografischer Bereiche (Schule, Spielplatz, etc.) zu erhalten, u. a. m.: Die Übersicht Kinderschutz zeigt auch dafür nützliche Apps auf.

Für die eingangs genannten Freunde und Kollegen wäre ein solcher Kinderschutz allerdings ein wenig über das Ziel hinaus geschossen: Ein einfacher App-Locker eignet sich da weit besser. So bietet etwa Super AppLock die Möglichkeit, ausgewählte Apps (und auch die Liste zuletzt genutzter Apps) mit einer PIN, einem Passwort oder einem Pattern zu schützen. Darüber hinaus kann sie sogar als „Vault“ für Fotos, Videos sowie andere Dateien dienen, und lässt

sich per Addons beispielsweise um einen Gast-Modus oder Kinderschutz ergänzen. Wer am „Look-and-Feel“ schrauben möchte, findet ebenso etliche Themes als Addon.

Den ultimativen App-Locker gibt es wohl kaum, da die Ansprüche doch recht verschieden sind. Daher sollte man sich in genannter Übersicht ruhig einmal genauer umschauen: Andere Locker bieten u. a. Features wie eine optionale „Selbstzerstörung“ bei zu vielen Fehlversuchen (quasi eine Art Diebstahl-Schutz für die Daten), machen in selbiger Situation ein Foto des „Hackers“, gaukeln statt einem „Zutritt verboten“ einen Absturz der entsprechenden App vor, und bieten andere interessante Features.

Zugriffsrechte

Permissions werden oftmals zu wenig beachtet oder falsch interpretiert. Nur wenige scheinen ihre Bedeutung überhaupt *richtig* zu verstehen und korrekt mit ihnen umzugehen – das betrifft nicht nur die Anwenderseite, sondern ebenso die Entwickler. Wofür sind diese „Permissions“ eigentlich gut, was nützen sie? Um das zu verstehen, müssen wir einen kurzen Blick hinter die Kulissen werfen.

Android baut bekanntermaßen auf Linux auf. Hier verwaltet das System u. a. Benutzer und Benutzergruppen, um die „Eigentumsverhältnisse“ zu klären: Auf die Daten eines anderen Benutzers kann somit nicht ohne weiteres zugegriffen werden. Während unter Linux ein Benutzer vereinfacht betrachtet einer am Rechner angemeldeten Person entspricht, teilt Android jeder App eine eigene Benutzer-ID zu. Somit kommt eine App in der Regel nicht ohne weiteres an die Daten der anderen Apps heran. Ist sie mit keinerlei expliziten Permissions ausgestattet, kann sie nur „völlig ungefährliche“ Aktionen ausführen: Etwas auf dem Bildschirm anzeigen, eigene Daten speichern, und ähnliche Dinge.

Für den Zugriff auf Spezielleres muss der Entwickler seine App entsprechende Permissions anfordern lassen – und zwar bereits bei der Installation (im Nachhinein ist dies auf normalen Wegen vor Android 6.0 nicht mehr möglich). Am bekanntesten ist hier sicher der Zugriff auf das Internet. Aber auch der Zugriff auf persönliche Daten wie Kontakte und Termine, das Benutzen von Hardware-Elementen wie Kamera und Vibrator, oder gar das Anpassen von Systemeinstellungen bedarf expliziter Genehmigung. Eine Auswahl verfügbarer Permissions und ihrer Bedeutung findet sich im Anhang Android Permissions.

Wo liegen nun die größten Fehler im Umgang mit diesen Permissions?

Bei Anwendern am ehesten darin, dass sie diese häufig ohne genaueres Hinschauen abnicken, weil sie die App ja haben wollen. Dies liegt nicht zuletzt in einem Schwachpunkt des Android-Systems begründet: „Alles oder nichts“ – entweder alle geforderten Permissions werden akzeptiert, oder die fragliche App kann halt nicht installiert werden. Wünschenswerter wäre es hier, einzelne Permissions gezielt ausklammern zu können. Das würde zum Einen die Anwender motivieren, sich überhaupt einmal mit ihnen zu beschäftigen – und zum Anderen so manch einer Malware die Grundlage entziehen. Möglich ist dies derzeit nur, sofern man über root-Rechte verfügt, oder die entsprechenden Apps modifiziert.

Bei Entwicklern ist der Fehler häufig, dass sie zu viele Permissions anfordern. Entweder aus Unwissen, aus Bequemlichkeit („so kann ich nichts vergessen“) – oder mit dem Hintergedanken: Wenn das später mal gebraucht wird, und ich es erst dann hinzufüge, bekomme ich nur wieder böse Kommentare …

Was sollte man also im Umgang mit Permissions besser beachten?

Der Anwender sollte sich vor der Installation einer App genauer fragen: Ergeben die geforderten Permissions für die gebotene Funktionalität Sinn? Sicher benötigt eine SMS-App Zugriff auf die Kurznachrichten, und muss selbige auch senden können (auch wenn dies natürlich Kosten verursachen kann). Sie muss aber weder Telefonanrufe tätigen, noch den Netzwerkstatus ändern können. Eine Wallpaper-App wiederum hat an den Kurznachrichten gar nichts zu suchen. Die Alarmglocken läuten in den lautesten Tönen, wird etwa Permission zur Installation weiterer Packages gefordert (was allerdings i. d. R. ins Leere laufen dürfte, da an dieser Stelle weitere Sicherheitsmaßnahmen des Systems greifen: Diese Permission wird nur System-Apps genehmigt).

Sicher gibt es hier Grenzfälle: Besagte SMS-App könnte das Internet zum Versenden kostenloser/kostengünstiger SMS über entsprechende Web-Services nutzen. Sie benötigt natürlich auch Zugriff auf die Kontakte (wohin schickt man wohl SMS?). Und spätestens hier taucht die Frage auf: Könnte sie dann eine Kopie der kompletten Kontaktliste an eine dubiose Website verschicken? Die Antwort: Ja, sie könnte – sie hat schließlich sowohl vollständigen Zugriff auf die Kontakte (lesend genügt ja), als auch auf das Internet. Da kann man wohl nichts machen? Oh doch: Auf den Entwickler zugehen, damit er sich den folgenden Abschnitt zu Herzen nimmt:

Entwickler sollten natürlich ihre Apps nur die Permissions anfordern lassen, die wirklich benötigt werden. Das lässt auch ihre Apps vertrauenswürdiger aussehen. Bei „gefährlichen Kombinationen“ wie „Kontaktdaten lesen & Internet“ sollte, sofern möglich, die Auslagerung in ein AddOn in Erwägung gezogen werden. Positives Beispiel: Locus Maps. Sicher ergibt hier ein Zugriff auf die Kontaktdaten Sinn: Um deren Adresse auf der Karte anzeigen zu können. Für den Download von Kartenmaterial wird allerdings der Zugriff auf das Internet benötigt. Der Entwickler entschied sich also, den Kontaktdaten-Zugriff in ein eigenes AddOn

(Locus - addon Contacts) auszulagern. Somit sind die Permissions der Kern-App wieder „unanstößig“ – und dem Anwender steht es frei, die App mit oder ohne diese „Bequemlichkeit“ des Zugriffs auf die Kontaktadressen zu nutzen. Vorbildlich, Menion! Mögen sich viele Entwickler daran ein Beispiel nehmen!

Permissions überprüfen

Leider kommt es oftmals vor, dass man die Liste der geforderten Permissions vor der Installation eben *nicht* (oder nicht genau) gelesen hat – obwohl man es sich fest vorgenommen hatte. Oder, dass die Einsicht erst beim Lesen dieses Buches kam – und natürlich bereits allerhand „unbedacht installiertes“ auf dem Androiden „herumliegt“. In beiden Fällen ist eine nachträgliche Überprüfung geboten.

Was nicht immer einfach ist: Bei den vielen unterschiedlichen Permissions ist nicht immer klar, was sie eigentlich bedeuten. Wer hier einmal alles ein wenig genauer abklopfen will, findet unter Umständen in aSpotCat das Werkzeug seiner Wahl. Mit dieser App lassen sich zum Einen alle Apps auflisten, die eine gewählte Permission verlangen – oder umgekehrt alle Permissions, die eine bestimmt App verlangt. Dabei wird zu den Permissions auch jeweils eine Erklärung geliefert, was diese bedeuten. Ob von ihnen eine potentielle Gefahr ausgeht, besagt auch die kleine jeweils neben ihnen angezeigte „LED“. Auch hier heißt „rot“ nicht gleich, dass eine App „böse“ ist – sondern lediglich, dass sie das potentiell sein *könnte*.

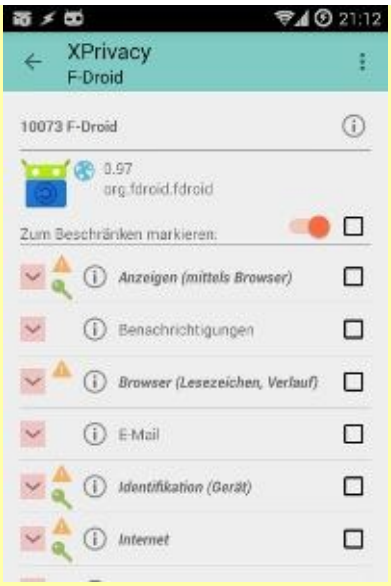
Es gibt natürlich noch weitere Apps, die ähnliches tun – einige davon sind in dieser Übersicht aufgeführt.

Permissions überwachen

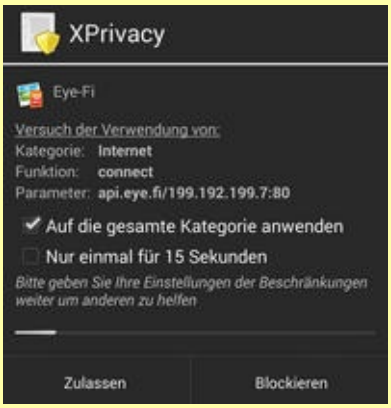
Eine echte Überwachung einschließlich der Möglichkeit, einer App bei „unbefugtem Zugriff“ auf die Finger zu hauen – das ist ohne root leider nicht möglich. Denn dazu bedarf es ja eines Zugriffs auf die Aktivitäten der jeweiligen App – eine Sache, die das Berechtigungssystem so nicht erlaubt. Schade daher, dass die entsprechende Möglichkeit nicht von Haus aus ins System integriert ist; eine solche Schutzfunktion ist aus meiner Sicht unerlässlich.

Dass ich mit dieser Meinung nicht allein dastehe zeigt sich auch darin, dass

einige Entwickler sich der Sache bereits angenommen haben. So erlaubt etwa das Custom ROM von Cyanogen, Apps einzelne Permissions zu entziehen. Und es gibt Apps wie XPrivacy, welche dies dynamisch erlauben. Gemäß ihres Namens versteht sich die App dabei weniger als „Berechtigungs-Manager“, sondern vielmehr als „Manager der Privatsphäre“ – was auch das Fehlen einiger Permissions in den Einstellungen erklärt.



Die Zugriffe lassen sich dabei sehr granular regeln: In der Experten-/Detail-Ansicht werden „Unbedarfte“ regelrecht überfahren von den Möglichkeiten. Zum Glück muss man nicht so tief eintauchen, wenn man nicht möchte – wie der Screenshot zeigt, ist auch eine „übersichtliche Konfiguration“ möglich. Wahlweise lassen sich Berechtigungen komplett entziehen (was jedoch zum Absturz der dies nicht erwartenden betroffenen App führen kann) – oder aber auch durch Fake-/Zufalls-Daten ersetzen: So befindet man sich bei einer Ortsabfrage etwa am Nordpol, das Adressbuch ist leer, Internet gerade nicht verfügbar, und die IMEI ist eine Zufalls-Zeichenkette.



Neben dem generellen Zugriffsverbot und der Lieferung von Fake-Daten gibt es auch die Möglichkeit, die Entscheidung von Fall zu Fall zu treffen. Dies ist beispielsweise sinnvoll wenn man prüfen möchte, ob sich eine App auch an die Regeln hält: Greift sie wirklich nur dann auf das Adressbuch zu, wenn es aus Anwendersicht nötig ist? Bei einem solchen „On-Demand Popup“ lässt sich dann nicht nur entscheiden, *ob* der Zugriff gestattet wird – man kann ihn auch auf die nächsten 15 Sekunden beschränken, was dem beschriebenen Anwendungsfall sehr entgegenkommt. Verpasst man die Abfrage, wird der Zugriff je nach Konfiguration entweder automatisch verweigert oder gewährt – sobald der Fortschritts-Balken den rechten Rand erreicht hat.

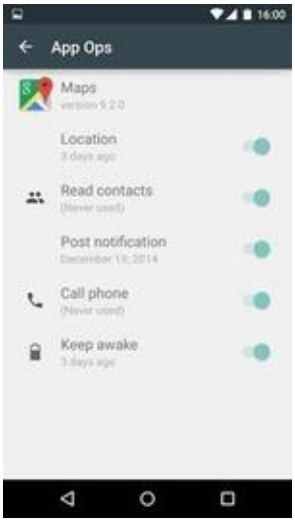
Neu installierten Apps wird per Default erst einmal alles verboten – sicher ist sicher. Wem das zu weit geht, der passt das entsprechende Template entsprechend an – und legt so selbst fest, was für neue Apps gelten soll.

XPrivacy ist Open Source (man findet es daher auch bei Github, einschließlich Dokumentation, FAQ und mehr) – was für eine Sicherheits-Anwendung (inbesondere die Privatsphäre betreffend) ein wichtiges Kriterium ist: Schließlich möchte man etwaige „Hintertürchen“ vermeiden. Einziger Wermutstropfen: Es funktioniert nur auf gerooteten Geräten, und setzt das Xposed Framework voraus. Dafür ist es aber mit Sicherheit weit flexibler, als die mit Android 6.0 endlich eingeführte Berechtigungs-Verwaltung.

Hat man ohne *root* denn wirklich nicht die geringste Chance? Ganz so ist es zum Glück nicht, denn es gibt beispielsweise noch Privacy Blocker (linkes Bild). Ohne *root* kann auch diese App natürlich nicht „in den laufenden Betrieb“ eingreifen – daher ist das Vorgehen hier ein vollständig anderes: Der Benutzer wählt zunächst die gewünschte App (die mit den unerwünschten Permissions) aus. *Privacy Blocker* greift sich daraufhin die APK-Datei dieser App, nimmt sie auseinander, und präsentiert dem Anwender die Liste der von ihr geforderten Permissions (siehe Screenshot). Hier lassen sich nun „unerwünschte“ Permissions auswählen – und schließlich wird eine neue APK-Datei erzeugt, die eben diese nicht mehr verlangt. Selbige kann man nun installieren.

Klingt soweit eigentlich ganz gut, nur hat die Sache zwei „kleine“ Haken: Zum Einen eignet sich die so erzeugte APK-Datei nicht für das Update einer bereits installierten Version (die Signatur hat sich geändert – Android hält es also für eine andere App, die dummerweise nur gleich heißt). Zum Anderen wird die App nun mit großer Wahrscheinlichkeit abstürzen, wenn sie auf eine Funktionalität zugreift, für die sie nun keine Permission mehr hat: Da der Entwickler (mit Recht) davon ausgehen konnte, dass ihm der Zugriff erlaubt wird, hat er diesen „Zugriffsfehler“ wahrscheinlich nicht abgefangen. Bei einem „bösen“ Zugriff ist dies mit Sicherheit die bessere Alternative. War der Zugriff aber doch funktional erwünscht, weiß man nun zumindest Bescheid, wozu diese Permission benötigt wird – und installiert entweder die Originalversion, oder baut sich mit *Privacy Blocker* eine weniger restriktive Fassung.

Die App ist im Playstore für ca. anderthalb Euro verfügbar. Eine gratis Testversion gibt es ebenfalls; diese beschränkt sich jedoch auf die Anzeige, geänderte APK-Dateien lassen sich damit nicht erstellen.



Eine ähnliche Funktionalität stellt SRT AppGuard zur Verfügung, welches jedoch bereits seit längerem nicht mehr im Playstore erhältlich ist. Diese App entfernt nicht die entsprechenden Berechtigungs-Anforderungen aus dem Manifest der zu behandelnden App, sondern fügt letzterer eigenen Code hinzu – und greift somit in die jeweilige App selbst ein – was der Grund für die Entfernung aus dem Playstore gewesen sein dürfte, und nicht nur rechtlich bedenklich ist.

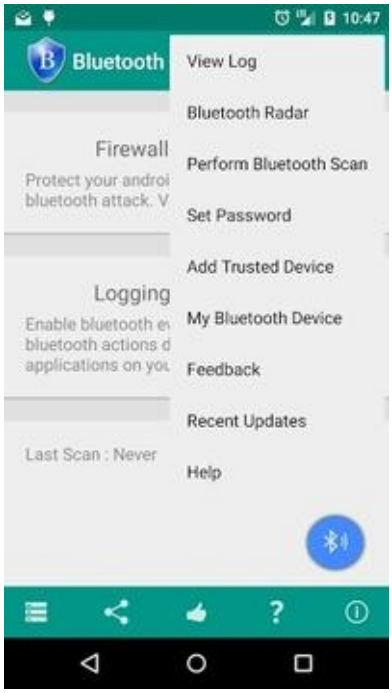
Wer auf seinem Gerät eine Android-Version zwischen 4.3 und 5.0 einsetzt, hat alternativ noch die Chance, eine versteckte Funktionalität zu nutzen: Bei diesen Versionen wurde mit *AppOps* experimentiert,

welches ab Android 6.0 nun offiziell für diese Aufgabe zuständig ist. Außer unter Android 4.4.2 wird kein root benötigt, um an diese Einstellungen heranzukommen: Eine kleine App wie App Ops macht die entsprechende Seite verfügbar. Die Möglichkeiten sind jedoch, im Vergleich zu *XPrivacy*, extrem eingeschränkt (was sie sicher auch unter Android 6.0 bleiben werden). So lässt sich etwa der Internet-Zugriff hier nicht regulieren. Ebensowenig ist meines Wissens die Verwendung von Fake-Daten oder eine „On-Demand Entscheidung“ vorgesehen.

Weitere Apps zu diesem Thema finden sich wiederum bei *IzzyOnDroid* in einer Übersicht aufgeführt.

Firewalls

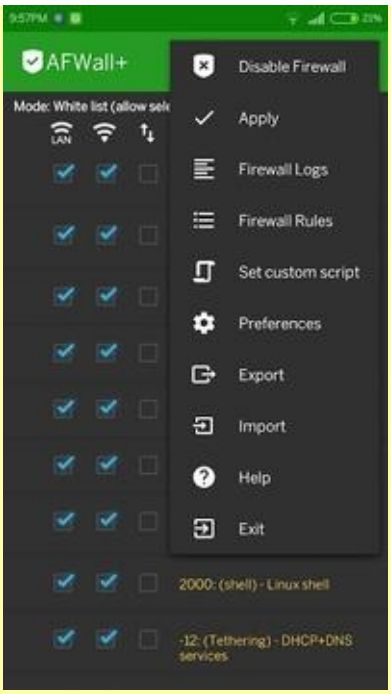
Auch sogenannte „Personal Firewalls“ fallen in die Rubrik „Zugriffsrechte“ – dienen sie doch dazu, den Zugriff auf bestimmte Dinge zu beschränken. Wieder einmal gilt die übliche Einschränkung: „Ohne root sich nichts tut“, denn so etwas greift tiefer ins System ein.



Um den Bereich „Bluetooth“ kümmert sich die rechts abgebildete Bluetooth Firewall – und benötigt dafür ausnahmsweise keinen *root*. Häh? Blauzahn-Attacken? Blauwal, auf See, klar. Aber was kann denn schon … Ho-ho-ho. Da kann. Jemand an den Einstellungen drehen, oder die Kontaktliste auslesen, oder … Ja, kann böse sein. Und was tut so eine Firewall? Sie sperrt halt alles unerwünschte aus. Oder ein. Je nachdem – eine Verbindung kann ja immer von zwei Seiten her aufgebaut werden.

Und so schlägt die *Bluetooth Firewall* beispielsweise Alarm, wenn eine App mit Bluetooth-Fähigkeiten installiert bzw. aktualisiert wurde, oder Bluetooth-Aktivitäten starten will. Ebenso aber auch, wenn jemand von außen per Bluetooth zugreifen möchte. Außerdem ermöglicht sie einen Bluetooth-Scan (welche Bluetooth-Geräte sind in Reichweite?), protokolliert alle Bluetooth-Aktivitäten, und gibt dem Anwender die Möglichkeit, ausgewählte Bluetooth-Geräte als „vertrauenswürdig“ einzustufen.

Für die Internet-Verbindungen hingegen ist AFWall+ zuständig – eine Weiterentwicklung der beliebten Firewall-App DroidWall. Genau genommen handelt es sich hier „nur um eine Eingabemaske“ zur Erfassung von Regeln – die eigentliche Arbeit erledigt iptables, welches hoffentlich bereits auf dem Androiden installiert ist – denn dies ist natürlich Voraussetzung für die Funktionalität. Aber leider nicht auf jedem Androiden gegeben – sollte *iptables* nicht vorhanden sein,



hilft daher nur ein Custom ROM.

Ein schönes Feature von *AFWall+* ist auch, dass sich der Zugriff getrennt nach Netzwerk-Interface steuern lässt: So kann man beispielsweise einer datenhungrigen App die Nutzung des „mobilen Internets“ verbieten, und sie auf WLAN-Verbindungen beschränken – während man (vermeintlichen) „Schnüfflern“ den Netzzugang komplett untersagt.

Als Nachfolger von *DroidWall* kann *AFWall+* auch deren Regeln importieren. Mehrere Profile lassen sich verwenden, Tasker/Locale werden unterstützt. System-Apps werden farblich hervorgehoben, bei Installation neuer Apps erfolgt ein entsprechender Hinweis (dass man für selbige evtl. noch Einstellungen vornehmen möchte). Und das reisst die Feature-Liste nur grob an.

Übrigens zeigt der Screenshot noch etwas anderes recht eindrücklich: Die „Zahlen“ vor den Namen der aufgeführten Apps bezeichnen die „User-ID“ derselben. Womit man sieht, was ich eingangs bereits beschrieb: Unter Android bekommt jede App eine eigene User-ID. Und über diese kontrolliert auch *AFWall+* den Netzzugriff.

Über eine Hintertür lässt sich ähnliches auch ohne root erreichen, wie die App Mobiwol zeigt: Hier wird eine VPN Verbindung genutzt. Die jeweilige App hat dann zwar prinzipiell noch Netzwerk-Zugriff – kommt aber nicht weit damit.

Malware und Viren

Zu diesem Thema gibt es ja fast jede Woche (zumindest gefühlt) einen Artikel in den News. In Bild-Deutsch gesprochen: „140% aller Android-Apps sind mit Malware und Viren verseucht!“ Oder so ähnlich. Meist kommen diese Warnungen von wohlbekannten Firmen aus dem Sicherheitssektor, mit Namen wie Kaspersky oder McAfee. Moment, woher kommen uns diese Namen bekannt vor? Richtig: Diese Firmen bestreiten ihren Umsatz mit dem Verkauf von Anti-Virus und Anti-Malware Produkten. Da ist sicher die Frage erlaubt: Wollen die nur den Umsatz ankurbeln – oder ist an den Meldungen etwas dran?

Die Wahrheit liegt, wie so oft, in der Mitte: Natürlich will man den Umsatz ankurbeln. Trotzdem handelt es sich um Fakten – es wird also nicht gelogen. Nur ein wesentlicher Teil der Wahrheit im Kleingedruckten versteckt.

Was jedoch bei genauerem Lesen mit ein wenig Hintergrundwissen (welches die vorigen Kapiteln hoffentlich aufgebaut haben) auffallen sollte. Denn was macht

diese böse Malware? Als Beispiel nennt McAfee da eine Kalender-App, die heimlich Premium-SMS verschickt. Was haben wir jedoch bereits im Kapitel Zugriffsrechte gelernt: Vor der Installation prüfen, ob Berechtigungen Sinn ergeben. Wo macht bitte die Permission zum SMS-Verschicken bei einer Kalender-App Sinn? Eigentlich hätte sie bereits an der Stelle auffallen müssen. Bei vielen der anderen genannten Beispiele verhält es sich ähnlich; es kam mir bislang kein Beispiel unter, welches sich nicht in dieses Muster eingepasst hätte. Und leider gehen die Sicherheits-Apps der namhaften Hersteller, was das Anfordern von Permissions betrifft, nicht gerade mit gutem Beispiel voran – schon mehrfach musste ich dabei denken: Liebe Leutz, nach Euren eigenen Worten dürfte ich Eure App gar nicht installieren … Und wie um noch eins obendrauf zu setzen, schreibt TelTarif: Smartphone-Viren sind oft nur heiße Luft. *Man kann sagen, dass die Antiviren-Software-Hersteller ein großes Interesse haben, die Gefahr groß aussehen zu lassen, manchmal auch größer als sie wirklich ist* (ebenda).

GMV

Womit ich bei einem meiner Lieblings-Themen angekommen bin: Seit ich GMV zum ersten Mal in einem Post bei AndroidPIT und dann auch in meinem eBook genannt habe, bekam ich des Öfteren Anfragen: Wo könne man diese Android-App denn herunterladen? Auch Google-Suchanfragen nach „GMV App Android“ fanden schon den Weg auf meinen Server. Dabei braucht man doch nur dem Link zu folgen!

Also noch einmal im Klartext: Bei GMV handelt es sich *nicht* um eine Android-App. GMV sollte im *biologischen Speicher* (auch als „Brain 2.0“ bekannt) *vorinstalliert* sein. Die Abkürzung steht nämlich für „**G**esunder **M**enschen**v**erstand“. Und den hat doch hoffentlich jeder, zumindest in gewissem Umfang. Natürlich kann man sich alle möglichen Anti-Viren und Anti-Malware Apps auf seinem Androiden installieren – doch das Einzige, was damit garantiert ist, sind belegter Speicherplatz und Ressourcen-Verbrauch. Keiner der Hersteller wird garantieren, dass damit die Gefahr zu 100% gebannt ist, dass jeder Schädling erkannt sowie erfolgreich „abgeschossen“ und entfernt wird. Im Gegenteil las ich in einem Forum von einem Selbstversuch: Jemand installierte sich eine dieser Anti-Virus-Anti-Malware Apps, und versuchte anschließend eine Warnung zu provozieren – doch die gab keinen Laut, obwohl er sich richtig böse klingende Sachen herausgesucht hatte (leider finde ich den Post nicht mehr – ersatzweise lohnt aber auch ein Blick in diesen Blog-Eintrag).

Auf was sollte man also bei der Installation einer App achten? Ich zitiere hier einmal ganz dreist aus meiner Androiden-Fibel:

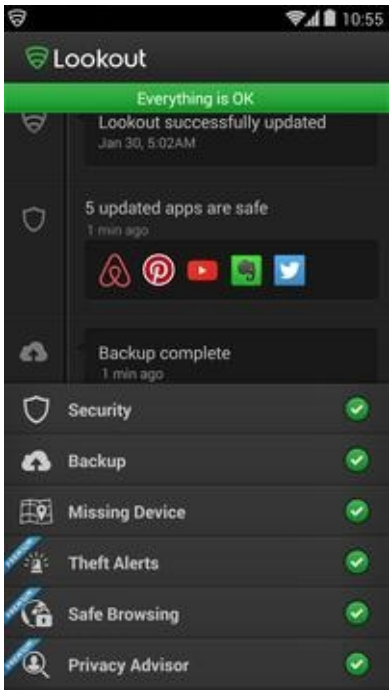
- Ist die Quelle vertrauenswürdig?
  - Positiv-Beispiele: AndroidPIT-Market, AppCenter, Google Play Store,

Website des bekannten (!) Entwicklers
    - Negativ-Beispiele: Bei Rapidshare „gefunden“, in einer Tauschbörse aufgetrieben, per eDonkey aus unbekannter Quelle gezogen …
- Sehen die Permissions vernünftig aus?
    - Positiv-Beispiele: Ein Webbrowser muss ins Web, eine SMS-App kann natürlich SMS lesen/schreiben/schicken und braucht ggf. auch (lesend) Zugriff aufs Adressbuch
    - Negativ-Beispiele: Eine Wallpaper-App braucht in der Regel keine Telefonnummern anrufen, ein Ballerspiel muss keine SMS senden.
    - Besondere Vorsicht: Apps, die auf persönliche Daten (Kontakte, Kalender, Nachrichten) zugreifen und gleichzeitig ins Internet wollen. Leider lässt sich bei letzterem (Internet) die Frage der Notwendigkeit nicht so einfach beantworten – es könnte auch einfach nur für Werbung-Laden gebraucht werden…
- Was sagen andere Nutzer zur App/zum Entwickler (Bewertungen, Forum)?
    - Auch hier wieder GMV aktivieren. Kommentare wie „Geil!“, „Super“, etc. sagen nicht wirklich etwas aus (da hat eher jemand bei deaktiviertem GMV einen Kommentar hinterlassen)
    - Gleiches gilt für manchen negativen Kommentar: Nicht gerade selten passiert es, dass jemand einfach zu blöd war. Oder die Anforderungen der App gar nicht verstanden hat.
    - Nicht alle Bewertungen beziehen sich wirklich auf die App. Die kann schließlich nix dafür, wenn der Playstore mal wieder klemmt, und daher der Download nicht funktioniert. Oder die HD-Video-App, die mindestens WVGA benötigt, mit dem Motorola Flipout (mini-Display) im Playstore nicht gefunden wird.
    - Ganz neue App? Noch keine Bewertungen? Im Zweifelsfall im Forum nachfragen, ob schon jemand die App kennt und etwas dazu sagen kann.

Wer diese Punkte beachtet, hat eigentlich bereits mehr als die „halbe Miete“ eingefahren. Dann noch Vorsicht mit diversen Werbebannern, die einen mit Abos „beglücken“ wollen – und ein 90%iger Schutz sollte gegeben sein. Natürlich können andere Apps aus der „Sicherheits-Abteilung“ eine gute Ergänzung zu GMV bieten. Insbesondere bei Verlust des Gerätes – denn dagegen macht auch GMV nicht immun.

Rundum-Sorglos-Pakete

Die sollen einen wohl in Sicherheit wiegen – versprechen sie doch, sich um alles zu kümmern: Viren, Malware und Diebe sollen gleichermaßen eins auf die Mütze bekommen. Interessant ist hier: Nach einem Blick auf die verlangten Permissions müssten viele dieser Apps eigentlich gleich selbst draußen bleiben.

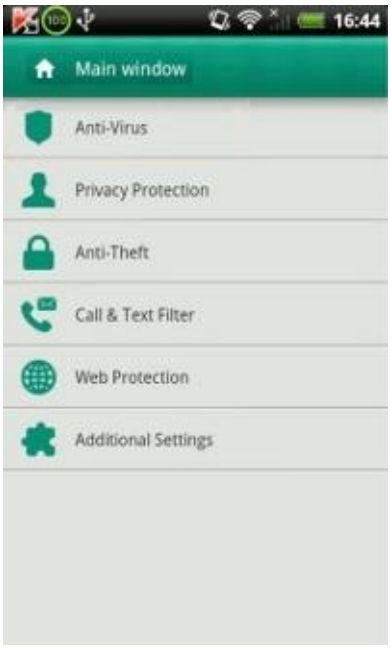


Wozu bitte muss Lookout Security Antivirus denn bitte Synchronisierungseinstellungen lesen/schreiben, Kontaktdaten lesen/schreiben, Kalenderdaten schreiben, Benutzerwörterbuch lesen/schreiben … Das Ding soll sich doch um die Abwehr der „bösen Jungs“ kümmern, und nicht selbst an diesen persönlichen Dingen herumbasteln! Das kommt mir vor, als wolle man den Teufel mit dem Beelzebub austreiben.

Wer jetzt glaubt, die Antwort darauf in der App-Beschreibung zu finden: Fehlanzeige, über Jahre hinweg. Die Mühe, die ganze Website hinter dem angegebenen Link zu durchsuchen, habe ich mir nicht gemacht – zumindest in den dortigen FAQs fand sich auch keine Antwort. Mittlerweile hat man aber wohl mein Buch gelesen: In der App-Beschreibung findet sich nun ein kurzer Text („… um die zugehörigen Informationen von einem gestohlenen Gerät entfernen zu können“ – aha, dafür liest man Sync-Settings?), und etwas ausführlichere Informationen auf der Website. Ein Durchschnitt von 4,4 Sternen bei über 800.000 Bewertungen stimmt hingegen beruhigend: Offensichtlich wird mit den genannten Permissions wirklich nichts „böses“ angestellt (oder es hat einfach noch niemand gemerkt, wofür die damit gesammelten Daten verwendet werden). Allerdings sollten sich die Entwickler solcher Software auf die Fahnen schreiben, hier mit gutem Beispiel voran zu gehen, anstatt die Anwender zu verunsichern – *Lookout* ist in dieser Hinsicht kein Einzelfall.

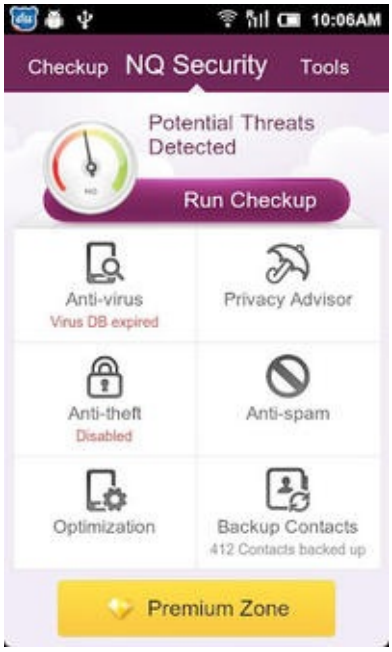
Davon abgesehen klingt *Lookout* eigentlich recht vielversprechend: Jede heruntergeladene App wird hier auf „Schädlingsbefall“ geprüft. Zusätzlich kann man „Sicherheits-Komplettscans“ seines Gerätes einrichten, die automatisch zum angegebenen Zeitpunkt (etwa wöchentlich) starten.

Das Gerät ist abhanden gekommen? Verlegt oder gestohlen? Per Google Maps lässt es sich lokalisieren (auch bei abgeschaltetem GPS; es wird dafür jedoch ein Account bei myLookout.com benötigt). Auch ein lauter Alarm vom Gerät selbst kann ausgelöst werden – sogar wenn es gerade auf „lautlos“ steht (da eignet sich bestimmt eine MP3, wo jemand laut „DIEBE!“ schreit …). Als Dreingabe gibt es noch Backup und Restore wichtiger Daten, was Grund für einige der o. g. Permissions sein dürfte. Das Schöne an der Sache: All dies ist gratis.



Auch die Großen im Bereich Antivirus & Co. sind hier natürlich vertreten – wie etwa Kaspersky mit seiner Kaspersky Internet Security oder Symantec mit der Norton Mobile Security. Beide dürfen sich die gleichen Vorwürfe bezüglich der verlangten (und nicht begründeten) Permissions machen lassen – mit dem Zusatz, dass sie diese bis heute nicht erklärt haben. So möchte die Norton-Suite u. a. gern lesend und schreibend auf die Kontaktdaten zugreifen, und so weiter. Bei Kaspersky sieht es nicht besser aus: Auf Kontaktdaten und Kalender möchte die App lesend und schreibend zugreifen (sicher für den Vermerk, wer das Gerät wann gestohlen hat). Auch will diese App an den Synchronisierungseinstellungen und APN-Einstellungen basteln. Bei einer solchen Kombination läuten eigentlich bei mir sämtliche Alarmglocken und schreien: Vorsicht, Malware!

Was aber bringt Eugen Kaspersky uns nun mit seiner App – bzw. was verspricht er uns als Gegenleistung für die ca. zehn Euro Kosten? Echtzeit sowie „On-Demand“ Scans nach Viren (die es unter Android gar nicht gibt) und Malware, Privacy für Kontakte (bestimmte Kontakte markieren, sodass keine Spuren der Kommunikation mit ihnen zu sehen sind – aha, das erklärt zumindest die Zugriffe auf die Kontaktliste), Anruf- und SMS-Filter, Diebstahl-Schutz (Lock, Wipe, Anzeige einer Nachricht wie z. B. „Ich wurde verloren/gestohlen, bitte xxx anrufen“, Anzeige der Location in Google Maps, sowie Email-Alert mit neuer Rufnummer bei SIM-Wechsel – sofern der Dieb nicht schlau genug war, vorher einen Wipe zu machen).



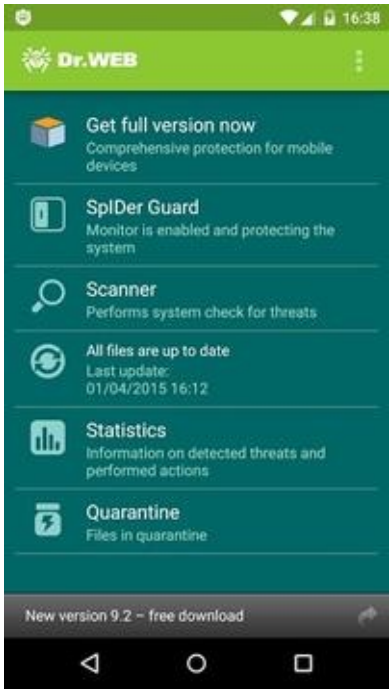
NetQin Security & Anti-virus gehört ebenfalls hierher – bietet es doch gratis nicht nur Schutz vor Viren und Malware, sondern, wie der Screenshot zeigt, auch einen Diebstahlschutz und sogar einen Traffic-Manager. Hinzu kommt noch ein System-Monitor mit Anzeige der Speicherauslastung und Taskkiller sowie „Geräte-Optimierung“ mit der tollen Ansage „Increase efficiency by closing apps that run in the background without your knowledge“. Ich hoffe, das ist lediglich als Vorschlag gemeint – und beschreibt nicht etwas, was die App ohne Rückfrage selbst tut. Ach ja: Ein Monitor für Zugriffe auf private Daten ist ebenfalls mit dabei, „böse Apps“ werden zur Deinstallation gezwungen („force uninstall“), und obendrein kann man auch die Kontakte sichern und wieder herstellen.

Dem Umfang der App entsprechen auch die wieder einmal nicht begründeten Permissions. Wer alles können will, muss natürlich viel anfordern – doch bei

einigen Dingen sollte man dennoch angeben, wozu dies genutzt wird. Da fordern Kaspersky & Co. in ihren Sicherheitsberichten doch explizit auf, man solle nichts installieren, was suspekte Permissions fordert – und dann liefern sie gleich selbst die Beispiele dafür? Wozu möchte beispielsweise *NetQin* Systemeinstellungen oder UI-Einstellungen ändern, ausgehende Anrufe umleiten, sowie lesend und schreibend auf die Kontaktdaten zugreifen? Genau so stellt man sich die Permissions von Malware vor – hier täte eine genauere Beschreibung dringend Not!

Doch davon einmal ab: Was verspricht *NetQin* denn nun zu leisten? Die Bereiche habe ich ja bereits grob umrissen: Die App kümmert sich um Viren, Malware und Diebstahl-Schutz. Ersteres sowohl in Echtzeit, als auch als „Device-Scan“. Für letzteres ermöglicht es die Anzeige der gegenwärtigen Position des Gerätes auf der Karte, eine Sperrung des Gerätes, das Auslösen eines lauten Alarms, und auch das Löschen aller persönlicher Daten. Hinzu kommen noch Tools zum Sichern persönlicher Daten, Memory-Booster, Task-Killer, Traffic-Monitor, File-Manager … Da stellt sich die Frage, was die App eigentlich nicht machen soll. So schön es ist, eine eierlegende Wollmilchsau zu haben – besteht da nicht ein wenig die Gefahr, sich zu verzetteln? Sollte man sich nicht besser auf das Kerngebiet konzentrieren? Doch dieses Urteil überlasse ich besser denen, die die App nutzen.

Anti-Virus und Anti-Malware



Darum geht es in diesem Kapitel ja eigentlich: Malware (nicht verwechseln mit Paintshop & Co.), Viren, und die dazugehörigen Gegenmittel. Mensch holt sich bei Virenbefall (oder vorbeugend) eine Impfung beim Doktor – und Androide greift beispielsweise zu Dr.Web Anti-virus. Endlich mal eine App, die auch bei den Permissions mit gutem Beispiel vorangeht! Zwar wird auch hier nicht erklärt, wozu die App z. B. die „Netzwerkkonnektivität ändern“ muss – doch damit hat es sich auch bereits: Nur vergleichsweise wenige Permissions werden angefordert, und diese sind (mit genannter Ausnahme) durchaus nachvollziehbar. Zudem sind App und Hersteller bereits aus dem PC-Bereich bekannt, und auch durchschnittliche 4,5 Sterne bei über 900.000 Bewertungen sind geeignet, Vertrauen zu erwecken.

Und was hat der Herr Doktor studiert, was kann er? Die gratis-Version scannt einfach auf „böse Dateien“, und sperrt diese in die „Quarantäne“. Hierbei scheint sowohl ein Echtzeit-Scan zu erfolgen – als auch die Möglichkeit zu einem „On-

Demand-Scan“ zu bestehen. Außerdem lässt sich noch einstellen, dass die SD-Karte ebenfalls bei jedem Einbinden geprüft werden soll. Die Vollversion bietet dazu eine Filterung eingehender Anrufe und SMS, inklusive Blacklist (z. B. für nervige Werbe-Anrufer und Spam-SMS), sowie einen Diebstahl-Schutz.

Hinter Anti-Virus Free steht die auch vom PC-Anti-Viren-Markt her bekannte Firma AVG. Und bei der App beschränkt man sich schon längst nicht mehr auf das, was der Name suggeriert: Mit an Bord sind auch App-Locker, Task-Killer, Backup, Diebstahlschutz … Kurzum: Auch AVG möchte einen Alleskönner präsentieren.

Was der definitiv kann, ist jede Menge Permissions anzufordern – natürlich wiederum ohne diese zu erklären. Offensichtlich eine verbreitete Unsitte bei allen Apps in diesem Segment (mit wenigen Ausnahmen): Ich habe einen Namen, ich darf das. Auch hier darf sich der Anwender wieder vergeblich fragen: Wozu bitte will diese App Systemeinstellungen ändern, Synchronisierungseinstellungen schreiben, abonnierte Feeds schreiben, Kontaktdaten lesen und schreiben (da steckt vermutlich das Backup dahinter), das Benutzer-Wörterbuch schreiben (aber nicht lesen?), Kalenderdaten schreiben (aber nicht lesen – hier steht also kaum das Backup als Grund) …

Nun gut: Der Name ist bekannt, die Firma auch. Die Anzahl der Bewertungen liegt im 7-stelligen Bereich, der Schnitt ist über 4 Sternen angesiedelt – alles Dinge, die geeignet sind, die gerade aufgeworfenen Zweifel ein wenig zu zerstreuen. Schauen wir uns also einmal an, was die App zu können meint:

Es wird ein Echtzeit-Scan angeboten, der auch SMS (huch?) und MMS einschließt. Dazu kommt ein Backup von Kontakten, Anrufprotokollen, Lesezeichen, Text- oder Medianachrichten auf die SD-Karte, Handyortung (durch SMS ausgelöst) sowie „Alarmton“ (auch wenn auf lautlos gestellt), Task-Killer, App-Locker (installierte Anwendungen gegen unbefugte Benutzung sperren), sowie der komplette Wipe. Hier als „lokaler Wipe“ beschrieben. Keine Erwähnung von Benachrichtigung bei SIM-Wechsel, oder der Möglichkeit, den Wipe remote auszulösen – aber auch diese Funktionalität dürfte enthalten sein.

Aggressive Werbung

Klar müssen die Programmierer von etwas leben. Leider sind die meisten Anwender jedoch nicht bereit, für eine gute App auch nur einen Euro auszugeben – weshalb sich Entwickler nach anderen Möglichkeiten umsehen müssen. Betreibern von Werbenetzwerken ist dieser Umstand sehr wohl bewusst – und so

liefern sie fertige Module, welche die Entwickler lediglich noch in ihre Apps integrieren müssen.

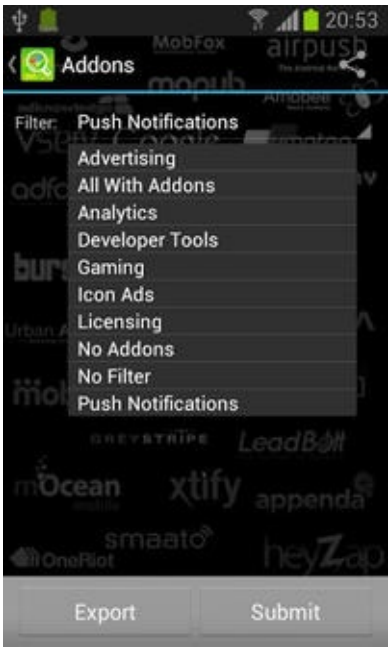
Was auf den ersten Blick nach einer typischen Win-Win Situation aussieht, hat jedoch durchaus seine Tücken:

- Um mehr Einnahmen zu generieren, greifen einige dieser Module auf persönliche Daten des Anwenders zu: Offiziell für „zielgerichtetere Werbung“ – de facto aber u. U. auch für den Datenhandel. Im Extremfall wird da gleich einmal das komplette Adressbuch heruntergeladen, und die Anrufliste gleich mit.
- Ein solches Werbe-Modul erbt automatisch alle Berechtigungen, welche der App selbst zugeteilt werden – was einige Kandidaten teilweise recht schamlos ausnutzen.
- Werbung muss als solche klar erkennbar sein – aber nicht jedes Werbe-Modul hält sich daran. So taucht dann ggf. eine Benachrichtigung auf, die sich als vermeintliche Optimierungs-Maßnahme des Systems ausgibt (siehe Screenshot), um den Anwender zu Downloads zu verleiten. Unnötig zu sagen, dass diese Downloads nicht unbedingt vorteilhaft für den Benutzer sind. Andere unduldbare Platzierungen sind etwa Lesezeichen im Browser, oder als Apps getarnte Icons auf dem Homescreen.

Einer der bekanntesten Vertreter dieser Gruppe ist sicher AirPush. Oder besser war – denn mittlerweile verbieten die Richtlinien für Google’s Playstore diese Art aufdringlicher Werbung. Was nicht heißt, dass sich keine derartigen „Bösewichte“ mehr im Playstore finden lassen. Dem Entwickler der eigentlichen App ist dies vielleicht nicht einmal bewusst: Das Modul wurde ja lediglich integriert, damit die App wenigstens eine kleine Aufwands-Entschädigung abwirft. Das beschriebene Verhalten (welches natürlich nicht auf alle Werbemodule zutrifft) ist eher das eines typischen Trojaners – womit diese Art von Werbung aus meiner Sicht definitiv in die Kategorie „Malware“ fällt.

Ein Blick auf den Airpush-Screenshot zeigt die Perfidität: Ist das jetzt eine System-Meldung, die mir Tipps für bessere Einstellungen für optimalere Akku-Laufzeit geben will? Mitnichten! Über einen „Klick“ auf derartige Einträge hat sich schon mancher Ungewolltes eingehandelt. Das Gemeine daran: Es ist nicht ersichtlich, woher das kommt – etwa welche App dafür verantwortlich ist. Mit Android 4.1 wurde daher ein neues Feature eingeführt, das erlaubt, durch langes Drücken auf eine Benachrichtigung den „Übeltäter“ zu entlarven – und ihm das „Übeltun“ zu untersagen. Das betrifft jedoch nur die Anzeige von Werbung in der Benachrichtigungsleiste; wie aber kann man die restlichen Kandidaten ermitteln?

Natürlich muss niemand in die Röhre gucken. Recht bald nach Auftauchen dieser Schädlinge fand sich auch Software, welche diese aufspüren konnte. Obwohl der Begriff „AirPush“ oft als Synonym für diese Art unlauterer Werbung verwendet wird, gibt es schließlich noch weitere Netzwerke dieser Art. Und



glücklicherweise auch kleine Helferlein, die uns bei deren Identifikation unterstützen. Ein Beispiel dafür ist der Addons Detektor von Denper. Wie im Screenshot bereits erkennbar, spürt diese App nicht nur AirPush-Trojaner auf – sondern ebenso andere Advertiser, Analytics-Tools (die auch gern einmal persönliche Daten mitnehmen), sowie eine ganze Reihe weiterer (in der Regel jedoch „unschädlicher“) Bibliotheken.

Damit man aber nicht erst warten muss, bis man sich bereits „infiziert“ hat, gibt es auch Vorsorge-Apps. So lässt sich etwa mit dem AppBrain Ad-Detector schon vor der Installation feststellen, welche App eventuell verseucht ist. Passenderweise wird das zusammen mit den verlangten Berechtigungen angezeigt. Wer ohnehin bereits auf die AppBrain Market App als gute und schnelle Alternative zur überladenen Playstore-App setzt, kann diesen Ad-Detector dort integrieren – so ist dann alles passend zusammengefasst: Neue App gesucht, direkt geprüft, sofort entsorgt – und eine saubere Alternative gefunden sowie installiert.

Natürlich gibt es auch zu diesem Thema wieder eine passende Übersicht bei *IzzyOnDroid*, in der sich weitere Alternativen und Tipps finden. Ebenfalls ansehen sollte man sich die Liste mit weiterführenden Links am Ende derselben.

Diebstahlschutz

Kann eine App wirklich vor Diebstahl schützen? Sicher nur bedingt. Dafür müsste sie den Dieb „in flagranti außer Gefecht setzen“: Stromstoß, Tränen- und Nervengas, Keule auf die Rübe … Menschenrechts-Kommission vor der Tür. Geht so nicht. Abgesehen von der schwierigen technischen Umsetzung.

Dass das doch zumindest ansatzweise geht, zeigt u. a. Anti Theft Alarm: Hier soll der Dieb mit Sirenengeheul in die Flucht geschlagen werden. Der Krawall geht dabei los, sobald man das Gerät bewegt (bzw. das Ladekabel trennt), ohne rechtzeitig den Alarm-Code einzugeben. Eine Sache, die sich sicher auch mit Tasker umsetzen ließe – eventuell sogar mit einer Gesten-Deaktivierung (einmal kurz schütteln, einmal drehen: Deaktiviert).

Die meisten Anti-Theft Apps sind eher die „Pille danach“ – sie greifen erst dann

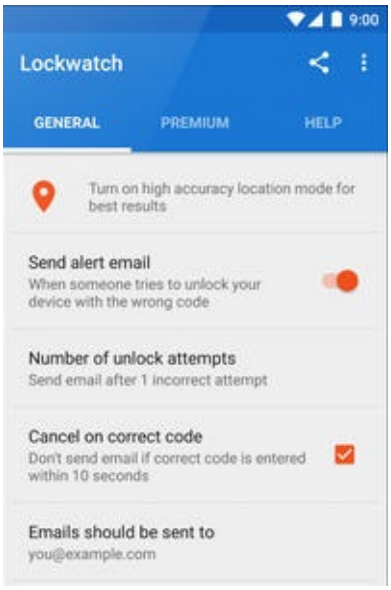
ein, wenn das Kind in den Brunnen gefallen und der Androide verschwunden ist. Die dann angebotenen Möglichkeiten fallen sehr unterschiedlich aus; vereinfacht ausgedrückt, reichen sie von einfachen Lokalisierungshilfen bis hin zur Selbstzerstörungs-Automatik.

Eindeutig in die erste Kategorie gehört das links abgebildete Wheres My Droid: Ist der Droide verschwunden, benötigt man eben mal sein Zweitgerät, das Telefon eines Freundes, oder einer beliebigen anderen Person. Damit schickt man sodann eine SMS mit dem „Attention Word“ (also dem passenden Code, den man zuvor konfiguriert hat) an das verschwundene Gerät – und es antwortet per SMS mit den Koordinaten. Ein Blick in Google Maps – aha, da isses (gerade)! Ein anderes „Attention Word“, und das Gerät macht mit ordentlich Krach auf sich aufmerksam. Auch wenn es gerade „ruhig gestellt“ (silent mode) war.

Das setzt natürlich voraus, dass die eigene SIM-Karte noch im Gerät steckt (sonst läuft die SMS ja ins Leere). Aber per EMail klappt es auch, und das sollte nach SIM-Wechsel auch noch tun. Bei selbigem benachrichtigt diese App auch; in der Kaufversion ist aus der Ferne sogar eine Gerätesperre, das Auslösen der Kamera sowie ein „Wipe“ persönlicher Daten möglich.

WatchDroid Lite kann dies auch, bietet aber zusätzlich einen „Stealth Mode“ (Tarn-Modus), damit der „Finder“ die App nicht einfach löscht. Krach schlagen und SMS mit GPS-Daten verschicken sollte damit kein Problem sein. WatchDroid Pro bietet für etwa anderthalb Euro einen Mehrwert dazu: Das Telefon lässt sich nun auch aus der Ferne sperren, oder komplett löschen (Wipe). Auch ein eventueller SIM-Karten-Wechsel wird von der Pro-Version erkannt – und mit dem Versand einer SMS an den hinterlegten Empfänger beantwortet. Der Trend geht eindeutig zum Zweitgerät …

Ebenfalls interessant klingt auch der Ansatz von Lockwatch – zumindest wenn der Androide über eine Front-Kamera verfügt. Bei Falscheingabe des Entsperrcodes/Entsperrmusters schickt diese App ein Foto des „Täters“ inklusive einem Google-Maps-Link seiner gegenwärtigen Position per Mail. Die Vollversion benachrichtigt auch bei Wechsel der SIM-Karte, schickt eine Email nach einem etwaigen Reboot, macht mehrere Fotos (3 Stück im Abstand von etwa einer Sekunde), und schließt eine



Tonaufnahme mit ein (welche den etwaigen Kraftausdruck des „neuen Besitzers“ beinhalten könnte).

Dann wären da natürlich noch die als Rundum-Sorglos-Pakete aufgeführten Apps, die sich ja „mal so eben nebenbei“ auch noch mit um den Diebstahl-Schutz kümmern wollten. Und natürlich gibt es auch hier wieder eine Übersicht bei *IzzyOnDroid*, die noch eine ganze Reihe weiterer Apps zum Thema kennt.

App-Sicherheit

Nachdem nun etliche Sicherheits-Apps besprochen wurden, wird es Zeit, das Wort einmal umzudrehen: Wie steht es um die App-Sicherheit? Denn was nützen all die tollen Sicherheits-Apps, wenn die Anwendungen selbst unsicher sind? Das beste Sicherheits-Schloss ist sinnlos, wenn man den Schlüssel auf dem Fensterbrett ablegt, und dann auch noch das Fenster offen stehen lässt.

In diesem Kontext gilt es, mehrere Bereiche zu betrachten: Zum Einen geht es um die auf dem Gerät gespeicherten Daten – und zum Anderen sind da die Daten, die über diverse Netzwerke übertragen werden. Werfen wir also einmal einen Blick auf beide Themen.

Gespeicherte Daten

Es soll ja vorkommen, dass jemand „zufällig“ ein Android-Gerät „findet“ (manchmal auch in der Tasche der Jacke, die der Eigentümer gerade trägt). Dieser „Finder“ kann nun lediglich daran interessiert sein, die „gefundene“ Hardware zu Geld zu machen – oder aber auch, die darauf befindlichen Daten zu nutzen. Besonders interessant sind dabei natürlich Benutzernamen und Passwörter, aber auch andere persönliche Daten – denn beide lassen sich vorzüglich zum Identitäts-Diebstahl nutzen. Um „im Namen des Eigentümers“ Einkäufe zu tätigen, oder Spam zu versenden, um nur zwei Beispiele zu nennen. Daher sollten diese Daten – wenn überhaupt – auf sichere Weise auf dem Gerät gespeichert werden.

Wie so etwas sicher geschehen kann, dafür gibt es mehrere Ansätze, die zumeist etwas mit Verschlüsselungs-Techniken zu tun haben. Absolut klar sollte jedoch sein: Passwörter im Klartext (also ohne zusätzliche Hilfsmittel lesbar) auf dem Gerät zu speichern, ist ein absolutes No-Go – selbst die Verwendung einer einfachen „Cäsar-Chiffre“ (siehe Texte verschlüsseln) bietet da schon wesentlich höhere Sicherheit, auch wenn sie alles andere als sicher ist: Auch „Daten-Diebe“ sind von Natur aus faul; Klartext-Passworte sind ohne großen Aufwand nutzbar, für alles andere muss zunächst der Algorithmus ermittelt werden.

Wer jetzt meint, das wäre doch selbstverständlich – der lese bitte weiter. Denn offensichtlich ist dem nicht so. Zur Ermittlung einiger Schwachstellen habe ich ein Nandroid-Backup meines Motorola Milestone 2 hergenommen, welches ich gleich nach dem Rooten und der Installation eines Custom Recovery Images erstellt habe (Stock Android 2.2.2). Daraus habe ich die `/data` Partition extrahiert, und bin die Verzeichnisse durchgegangen. Da gibt es Erstaunliches zu entdecken …

System-Daten

Beginnen wir bei „des Pudels Kern“: Dem Android System. Dies sollte ja gerade in diesem Punkt ein großes Vorbild sein – was auch weitgehend zutrifft. Finden sich hier etwa sensible Daten im Klartext? Leider ja.

Ein Beispiel, das ich hier aufführen möchte, ist die Datei `/data/misc/wifi/wpa_supplicant.conf`. Hier werden die konfigurierten WLAN Accesspoints gespeichert. Ein Ausschnitt dieser Datei sieht (auch noch unter Android 4.4) etwa wie folgt aus:

```
network={
        ssid=“platzhotel”
        psk=“hotel123”
        key_mgmt=WPA-PSK
        priority=46
}
```

Da gibt es also einen WLAN-Accesspoint namens „platzhotel“, das mit WPA verschlüsselt ist. Der zugehörige Schlüssel lautet: „hotel123“. Bummer! Ja, tatsächlich: Alles im Klartext (Anmerkung: Die Daten habe ich natürlich leicht angepasst, um niemanden in Bedrängnis zu bringen). Hier lässt der Name des Netzes zudem noch leicht auf die Institution schließen: Es handelt sich offensichtlich um ein Hotel, welches den Namen “Platzhotel” trägt. Ein deutscher Name, was die Location zusätzlich eingrenzt. Hat der Angreifer zusätzlich die Möglichkeit zu sehen, wo der Eigentümer zuletzt unterwegs war (etwa über den entsprechenden Verlauf im Google-Account, oder über entsprechende Facebook-Posts) – findet er mit etwas Glück auch leicht den passenden Platz.

Im Falle des WLANs eines Hotels oder Cafés ist dies vielleicht noch kein „großes Ding“, da die Betreiber ihren Gästen den Zugang meist auf simple Anfrage mitteilen. Die genannte Datei speichert aber auch die Zugangsdaten für sensiblere Firmen-Netzwerke, so genutzt – und spätestens hier wird so etwas zum Sicherheits-Risiko!

Daten von Hersteller-Apps

Hersteller und Netzbetreiber „beglücken“ uns ja mit weiteren vorinstallierten Apps. Manche davon sind auch wirklich für die breite Masse nützlich – andere möchte der Anwender viel lieber loswerden (siehe Bloatware). In die erste Kategorie gehört u. a. auch eine App zum Empfang, Lesen, und Versand von Mails – die sich möglichst mit vielen Anbietern verträgt.

Hierfür sorgt u. a. auch HTC bei seinen Geräten, und die App ist recht beliebt. Insbesondere auch bei denjenigen, die etwa auf ihre Firmen-Mails auf dem Exchange-Server zugreifen möchten. Und hier lauert er wieder, unser „schwerwiegender Systemfehler“ namens „Zugangsdaten im Klartext“: Account, Passwort, und sogar Domain-Daten lassen sich so direkt einsehen! Wie das an einem Beispiel aussieht, darauf geht die „Verlags-Edition“ dieses Buches ein.

App-Daten

Wenn sich schon beim System solche Lücken finden, überrascht es sicher kaum, dass so etwas auch bei Apps auftritt. Ein wirklich extremes Beispiel liefert etwa die populäre App Skype, wie ein Artikel bei AndroidPolice.COM ausführlich beschreibt.

Nicht nur speichert die App den Benutzernamen, Profildaten, Chat-Protokolle, und mehr unverschlüsselt in diversen SQLite-Datenbanken in seinem Daten-Verzeichnis (unterhalb von `/data/data/com.skype.merlin_mecha/`) – sondern sie hebelt auch gleich noch sämtliche vom System vorgesehene Sicherheitsmaßnahmen aus. Eigentlich haben nämlich nur die App selbst und natürlich root Zugriff auf diese Verzeichnisse. Skype legt dies jedoch anders fest, und macht sie für „alle Welt“ lesbar und schreibbar! Beispiele aus dem verlinkten Artikel:

```
# ls -l /data/data/com.skype.merlin_mecha/files/shared.xml
-rw-rw-rw- app_152  app_152     56136 2011-04-13 00:07 shared.xml

# grep Default /data/data/com.skype.merlin_mecha/files/shared.xml
      <Default>jcaseap</Default>
```

Damit lässt sich der Name des Anwenders ermitteln – und somit auch der Name des Verzeichnisses, in dem seine Daten gespeichert wurden. Aber auch ohne diesen Schritt wäre es zu finden gewesen: So viele Verzeichnisse gibt es hier nicht, und alles steht ja offen wie ein Scheunentor. Werfen wir also einen Blick auf die Daten dieses Benutzers:

```
# ls -l /data/data/com.skype.merlin_mecha/files/jcaseap
-rw-rw-rw- app_152  app_152    331776 2011-04-13 00:08 main.db
-rw-rw-rw- app_152  app_152    119528 2011-04-13 00:08 main.db-journal
-rw-rw-rw- app_152  app_152     40960 2011-04-11 14:05 keyval.db
-rw-rw-rw- app_152  app_152      3522 2011-04-12 23:39 config.xml
drwxrwxrwx app_152  app_152           2011-04-11 14:05 voicemail
-rw-rw-rw- app_152  app_152         0 2011-04-11 14:05 config.lck
-rw-rw-rw- app_152  app_152     61440 2011-04-13 00:08 bistats.db
drwxrwxrwx app_152  app_152           2011-04-12 21:49 chatsync
-rw-rw-rw- app_152  app_152     12824 2011-04-11 14:05 keyval.db-journal
-rw-rw-rw- app_152  app_152     33344 2011-04-13 00:08 bistats.db-journal
```

Hier gibt es also eine SQLite-Datenbank mit dem Namen `main.db`. Klingt verlockend. Und lässt sich mit einem beliebigen SQLite-Client öffnen – es darf ja jeder zugreifen, Dank der tollen Zugriffsrechte. In dieser Datenbank findet sich u. a. eine Tabelle namens `accounts` – wenn das nicht interessant klingt! Und was da alles schönes drin gespeichert ist: Kontostand, vollständiger Name, Geburtsdatum, Anschrift, Telefon, Mail-Adresse, Website, Biografie, und mehr – kurzum alles, was man seinem Skype-Account anvertraut hat.

Skype ist den Hinweisen glücklicherweise nachgegangen, und hat den Bug recht schnell behoben. Das kann in anderen Fällen durchaus anders aussehen.

Übertragene Daten

Auch hier gilt wieder: Wer bei Problemen mit unverschlüsselt übertragenen Daten nur an Apps von „irgendwelchen kleinen Drittanbietern“ denkt: Fehlanzeige. Bis etwa Mitte 2011 taten sich hier insbesondere Facebook und sogar Google hervor, die zur Datenübertragung statt auf sicheres HTTPS auf unverschlüsseltes HTTP setzten – und zwar auch für Anmeldedaten. War das Gerät dann auch noch in einem „offenen WLAN“ eingebucht, war es für potentielle Angreifer ein einfaches, diese Zugangsdaten abzugreifen – indem sie sich im selben WLAN anmeldeten, und beispielsweise einen Sniffer einsetzten.

Zumindest haben die beiden Genannten diese Sicherheitslücke geschlossen. Heißt es nun also „generelles Aufatmen“? Keineswegs! Denn dieses Problem taucht leider immer wieder auf. Ein weiteres Beispiel ist die beliebte App WhatsApp Messenger, die bis Mitte 2012 das gleiche tat. Auch hier wurde die Lücke zum Glück bereits geschossen. Doch wer weiß, wo sie sich als nächstes zeigt? Selbst wenn die Beispiele deutlich machen, dass insbesondere Apps sozialer Netzwerke dabei gern in den Vordergrund geraten – das zeigt lediglich, dass es hier besonders auffällt. Es bedeutet jedoch nicht, dass sich das Phänomen darauf beschränkt.

Auch diese Behauptung möchte ich nicht einfach so im Raum stehen lassen. So hat AndroidPIT-Mitglied Jörg Voß die Nachrichten-App N24 genauer unter die Lupe genommen. Hatten deren Entwickler doch versprochen, nicht auf sensible Daten wie insbesondere die IMEI zuzugreifen – da diese den Benutzer eindeutig identifizierbar machen würde. Gewissermaßen wurde auch Wort gehalten – doch stattdessen verwendet man nun die Geräte-ID, die ebenso eindeutig ist. Und überträgt sie natürlich auch noch unverschlüsselt über das Netz. Als ob dies nicht ausreichen würde, ist Ziel der Übertragung auch noch ein Drittanbieter. Noch nicht schlimm genug? Es geht schlimmer: Bei diesem Drittanbieter handelt es sich um ein Werbe-Netzwerk (der Ziel-Server: `mobile.smartadserver.com` in Frankreich).

Aber wie kann man sich vor derartigen Problemen schützen? GMV allein hilft hier offensichtlich wenig: Man sieht so etwas einer App ja wohl kaum „von Außen“ an. Und in den App-Beschreibungen wird sich kaum ein Satz finden wie „Wir übertragen ihre persönlichen Informationen zu Werbezwecken unverschlüsselt an Dritte“. Man könnte jetzt ständig sämtliche Blogs und News-Feeds verfolgen (mal ehrlich, wer macht so etwas – außer mir natürlich?), doch derartige Sicherheitslücken sind ja nicht bereits bei Erscheinen einer App bekannt.

Ein wichtiger Punkt dürfte definitiv deutlich geworden sein: Im Umgang mit offenen (und öffentlichen) Netzen besonders vorsichtig zu sein; erstere am Besten weitgehend meiden. Das schränkt das Risiko schon ein ganzes Stück ein.

Aber auch Andere haben dieses Problem erkannt, und bieten ihre Unterstützung an. So ist die Website des Services AppWatchDog der Forensik-

Firma viaForensics definitiv einen Besuch wert. Hier nahm man bekannte Apps genauestens unter die Lupe – und es war nicht gerade leicht für eine App, die strengen Tests mit einem grünen Haken zu bestehen. Es ist eher erschreckend, wie viele Apps hier durchgefallen sind: Schränkt man die Anzeige auf „Android“ und „Productivity“ ein, findet sich etwa kein einziger grüner Haken! Zwar sieht es für das iPhone nicht wesentlich besser aus (peinlicherweise ist der einzige grüne Haken hier bei „Google Mail“ gesetzt – da fragt man sich ernsthaft, warum die App das bei Android nicht geschafft hat?) – das zeigt jedoch lediglich, dass es im Bereich mobiler Apps um die Sicherheit nicht sonderlich gut bestellt ist. Ein Blick auf die Details entschärft den Schreck häufig ein wenig, wie beispielsweise der Eintrag zu K-9 Mail zeigt: Keine unverschlüsselt gespeicherten Passwörter – jedoch werden Benutzernamen und Daten unverschlüsselt abgelegt.

Bei *AppWatchDog* wurden die Apps in Intervallen erneut geprüft – man sah also, ob die Entwickler die Warnung ernst genommen haben. Erfreulich zu sehen, dass sich da bei einigen Apps tatsächlich etwas tat! Leider ist die Studie bereits seit geraumer Zeit beendet, sodass man die Weiterentwicklung nicht beobachten kann. Für Interessierte steht jedoch der Studien-Report gratis zum Download zur Verfügung.

Weitere Sicherheits-Probleme

Als ob obiges noch nicht genügt, bietet die Sicherheit noch einige andere Tücken – auf die ich hier nicht in voller Breite eingehen möchte. Ein paar Beispiele seien jedoch noch genannt:

Um die Authentizität von Apps zu gewährleisten, muss jede im Playstore eingestellte App mit einem Zertifikat signiert werden, welches den Entwickler eindeutig identifiziert. Dies erschwert es u. a. Betrügern, eine schädliche App als „Update“ einer beliebten anderen App auszugeben. Was aber, wenn ein solches Zertifikat einmal kompromittiert wurde – also „in fremde Hände“ geriet? Das mag zwar dem einen oder anderen Entwickler „peinlich“ sein – ein guter Zug ist es jedoch, die Anwender darauf unverzüglich hinzuweisen (etwa in der App-Beschreibung). Auf jeden Fall ist dann ein neues Zertifikat fällig.

Die Sache hat jedoch einen kleinen Haken: Neues Zertifikat heißt auch, dass Updates auf die mit dem alten Zertifikat signierten Versionen nicht mehr möglich sind (sonst wäre das System ja ad absurdum geführt). Besonders ärgerlich für die Anwender, die bei einem Update ihre Daten nicht mehr übernehmen können! Hier könnten Entwickler entsprechend Vorsorge treffen, indem sie – auch in den gratis-Versionen ihrer Apps – einen Datenexport ermöglichen. Gibt es zu der App auch eine Kaufversion, könnten auf diese Weise auch dort die Daten dann übernommen werden. Ebenso wäre dies bei einer neuen Version mit neuem Zertifikat möglich.

Wer bis hierher mitgelesen hat könnte nun denken: „Aha, wenn ich also auf

Apps von Drittanbietern verzichte …“ Leider nicht ganz. Zum Einen haben obige Beispiele ja gezeigt, dass durchaus auch vorinstallierte Apps betroffen sein können – und zum Anderen macht Android so ganz ohne Apps ja auch nicht wirklich Spaß. Wer derart paranoid ist, sollte dann besser zu einem der „guten alten Knochen“ greifen, die lediglich Telefonie und SMS unterstützen.

Und dummerweise gibt es auch ohne Apps hin und wieder Sicherheitsprobleme – etwa mit proprietärer Software vom originalen Hersteller. So berichtete etwa Android-TV von einer Sicherheitslücke in der Software „Kies“, die Samsung für seine Geräte einsetzt (für die Daten-Synchronisation sowie auch für Firmware-Updates). Hier zeigte sich allerdings das vorbildliche Verhalten des Herstellers: Kaum war die Lücke bekannt, schon war sie wieder geschlossen. Und Hand aufs Herz: Vor Fehlern gefeit ist niemand – so etwas kann also wirklich jedem Entwickler passieren. Gut, wenn dieser dann so schnell reagiert, wie in diesem Falle Samsung!

Wen das Thema noch näher interessiert, der findet im Netz natürlich weitere detaillierte Informationen. Etwa in einem sehr ausführlichen Artikel der Computerwoche zum Thema Android Sicherheitskonzept – oder in einem Block bei AndroidPIT mit dem Titel Android muss sicher werden. Eine weitere empfehlenswerte Quelle ist das Buch *Android Forensics and mobile Security* von Andrew Hoog, welches Ende Oktober 2012 auch in einer deutschen Übersetzung im Franzis-Verlag erschien.

System

Im zweiten Teil dieses Buches soll es darum gehen, einen Überblick über das Android-System zu erhalten – und es in den Griff zu bekommen. Die folgenden Kapitel sollen daher Antworten auf die Frage „Wo finde ich was?“ liefern – und ebenso aufzeigen, wie sich Dinge an den eigenen Bedarf anpassen lassen.

Der Super-User „root“

Von ihm ist in diesem Buch häufiger einmal die Rede: Dem „Wurzel-Account“ (engl. „root“ = „Wurzel“), ohne den das Eine oder Andere wieder einmal nicht geht. Und da es sich dabei um den zentralen System-Account handelt, soll er in diesem Kapitel auch gleich als Erstes behandelt werden.

Für die Hersteller/Provider scheint es sich hier um „die Wurzel allen Übels“ zu handeln: Dieser Account hat so viel Power, dass man den Anwender da besser gar nicht erst heran lässt. Zu groß die Gefahren, dass er sich kurze Zeit später mit „Garantieansprüchen“ meldet, weil er sich damit „etwas kaputt gemacht“ hat. Die einzige mir bekannte Ausnahme betrifft den US-Provider *Cincinatti Bell*, der seine Kunden nicht länger auf ein Motorola-Update warten lassen wollte – und daher schlicht eines selbst erstellte. Damit die Kunden dies auch installieren konnten, wurde die Anleitung zum Rooten gleich mitgeliefert – bei voller Garantie durch den Provider selbst. Eine rühmliche Ausnahme. Die aber noch immer nicht erklärt, um was es sich bei „root“ eigentlich handelt.

Kauft man einen Windows-PC, gibt es auf diesem einen Account für den Benutzer „Administrator“ – dem man bei der Ersteinrichtung ein Passwort verpasst. Installiert man Linux, heißt das Pendant „root“ (bei einem Mac sicher ähnlich). Android basiert auf Linux – aber trotzdem gönnen uns die Hersteller den root-Zugang in der Regel nicht, sondern drohen: „Wer sich root-Zugang zu seinem Gerät verschafft, verwirkt damit den Garantieanspruch.“

Bei dem Wort „root“ geht es also um den administrativen Zugang zum System, mit dem man alles (kaputt) machen kann. Naja, fast alles – die Hardware wohl eher nicht. Weshalb die Warnung mit der Garantie wohl letztendlich vor Gericht kaum haltbar sein dürfte, wenn man z. B. das Display wechseln lassen muss, oder der interne Speicher den Geist aufgibt: In diesen Fällen stellt eine EU-Richtlinie sicher, dass der Gewährleistungs-Anspruch trotz root nicht erlischt (anders sieht es aus, wenn die CPU verglüht, weil man sie hoffnungslos übertaktet hat – siehe CPU übertakten).

Braucht man den root-Zugang denn nun wirklich? Ja und nein. Wer mit seinem Gerät, dessen Funktionen, sowie der verwendeten Software bereits rundum zufrieden bist, alles so läuft, wie gewünscht, und „eigentlich“ nichts vermisst – der braucht auch keinen root-Zugang. Er hat ja bereits alles, was er braucht. Hat man hingegen ein Problem, was sich ohne den root-Zugang nicht lösen lässt, sieht das schon anders aus: Je nachdem, wie schwer es einen trifft, neigt sich das Zünglein an der Waage immer mehr der Anzeige zu, die mit „mach mich root!“ beschriftet ist.

Um die Entscheidung zu erleichtern, schauen wir uns zuerst einmal die Vor- und Nachteile an, die mit dem Rooten verbunden sind:

Vorteile des root-Zugangs

Eine ganze Reihe von Apps setzen ein gerootetes Gerät voraus. Oder stellen auf einem solchen zusätzliche Funktionen bereit. Zahlreiche Einstellungen und Änderungen lassen sich ohne root-Zugang gar nicht vornehmen:

- Übertakten bzw. Untertakten der CPU – um entweder mehr Leistung herauszukitzeln, oder den Akku zu längerem Durchhalten zu animieren
- Entfernen/Einfrieren vorinstallierter Apps – da kommen oftmals Dinge mit, die kein Mensch braucht. Ab Android 4.0 (aka *Ice Cream Sandwich*) lassen sich unerwünschte Kandidaten auch ohne root zumindest „einfrieren“, also deaktivieren.
- Bearbeiten der Start-Events und somit verhindern, dass unerwünschte Apps automatisch gestartet werden. Zwar gibt es auch Apps, die versprechen, das ohne root zu erledigen – doch diese schießen die betreffenden Apps lediglich nach dem Start wieder aus dem Speicher. Worauf sich die jeweiligen Apps oft einfach wieder starten. Mit root kann man den Start selbst verhindern.
- Neu-Kalibrieren des Akkus – eigentlich eher dem Android-System die Akku-Daten neu beibringen, indem veraltete Akku-Statistiken gelöscht werden. Als Folge davon ist die Anzeige der verbleibenden Kapazität/Laufzeit wieder genauer. (Ein Mythos – aber immer wieder gern zelebriert)
- Optimierung der Speicherverwaltung – personalisierte Einstellungen statt 08-15, damit der Droide wieder flüssiger läuft
- Anlegen einer Swap-Datei und somit Nutzung von „virtuellem RAM“
- App2SD auch mit Android < 2.2, bzw. die Nutzung erweiterter Varianten wie App2SD+/Link2SD
- Aufspielen alternativer Firmware (AKA „Custom ROM“)
- Ändern der Systemschriftart(en)
- Erstellen eines wirklich vollständigen Backups des Android-Systems (siehe auch Nandroid Backup)
- Einrichten einer Firewall
- Diverse weitere systemnahe Änderungen (aka „Modding“)

Diese Liste ist keinesfalls vollständig (natürlich auch nicht nach Relevanz sortiert – die wäre ohnehin wieder sehr subjektiv). Mit root hat man quasi überall Zugang – keine Ecke des Android-Systems bleibt verschlossen. Genau da liegt auch das Risiko – aber da liegt es auch beim root-Zugang auf dem Linux PC, oder dem Administrator-Zugang beim Windows-PC:

Risiken des root-Zugangs

Die Risiken sind schnell mit einem Satz beschrieben: Falsch angewendet, kann man sich mit root-Zugang das System unbrauchbar machen. Im schlimmsten Fall

verwandelt man gar seinen Androiden in einen Ziegelstein – wenn man z. B. ohne Sinn und Verstand die CPU hoffnungslos übertaktet, und diese schließlich den Hitzetod stirbt. Mit Wissen und Verstand eingesetzt, ist der root-Zugang ein mächtiges und nützliches Werkzeug. Quasi wie ein Autoschlüssel: Setzt sich der 8-jährige Steppke damit hinters Steuer … Was den Eindruck erweckt, dass man uns für absolut unmündig hält …

… oder sich schlicht vor unnötigen Rückgaben und Garantie-Einforderungen von sich selbst überschätzenden Anwendern schützen möchte. Ein nachvollziehbarer Grund – denn solche Anwender gibt es leider zu viele. Da es somit keinen root-Zugang ab Werk gibt, liegt für Anwender mit Sinn und Verstand das größere Risko eher in der Erlangung eines solchen. Je nach Gerät und Verfahren ist dieses größer oder fast gar nicht vorhanden. Da dies jedoch vom verwendeten Verfahren abhängt, lässt sich hier keine allgemeingültige Aussage treffen. Für weitere Details bietet u. a. ein Artikel bei StackExchange einen guten Einstieg.

Wie bekomme ich root-Zugang?

Das jetzt so zu erklären, dass es für jeden gilt, führt ein wenig zu weit. Für diese Übersicht kurz zusammengefasst, gibt es da mehrere Möglichkeiten – und je nachdem, um welches Gerät es geht, greift davon eine, keine, oder mehrere.

Da ist zum einen „Software-root“: Man lädt sich die passende App auf den Androiden, startet sie, und bestätigt: „Ja, ich will root!“. Fertig. Toll: Mit so einem Gerät fühle ich mich absolut sicher. Wer sagt mir, dass eine andere App das nicht im Hintergrund tut, ohne mich zu fragen? Die Nichtverfügbarkeit eines solchen Software-root-Tools ist allerdings auch kein Beleg für das Fehlen der entsprechenden ausgenutzten Lücke.

OK, auch die zweite Variante ist im Prinzip eine Art „Software-root“ (schließlich geht es ja um Software-seitigen Zugang). Nur geht es hier nicht um eine „einfache App“, sondern es ist schwieriger: Zunächst muss das USB-Debugging im Gerät aktiviert werden (expliziter Schritt, schwer von einer App auszuführen). Dann ist der Androide per USB-Kabel mit dem PC zu verbinden (unmöglich, dass das eine App im Hintergrund macht). Und schließlich muss man auf dem PC die „root-Software“ starten, die über das Kabel auf das Android-Gerät zugreift. Die Schritte sind noch immer einfach und nachvollziehbar – aber hier habe ich keine Bedenken, dass das ohne mein Zutun passieren könnte. Leider sind die meisten „Allrounder“ aus dieser Kategorie (wie etwa das bekannte SuperOneClick) auf ein auf dem PC laufendes Windows angewiesen – womit Mac OS- und Linux-Anwender im Regen stehen bleiben.

Eine dritte Variante nutzt den Update-Mechanismus des Recovery-Modus, um die root-Funktionalität nachzurüsten: Das passende `update.zip` wird auf die SD-Karte kopiert, und über den Menüpunkt „Apply update.zip from SD“ aus dem

Recovery-Modus eingespielt. Was allerdings nicht bei jedem Gerät so einfach möglich ist, da sich das vorinstallierte Recovery i. d. R. gegen „fremde Pakete“ wehrt (die „eigenen“ sind dann mit einer Signatur geschützt).

Welche Variante jetzt für ein bestimmtes Gerät verfügbar ist, und welche Software dafür benötigt wird, recherchiert man am besten in einem Forum. Die meisten Foren bieten gerätespezifische Bereiche, in denen es oft auch eine Sektion für root-Fragen gibt. Außerdem gibt es auch noch das rooting tag-wiki im Android-Bereich bei *Stack Exchange*, u. a. mit Verweis auf den rooting index. In letzterem sind für zahlreiche Geräte entsprechende Anleitungen verlinkt.

Laufen dann alle Apps mit root-Rechten?

Eine oft aufkommende Befürchtung – zum Glück unbegründet. Also die kurze Antwort: Nein, nicht ohne ausdrücklichen Wunsch des Anwenders.

Für eine detaillierte Antwort muss ich etwas tiefer greifen. Und wir müssen uns in Erinnerung rufen: Ein Android-System läuft ja mit Linux, also gelten hier auch entsprechende Richtlinien. Und jede App läuft darüber hinaus unter einem eigenen Benutzer. Auch root ist ein Benutzer, wenn auch ein ganz spezieller. Wenn eine „normale App“ etwas mit root-Rechten ausführen möchte, muss sie „root“ dazu auffordern. Der Befehl dazu heißt sudo, was wir in unserem speziellen Kontext mit „SuperUser, DO …“ wiedergeben können.

Bei Android laufen, wie bereits ausgeführt, Apps i. d. R. nicht als „root“, sondern unter ihrem jeweiligen eigenen User (Ausnahmen bilden einige wenige System-Apps). Also müssen sie für Aktionen, die root-Rechte benötigen, root halt höflich bitten – und das tun sie, indem sie dem auszuführenden Befehl ein „su“ (was dem Linux „sudo“ entspricht) voranstellen. Also „su <Befehl>“. Derart geweckt, schaut der SuperUser in seiner „Datenbank“ nach, ob die aufrufende App denn so etwas darf. Beim ersten Aufruf steht sie da noch nicht drin: Die Folge ist ein Popup der SuperUser-App „App xyz möchte etwas mit SuperUser-Rechten machen. Darf sie das?“. Dazu zwei Buttons für „Ja“ und „Nein“, sowie eine „Checkbox“, ob sich SuperUser diese Entscheidung für die Zukunft merken soll.

Bei jedem weiteren Aufruf findet der SuperUser die App in seiner Datenbank, sofern die Checkbox markiert und mit „Ja“ bestätigt wurde, mit dem Vermerk „die darf das immer“, und führt den Befehl direkt aus. Zur Sicherheit wird dieser Fakt jetzt nochmals als Hinweis eingeblendet (siehe Screenshot). Die App wird dabei nicht gebremst, es ist auch keine Interaktion nötig. Daher sollte das in diesem Falle dann sogar vom Lockscreen aus funktionieren. Etwas störend ist das natürlich im Falle einer Screenshot-App, wie das Bild zeigt – da dieser Hinweis

dann auf jedem Bild verewigt ist. Deshalb lässt er sich auch in den Einstellungen der SuperUser-App abschalten.

Gibt es irgendwo weitere Info zu diesem Thema?

Wer noch unsicher ist, findet im Netz natürlich noch weitere Informationsquellen – überwiegend in Foren und Wikis. Eine kleine Auswahl sei hier kurz aufgeführt:

- AndroidPIT Blog: Rooten leicht gemacht (11/2011; Update 7/2015)
- Brutzelstube: Rooten – und Garantie? (5/2011)
  In der Brutzelstube gibt es noch zahlreiche weitere Informationen, insbesondere – aber nicht ausschließlich – zu Themen rund um root, z. B. auch diverse root-Guides
- Wikipedia (EN): Rooting (Android OS)
- Wikipedia (DE): CyanogenMod – einer der besten Gründe pro-root
- DroidWiki (DE): Root (und weitere interessante Themen rund um Android)
- Android.SE (EN): rooting tag-wiki (mit nützlichen Informationen und Links)
- App: ROM Toolbox – Werkzeugkasten für Wurzelmenschen (siehe Screenshot), mit ROM-Manager, App-Manager, CPU-Manager, Font-Installer, Root-Browser (FileManager) …

ROMs

Wie im Glossar zu ROM nachgelesen werden kann, findet sich an dieser Stelle das Android-System (sofern man vom Gerät selbst spricht) bzw. das entsprechende Installations-Archiv (wenn es um den Download eines ROMs geht). In diesem Kapitel soll das ROM ein wenig genauer betrachtet werden.

Irgendwo basiert eigentlich jedes Android-System auf dem AOSP, dem *Android Open Source Project* – den dieses stellt schließlich das Kernsystem zur Verfügung. Nun ist es leider nicht einfach damit getan, sich hier den Code herunterzuladen, zu kompilieren, und auf dem Gerät zu installieren – zu unterschiedlich ist die Hardware verschiedener Geräte, sodass fast immer Hersteller- und Geräte-spezifische Anpassungen nötig sind (außer natürlich für die direkt unterstützten Geräte). Und das ist der erste Punkt, in dem sich sogenannte „Stock ROMs“ von den „originalen AOSP ROMs“ unterscheiden.

Nebenbei erklärt dies auch viele der Verzögerungen bei Updates: Wenn das AOSP-Team eine neue Android-Version fertiggestellt hat, müssen die Hersteller ja noch ihre spezifischen Hardware-Anpassungen (Treiber etc.) vornehmen. Das ist jedoch nur einer der Gründe – denn schließlich haben sie ja (Dank Open Source) bereits während der Entwicklung Zugriff auf den AOSP-Code, und könnten so auch etwas zeitiger mit ihren Anpassungen beginnen. Da jedoch die meisten Hersteller mehr als nur ein Gerät im Rennen haben, müssen diese Anpassungen auch für mehrere Geräte durchgeführt werden.

Aber es gilt ja nicht nur, die Hardware-spezifischen Dinge anzupassen: Da wären ja auch noch die Hersteller-spezifischen Benutzeroberflächen. HTC Sense, Motorola MotoBlur, Samsungs TouchWiz, und wie sie alle heißen – auch diese müssen mit den „neuen Gegebenheiten“ klarkommen. Bis hierher ist das noch einzusehen; doch dann wäre da noch die sogenannte Bloatware, die noch mit rein muss: Apps, die weder vom AOSP, noch von der spezifischen Benutzer-Oberfläche zwingend benötigt werden – von denen man aber meint, dass die Anwender (also wir) nicht ohne leben könnten. Facebook, Twitter, Peep, die ach-so-tolle Aktien-App (die insbesondere für Europäer wenig interessant ist, da sie nur NASDAQ kennt) – alles Dinge, die sich der Anwender (so er sie denn wirklich braucht) auch bequem über den Playstore installieren kann. Nicht nur, dass viele dieser Apps nerven und Ressourcen fressen – so wirklich loswerden kann man sie wieder nur mit root. Und Apps wie `su` oder `SuperUser.apk` sucht man bei der Bläh-Ware vergebens.

Eine weitere Ebene oben drauf bekommen dann noch diejenigen, die ihr Gerät „mit einem Rabatt“ bei ihrem Netzbetreiber erworben haben: Dieser möchte natürlich auch noch seinen Senf dazugeben. Oder eher seinen Brandy – nämlich das sogenannte *Branding*. Meist an Dingen wie einer speziellen Boot-Animation mit einem tollen pinkfarbenen „T“ zu erkennen. Und an weiterer Bloatware. Das alles muss natürlich bei einer neuen Android-Version auch wieder angepasst

werden – womit klar ist, warum man hier besonders lange warten darf: *AOSP*-ROM → Stock-ROM des Herstellers → Stock-ROM des Netzbetreibers. Genau genommen sollte man hier also eigentlich nicht nur vom „Stock-ROM“ sprechen, sondern drei verschiedene Typen unterscheiden:

- *AOSP*-ROM (auch als „plain Vanilla“ bezeichnet): Das originale ROM des *AOSP*. Leider ohne spezielle Anpassungen nicht auf allen Geräten lauffähig
- Stock-ROM: Das vom Hersteller an die Hardware angepasste (und mit einiger Bloatware angereicherte) *AOSP*-ROM – natürlich passend zum Gerät
- Vendor-ROM: Das vom jeweiligen Netzbetreiber angepasste (und mit weiterer Bloatware angereicherte Stock-ROM – auch passend zum Gerät, nur viiieel später …

Je nachdem, welches Gerät man wo erworben hat, ist i. d. R. eine der letzten beiden Varianten vorinstalliert. Mit einem wirklich reinen AOSP ROM wird meines Wissens kein Gerät ausgeliefert.

Aktualisierungen des ROMs erhält man normalerweise vom selben „Produzenten“. Oftmals kommen sie als OTA-Update („Over The Air“, also über das Netzwerk) mit einem größeren Download daher. Wird einem ein solches angeboten, sollte man vor der Bestätigung eine WLAN-Verbindung sicherstellen: Der Umfang eines solchen Downloads liegt zumeist locker im zweistelligen Megabyte-Bereich.

Alternativ werden Installations-Archive oftmals auch auf den Service-Seiten des jeweiligen Herstellers zum Download angeboten. Diese benötigen dann für die Installation meist eine spezielle Software, die es ebenfalls dort zum Download gibt – und die dann häufig nur unter Windows funktioniert. Die Anzahl an Herstellern, die eine einfache, vom PC unabhängige Installation über das Recovery-Menü des Android-Gerätes nutzen, ist fast nicht-existent.

Eine weitere Kategorie von ROMs habe ich an dieser Stelle unterschlagen, und werde erst im Modding-Teil auf sie eingehen: Custom ROMs ermöglichen eine Bloatware-freie Android-Erfahrung sowie eine weitreichende Anpassung des Systems an die eigenen Bedürfnisse, nahezu ohne jegliche Einschränkung. Und im Gegensatz zu den von Herstellern angebotenen ROMs ist hier die einfache, vom auf dem PC installierten Betriebssystem unabhängige Installation über das Recovery-Menü eher die Regel als die Ausnahme.

Dateisysteme und Datenstrukturen

Dieses Kapitel wird ausführlich in der „Verlags-Edition“ behandelt. Dennoch einige Details bereits an dieser Stelle. Zum Einen, damit der Leser weiß, was ihn dort erwartet – und zum Anderen, um zumindest ansatzweise die Grundlagen zum Verständnis einiger technischer Eigenheiten zu vermitteln.

Dateisysteme

Unter Android kommen verschiedene Dateisysteme zum Einsatz, die alle ihre Eigenheiten haben. Warum man sich nicht einfach auf ein Dateisystem festgelegt, und dieses dann durchgängig verwendet hat? Würde das nicht Vieles vereinfachen? Vielleicht. Aber von unseren Hosentaschen-Computern erwarten wir schließlich, dass sie sowohl performant laufen – als auch, dass der Akku möglichst lange durchhält. Zwei Dinge, die sich nicht gleichzeitig maximieren lassen – wie auch das Kapitel Tuning noch zeigen wird. Jedes einzelne Dateisystem hat seine Vor- und Nachteile: Was für das eine Einsatzgebiet gut geeignet ist, passt nicht unbedingt auch für alle anderen. Also wird mit Watson kombiniert.

Welche Dateisysteme finden sich nun auf dem Android-Gerät? Kommt ganz darauf an. Denn das entscheidet sich wieder einmal von Gerät zu Gerät, Android-Version zu Android-Version, Hersteller, ROM-Küche, und weiteren Eigenheiten. Doch jedes Android-Gerät verwendet mehrere (nicht jedoch unbedingt alle) der folgenden Dateisysteme:

EXT

Das Extended Filesystem wurde 1992 speziell für Linux entworfen, und seitdem ständig weiterentwickelt. Im Einsatz sind heute noch die Versionen 2-4 (ext2, ext3, ext4) – die allesamt auch unter Android anzutreffen sind. Mit Version 3 wurde das Journal eingeführt (was das System sicherer und die Überprüfung schneller macht). Ext4 legte dann in Sachen Performance noch einmal zu.

procfs

Bei procfs handelt es sich um ein virtuelles Dateisystem, das zur Ausgabe und Änderung von System- und Prozessinformationen dient (daher auch der Name: vom englischen *process*).

tmpfs

Der Name lässt es bereits vermuten: Bei tmpfs handelt es sich um ein temporäres Dateisystem. Zum Einsatz kommt es auf RAM-Disks, also im Arbeitsspeicher – für Daten, die einen Reboot nicht überleben müssen/sollen.

reiserfs

Das Reiser File System ist unter Android eher selten zu finden (obwohl es von einigen Custom-ROMs eingesetzt wird). Es war das erste Journaling-Dateisystem, welches vom Linux-Kernel direkt unterstützt wurde. Im Alltag trifft man überwiegend nur noch die neueren Versionen 3 und 4 an.

vfat

VFAT steht für **V**irtual **F**ile **A**llocation **T**able, und kann sowohl auf FAT16 als auch FAT32 aufsetzen. Unter Android wird es ziemlich ausschließlich für SD- und eMMC- Karten eingesetzt.

YAFFS2

YAFFS ist wieder einmal eine Abkürzung aus dem englisch-sprachigen Bereich. Wie so viele dieser mit *YA* beginnende Akronyme (Linuxer kennen das bereits), haben wir es auch hier mit „einfach etwas anderem“ zu tun: **Y**et **A**nother **F**lash **F**ile **S**ystem. Es wurde speziell für den Einsatz mit NAND-Flash entwickelt – eben jenem Speichertyp, der in fast allen mobilen Geräten (also auch in unseren Androiden) verbaut ist. Daher kennt *YAFFS* die entsprechenden Eigenheiten, und kann so für eine längere Lebensdauer sorgen. In allen halbwegs aktuellen Android-Geräten wird, wenn überhaupt, die zweite Version davon eingesetzt – also *YAFFS2*. Da die NAND-spezifischen Dinge jedoch heutzutage vom Controller-Chip übernommen werden, kommt YAFFS immer mehr aus der Mode.

Auf die speziellen Eigenheiten der einzelnen Dateisysteme, insbesondere in Bezug auf Android, geht – wie bereits kundgetan – die „Verlags-Edition“ dieses Buches näher ein.

Datenstrukturen

Während wir uns unter Dateisysteme dem Speicher eher von der Seite der dahinter stehenden Technologie und Fragen wie *Warum ist YAFFS2 so besonders gut für den Flash-Speicher geeignet?* genähert haben, geht es jetzt vielmehr um das „Wo“: Wo ist welches Medium eingebunden? Wo finde ich welche Daten? Auch dafür gilt wieder: Grobes zum Verständnis hier – ausführlicheres, vertiefendes, und hintergründigeres in der „Verlags-Edition“ des Buches.

Eingebundene Dateisysteme

Der „einfache Anwender“ weiß einfach: Da ist Speicher drin, und darauf liegen sicher die ganzen Dateien. Der Anwender mit ein wenig mehr technischem Hintergrund kann sich zumindest denken, dass der interne Speicher wohl auf einem anderen „Laufwerk“ liegt als die SD-Karte. Aber wer würde schon vermuten, dass es insgesamt 15 oder mehr so genannte „Mount Points“ gibt –

wobei sich hinter jedem ein eigenes Dateisystem verbirgt? Kein Witz, das ist tatsächlich der Fall! Einige davon seien hier kurz aufgeführt – wobei dies nur exemplarisch ist, und nicht auf jedem Gerät etc. übereinstimmen muss:

Mount-Point	Typ	Kommentar
`/cache`	ext4	Anwendungs-Cache
`/data`	ext4	Dalvik-/ART-Cache, Logs, Konfigurations-Dateien, Anwendungs-Daten …
`/dev`	tmpfs	„Geräte-Dateien“ (devices)
`/mnt/asec`	tmpfs	„entschlüsselte“ Version der mit *App2SD* auf die Karte verschobenen `.apk` Dateien
`/proc`	procfs	System- und Prozess-Informationen
`/storage/sdcard0`	vfat/fuse	interne „SD-Karte“ (eMMC)
`/storage/external_sdcard`	vfat	externe SD-Karte
`/system`	ext4	vorinstallierte System-Dateien

Die in obiger Tabelle bezeichneten Partitionen bzw. Verzeichnisse für die SD-Karten sind nicht bei allen Geräten identisch: Manche Geräte verfügen über keinen Steckplatz für die externe SD-Karte, andere weisen keine interne SD-Karte aus – und selbst die übrigen binden selbige an den verschiedensten Stellen im System ein. Auf allen Android-Geräten an genau diesen Stellen vorhanden sein sollten jedoch die standardisierten Mount-Points: `/cache`, `/data`, `/dev`, `/proc`, und `/system`.

Dass `/dev` und `/proc` eine besondere Rolle spielen, ist zumindest eingefleischten Linux-Anwendern bekannt: An ersterer Stelle kann man nachschauen, welche „Geräte“ denn so erkannt wurden (so findet sich für die meisten anderen genannten Dateisysteme unter `/dev/block` das entsprechende „Block-Gerät“, über das auf den physischen Speicher zugegriffen werden kann). Und unter Letzterem lassen sich so einige System-Informationen auslesen (unter `/proc/mtd` etwa finden sich Informationen zu den im NAND-Flash angelegten Partitionen, und unter `/proc/sys/kernel` kann man verschiedene Kernel-Parameter auslesen oder auch anpassen). Doch auch `/system` hat eine Besonderheit: Es wird im reinen Lese-Zugriff eingebunden – Änderungen sind hier also nicht (ohne weiteres bzw. ohne root-Rechte) möglich. In dieser Partition finden sich nämlich die System-Programme – aber auch all die tollen „Zwangs-Beglückungen“, die uns Geräte-Hersteller und ggf. Netzanbieter so gern „zur Verfügung stellen“.

Einen ganz speziellen Fall stellt auch das temporäre Dateisystem dar, welches unter `/mnt/asec` eingebunden ist. Hat man mit *App2SD* Apps auf die SD-Karte ausgelagert, werden diese verschlüsselt im Verzeichnis `.android_secure/` gespeichert. In diesem verschlüsselten Zustand kann das System sie leider nicht ausführen. Daher werden sie zuvor entschlüsselt unter `/mnt/asec` abgelegt – wo sie bei einem Systemstart automatisch wieder verschwinden.

Weitere Partitionen

Es gibt noch einige weitere Partitionen, von denen u. a. die folgenden noch interessant sein dürften:

Partition	Kommentar
`/boot`	Kernel & RAM-Disk
`/recovery`	Alternative Boot-Partition für Recovery Mode
`/sd-ext`	kein Standard; bei App2SD+ als „ausgelagerte `/data` Partition“ genutzt

Einige wichtige Verzeichnisse

Während etwa an allem, was sich unterhalb von `/system` befindet, nur wirklich „fachkundiges Personal“ sich vergreifen sollte – finden sich so einige interessante Verzeichnisse unterhalb von `/data`. Etliche davon werden für verschiedene Protokoll-Funktionen (auch als „Logging“ bekannt) genutzt; diese sind unter Andere Logdateien aufgeführt.

Unter `/data/data` befinden sich dann die von den Apps verwalteten Daten – wobei jeder App ein eigenes Verzeichnis zugewiesen ist, auf das (normalerweise) nur sie selbst Zugriff hat (dass manch eine App diesen Schutz für die eigenen Daten aushebelt, wurde ja bereits zum Thema Sicherheit unter App-Daten gezeigt):

```
/data/data/com.example.demoapp
├── cache                Verzeichnis
│   └── webviewCache     Verzeichnis
│       ├── 027e59a0     Cache-Datei
│       └── 057606c4     Cache-Datei
├── databases
│   └── example.db       SQLite-Datenbank
├── lib                  Verzeichnis
└── shared_prefs         Verzeichnis
    └── example.xml      Konfigurations-Datei
```

So in etwa könnte das für eine „Beispiel-App“ aussehen – es können aber durchaus weitere Verzeichnisse hier vorhanden sein, oder nicht einmal alle aufgeführten. Unsere Beispiel-App hat also …

- … ein Cache-Verzeichnis, und darin noch ein paar Dateien von Seiten, die aus dem Web geladen wurden
- … ein Datenbank-Verzeichnis mit einer SQLite-Datenbank (es können natürlich auch mehrere sein), in der sie strukturiert Daten in mehreren Tabellen verwalten kann
- … ein ominöses Bibliotheks-Verzeichnis ohne Inhalt
- … ein Verzeichnis mit Konfigurations-Dateien im XML-Format

Die Zugriffs-Berechtigungen sollten hier normalerweise etwa wie folgt aussehen: `-rw-r-----` für Dateien (der Eigentümer – also die App selbst – darf also lesen und schreiben, die Gruppe noch lesen, und der Rest hat hier nichts zu suchen), sowie maximal `drwxr-x--x` für Verzeichnisse (Eigentümer darf lesen, schreiben und ausführen – für Verzeichnisse bedeutet das: Hineinwechseln – die Gruppe noch lesen und ausführen, der Rest allenfalls noch ausführen); wobei „der Rest“ hier eigentlich gar nichts mehr verloren hat.

Einige ausgewählte weitere Verzeichnisse (Quelle: Mein Motorola Milestone 2, als es noch Stock-Firmware mit Android 2.2.2 hatte – aktualisiert/abgeglichen mit meinem LG P880, Android 4.4.4) möchte ich hier kurz aufführen – für den Rest sowie weitere Details hingegen wieder auf die “Verlags-Edition” verweisen:

```
/data
├── backup                                          Android Backup Queue
│   ├── com.android.internal.backup.LocalTransport
│   ├── com.google.android.backup.BackupTransportService  Google Cloud
Backup
│   └── pending
├── dalvik-cache                                    Dalvik-Cache
│                                                   (.dex (Dalvik
EXecutable) Dateien)
├── local                                           Zwischenspeicher für zu
│                                                   installierende .apk
Dateien u. a. m.
├── misc                                            Verschiedene
Konfigurations-Dateien
│   ├── adb                                         adb_keys (autorisierte
Geräte)
│   ├── bluetooth                                   Bluetooth-Details, u. a.
Paired Devices
│   ├── dhcp                                        DHCP leases etc.
│   ├── keychain                                    vom Anwender
hinzugefügte Zertifikate / Blacklisten
│   ├── keystore                                    dito
│   ├── systemkeys                                  Keys für “encrypted
apps”
│   ├── vpn                                         VPN Einstellungen
│   └── wifi                                        WLAN Konfigs und APNs
└── system                                          Konfigurations- und Log-
Dateien
    ├── dropbox                                     Drop-Box (Ablage) für
System-Log Details
    ├── netstats                                    Statistiken zum
Datenverbrauch
    ├── registered_services                         XML-Dateien mit
Informationen zu
    │                                               verfügbaren
(registrierten) Services
    ├── security                                    Sicherheits-Zertifikate
(binär)
    ├── shared_prefs                                Konfigurations-Daten
(u. a. zu
    │                                               gesammelten Log-Dateien)
    ├── sync                                        Einstellungen von
    │                                               “Kontakte &
```

```
Synchronisation”:
    │                                                Welche Dienste sollen…?
    └── usagestats                                   Nutzungs-Statistiken
```

Eine Überraschung war für mich, dass das Verzeichnis `/data/system/dropbox` gut mit Protokollen gefüllt war, obwohl ich nicht einmal ein Dropbox-Konto habe! Nicht verwirren lassen: Das Verzeichnis hat nichts mit dem Cloud-Service gleichen Namens zu tun, sondern ist eine Art Zwischenspeicher für protokollierte System-Ereignisse.

Im Verzeichnis `/data/system` finden sich überdies Dateien wie `accounts.db` (die Account-Datenbank mit konfigurierten Accounts, Authentifizierungs-Tokens, und mehr), `appwidgets.xml` (Launcher und konfigurierte Widgets), `batterystats.bin` (Akku-Statistiken), `packages.list` / `packages.xml` (installierte Apps / App Permissions), und weitere.

Externer Speicher

Auf der SD-Karte sieht es auch immer recht wild aus – und so manch einer mag sich nicht nur einmal gefragt haben, ob hinter dem ganzen Wildwuchs eine definierte Verzeichnis-Struktur verborgen sein könnte. Die Antwort lautet: Teilweise. Einige Standards sind etwa in der Entwickler-Dokumentation festgelegt, andere lassen sich durch Vergleichen verschiedener Geräte „erraten“.

Verzeichnis	Kommentar
`Android/data/<package_name>/`	entspricht `/data/data/<package_name>`. Aktuelle Android-Versionen löschen dieses Verzeichnis bei Deinstallation der zugehörigen App.
`Music/`	Was der Medien-Scanner hier findet, stuft er als Musik ein
`Podcasts/`	… als Podcast ein
`Ringtones/`	… als Klingelton
`Alarms/`	… Alarmton
`Notifications/`	… Benachrichtigungston
`Pictures/`	Bilder (außer die der integrierten Kamera), z. B. mit dem Gerät erstellte Screenshots
`Movies/`	Filme (außer die des integrierten Camcorders)
`Download/`	verschiedene Downloads
`dcim/`	Fotos/Videos der intergrierten Kamera

Nicht immer wird sich an diese Strukturen gehalten – doch geben sie zumindest eine grobe Idee. Dummerweise machen die hier benannten Verzeichnisse meist weit weniger als die Hälfte der vorhandenen aus – was sich ab Android 4.4 (aka „Kitkat“) Dank des tollen Features „Schreibschutz für alle Apps“ ein wenig geändert haben mag.

System-Info

Der Google Play Store bietet zahlreiche Tools, welche die verschiedensten Information zum System zu Tage fördern. Manche dieser Informationen möchte man gern ständig im Blick haben, andere benötigt man seltener. Zur ersten Kategorie gehören vielleicht Dinge wie CPU-Auslastung oder verfügbarer Speicher: Da bieten sich zweifellos Widgets an. Diese eignen sich allerdings weniger für ausführlichere Informationen (wie etwa eine Liste laufender Prozesse).

Es ist nicht einfach, alles eindeutig zu gruppieren – dazu gibt es zu viele Überschneidungen. Ich hoffe dennoch, dass mir eine passende Zusammenstellung gelungen ist.

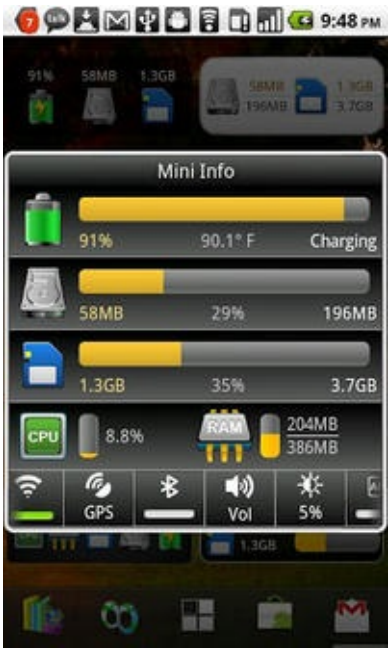
Systeminfo-Widgets

Und schon beim ersten Tool scheitert es, da es nahezu alle Bereiche umfasst. Aber es ist sauber in einzelne Pakete aufgegliedert. Die Rede ist von *Elixir 2*, dessen Basis-Paket im Kapitel Ausführliche System-Infos vorgestellt werden soll. Sehr vorbildlich hat der Entwickler den Zugriff auf persönliche Informationen in ein Personal Add-On ausgegliedert; so sind die geforderten Permissions der Elixir 2 Widgets zwar entsprechend dem, was sie im Auge behalten sollen, nicht wenige – dafür aber dennoch recht unbedenklich.

Der in diesem Kapitel interessante Teil hat im Screenshot einen blauen Hintergrund, und zeigt wichtige Informationen zum System auf einen Blick. Dabei hat man die Auswahl aus einigen Datenquellen: Akku (Füllstand, Spannung, Temperatur), externer/interner Speicher, CPU (Auslastung und/oder Frequenz), RAM, Anzahl laufender Prozesse, Uptime, Signalstärke (Wifi, Mobilfunk), Traffic (Wifi, Mobilfunk) – aber auch (roter Bereich) Anzahl ungelesener Mails/Kurznachrichten, verpasste Anrufe und mehr. Und um dem nächsten Kapitel gleich vorzugreifen: Auch eine ganze Reihe von „Toggle-Widgets“ (also „Schnellumschalter“, siehe obere Bildhälfte) sind mit an Bord, wie etwa APN, Wifi, Wifi Hotspot (aka Tethering), Bluetooth, GPS, Flugmodus, und sogar SD-Mount sowie SD-Refresh (Media-Scan anstoßen) stehen hier – unter anderem – zur Auswahl. Da bleibt fast kein Auge trocken.

Ein unheimlich erfolgreiches Päckchen, dieses *Elixir 2*. Das best bewerteste Päckchen in diesem Bereich, nebenbei bemerkt. Und ich trau mich kaum zu schreiben, was denn wohl das zweitbest bewertete Päckchen sein könnte – das trägt nämlich den Namen Elixir, und ist die Vorgänger-Version (bei der die

Widgets noch nicht in einer separaten APK-Datei ausgeliefert wurden).



Auch das an dritter Stelle folgende Mini Info fällt in diese Misch-Kategorie: Für den Home-Screen bietet es einige Widgets mit den wichtigsten Informationen, wie Akku (Reststand, verbraucht, Spannung, Temperatur, Status), RAM/CPU (in Nutzung bzw. verfügbar) und Speicherplatz (intern/extern; belegt, frei, total). Die dahinterstehende App stellt dann bei Bedarf weitere Informationen zur Verfügung – sowie eine Bar mit „Toggle-Widgets“. Die Info-Widgets lassen sich, wie auf dem Screenshot zu sehen, nach Gusto gruppieren. Ebenso ist das Festlegen eines Hintergrundes möglich: Von völlig transparent, über halbtransparent/Glas bis hin zur „Vollfarbe“.

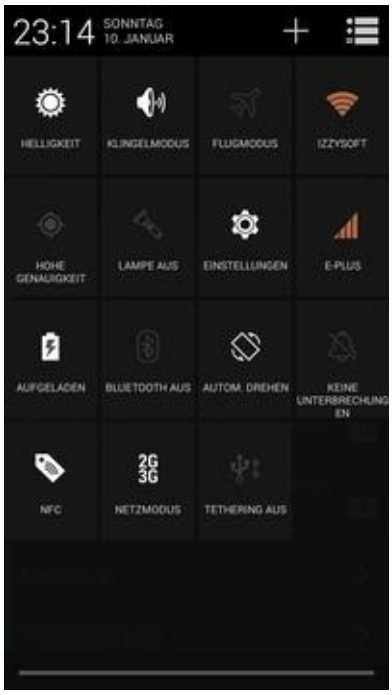
Zum Ausprobieren gibt es *Mini Info* gratis, der volle Funktionsumfang lässt sich per Donation (ca. anderthalb Euro) freischalten. Übrigens gibt es für diese App einen super Support mit kurzen Reaktionszeiten. Leider wurde die App jedoch seit 2012 nicht mehr aktualisiert.

Auch wenn sich diese Apps wieder nicht ausschließlich dem Thema „Systeminfo-Widgets“ widmen, finde ich die Kombination sehr sinnvoll: Uhr und Wetter (im Stil von *HTC Sense*, daher auch der Name), und unmittelbar darunter die wichtigsten System-Infos eingeblendet – so kommt Sense Analog Clock Widget 4x2 (auch in anderen Varianten mit 24-Stunden-Display, oder im Glass-Look erhätlich – einfach einmal die weiteren Apps dieses Entwicklers durchforsten) daher. Ja, auch die Wettervorschau für die nächsten Tage ist integriert – doch da es hier ja um die Systeminfo-Widgets gehen soll: Das Wichtigste wird dargestellt (RAM, interner/externer Speicher, Temperatur, und Akku-Stand), detailliertere Infos gibt es beim Antippen. Auch lässt sich konfigurieren, welche App beim Antippen der Stunden- oder der Minutenzahl geöffnet werden soll (Kalender und Wecker bieten sich hier an), sodass kein Platz „verschwendet“ wird. Obwohl die Screenshots etwas anderes suggerieren, scheint diese App noch aktuell gepflegt.

Eine Auswahl weiterer Widget-Sammlungen finden sich in dieser Übersicht bei *IzzyOnDroid*.

Schnellumschalter

Hier könnte ich es mir jetzt einfach machen und schreiben: „Siehe oben“ – zumindest die vorderen Plätze sind sogar identisch (*Elixir 2* und *Elixir*). Ebenso ist oben genanntes *Mini Info* recht weit oben mit dabei. Aber einige Kandidaten haben sich speziell auf das „schnelle Umschalten“ spezialisiert, und sollen daher auch in diesem Kapitel genannt werden.



Doch muss man nicht unbedingt eine App installieren, um die wichtigsten Schalterchen schnell zur Hand zu haben – aktuelle Androiden haben so etwas von Haus aus an Bord. Zu finden sind die so genannten *Quick Settings* (oder zu gut Deutsch: Schnelleinstellungen) im Benachrichtigungs-Bereich. Man öffnet sie, indem man die Nachrichtenleiste an ihrem rechten Rand anfasst und herunterzieht – bzw. wenn sie bereits geöffnet ist, über das entsprechende Icon am oberen rechten Rand. Was hier erscheinen soll, lässt sich konfigurieren, indem man auf das „+“ Symbol tippt. Je nach Android-Version und Implementierung führt entweder einfaches Antippen zum Umschalten und langes Antippen in die zugehörigen Systemeinstellungen, oder umgekehrt. Der große Vorteil: Hier sind die Schalterchen jederzeit erreichbar, ohne dass man dafür erst auf den Homescreen zurückkehren muss.

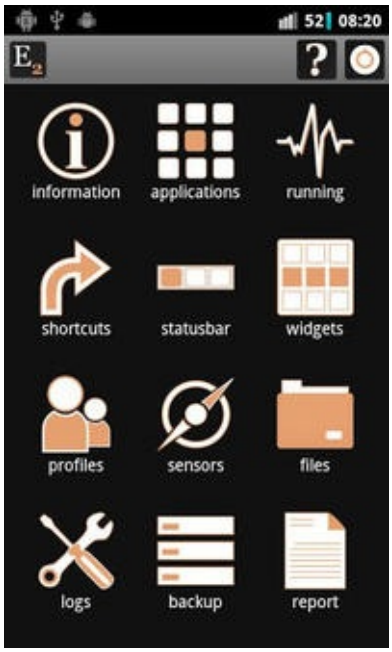
Eine feine Sache sind auch die Power Toggles. Hier lassen sich beliebige Kombinationen aus „Toggles“ in einem Widget kombinieren – auch sieben Schalter in einer Leiste (4x1 Widget) sind kein Problem in der Darstellung (anders aussehen mag das in Bezug auf die Treffsicherheit von Wurstfingern). Was auf dem Screenshot auch sehr schön zu sehen ist: Man kann durchaus mehrere Widgets anfertigen, die unabhängig voneinander konfigurierbar sind. Das ergibt größtmögliche Flexibilität – insbesondere, da sich auch Tasker hier mit einbeziehen lässt.

Darüber hinaus geht hier mehr als bloßes „An“ und „Aus“: Auf Wunsch gibt es auch „Pop-Up-Schieberegler“ etwa für stufenlose Regelung der Bildschirm-Helligkeit oder Klingelton-Lautstärke. Wobei für den letzteren auch ein Multi-Schalter möglich ist: An/Aus/Vibrieren, plus Knopf für Benutzerwert. Wer mag, kann auch einfach den Mobilfunk abschalten – etwa um ungestört von Anrufen und eingehenden Kurznachrichten im WLAN zu surfen. Will man sich mit den Toggles nicht den Homescreen zupflastern, lassen sie sich sogar in den Benachrichtigungs-Bereich integrieren.

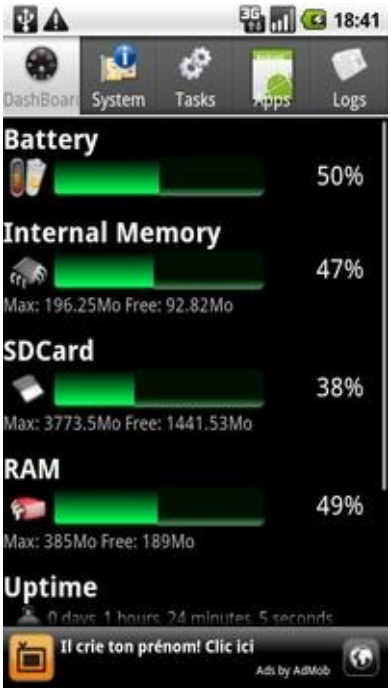
Damit könnte nun die Vorstellung der Schnellumschalter wirklich enden. Und

das tut sie auch – mit dem Hinweis, dass sich bei Bedarf weitere Apps in dieser Übersicht finden.

Ausführlichere System-Infos



Auch an dieser Stelle geht der Verweis zuerst „nach oben“: Wie schon bei den Widgets, steht hier ein *Elixir* an erster Stelle. Und zwar die Basis-App Elixir 2, die – wie im Bild zu sehen – Informationen zu vielen Bereichen zur Verfügung stellen möchte. Dazu zählen u. a. Geräte-Informationen wie Akku (Ladezustand, Spannung/Strom, Temperatur), interner und externer Speicher (insgesamt/frei, wo eingebunden), CPU (Auslastung, Frequenz), RAM (insgesamt/frei), Telefonie (Netzwerk-/Telefonmodus, Provider, APN); installierte Anwendungen (wo installiert, wie viel Platz belegt die App, wie viel ihre Daten, welche Version); welche Prozesse/Services laufen (und verwenden wie viel RAM/CPU etc.) – aber auch Daten der Sensoren. Ebenso lassen sich diverse Systemeinstellungen (wie etwa Bildschirmhelligkeit oder Lautstärke) anpassen, oder Aktionen wie Cache-Bereinigung, SD-Karte ein-/ausbinden oder Bluetooth-Scan durchführen. Als würde das noch nicht reichen, greift die App auch noch in das Kapitel Logcat, und zeigt die Systemlogs an. Großer Rundumschlag – oder Schweizer Offiziersmesser. Und dabei habe ich noch lange nicht alles aufgezählt.



Recht umfangreiche Informationen zum System stellt auch Android System Info bereit. Diese bestehen u. a. aus einem „Dashboard“ mit den wichtigsten Eckdaten, einem Task-Manager, der sowohl Task-Switcher als auch Task-Killer beinhaltet, einer Anwendungs-Verwaltung für die installierten Apps, einem farblich aufbereiteten System-Log (noch jemand, der dem Kapitel Logcat vorgreift), sowie weiteren Informationen rund um das System. Etliche Nutzer berichten, was ihnen besonders gefällt: Zahlreiche Informationen, und doch übersichtlich aufbereitet. Entsprechend weit oben ist es auch auf der Bewertungsliste platziert: Durchschnittlich viereinhalb Sterne bei ca. 40.000 Bewertungen. Wieder täuscht der Screenshot ein wenig: Diese App wurde zumindest im November 2011 nochmals aktualisiert.

Vom Aufbau her vergleichbar wäre OS Monitor – bringt allerdings noch ein paar Schmankerln mit. Wie etwa die auf dem Screenshot zu sehende „grafische

WhoIs-Abfrage“: Wohin hat sich diese App gerade verbunden? Aha, da sitzten die …

Im Hintergrund erkennt man bei genauerem Hinsehen noch die „Reiterlein“. Hinter selbigen verbergen sich eine Prozessliste (die sich nach verschiedenen Kriterien wie beispielsweise CPU oder Speicherverbrauch sortieren lässt, und auch einen Task-Killer beinhaltet), eine Übersicht über verfügbare Netzwerk-Interfaces (mit, sofern verfügbar, Details zur MAC-Adresse, zugeteilter IP, sowie bereits übertragener Datenmenge), aktuelle Datenverbindungen (welche App wohin, mit welchem Netzwerk-Protokoll?), eine Zusammenfassung „verschiedener“ Informationen (CPU, Akku, Speicher) sowie – ja, auch hier wieder – ein farblich aufbereitetes Systemlog.

Zusätzlich lässt sich die CPU-Auslastung mit einem kleinen Icon in der Statusleiste einblenden – entweder einmalig manuell, oder automatisch bei jedem Systemstart aktiviert. Sofern der Androide gerootet ist, kann man auch die Taktfrequenz der CPU anpassen. Die Permission-Liste ist sauber, und die Bewertungszahl (mit durchschnittlich viereinhalb Sternen bei über 10.000 Bewertungen) ist topp.

Weitere Apps zur Beschaffung von System-Informationen finden sich in dieser Übersicht. Ein paar grundlegende Informationen findet man übrigens auch mit Bordmitteln, indem man vom Home-Screen aus über *Menü → Einstellungen* geht und „Telefoninfo“ auswählt.

System Logs

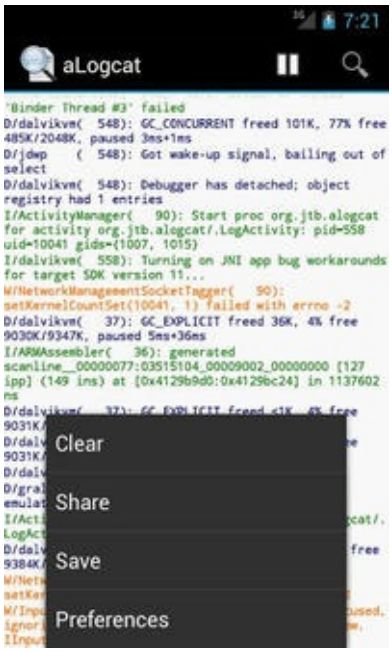
Spätestens wenn etwas nicht so tut, wie es sollte, kommt die Frage auf: „Was geht da eigentlich ab?“ Neugierige, Entwickler, und auch technisch interessierte stellen diese Frage selbst dann, wenn alles reibungslos läuft.

Auskunft darüber geben die Log-Dateien des Systems – von denen es in der Tat einige gibt. Und ebenso gibt es eine ganze Reihe von Möglichkeiten, an deren Inhalte zu kommen. Einige davon sollen hier vorgestellt werden. Doch bevor es an die Details geht, möchte ich eine grobe Übersicht voranstellen: Auf welche Arten lässt sich auf diese System-Protokolle zugreifen?

Methode	Vorteile	Nachteile
Per App	Kein weiteres Gerät und keine spezielle Software auf dem	kleiner Bildschirm (insbesondere auf Smartphones), eingeschränktes Multitasking, ab Android 4.2 meist nur noch mit root-Rechten

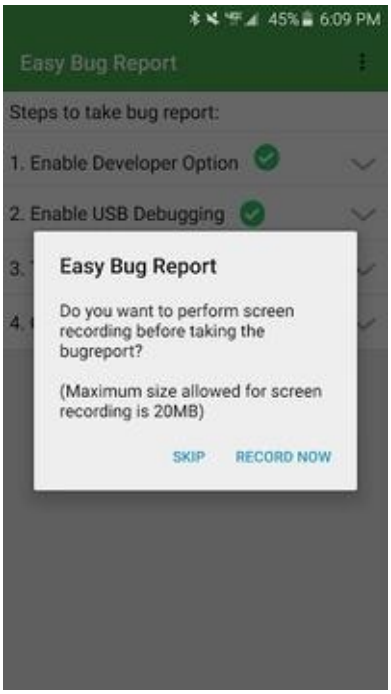
	Arbeitsrechner notwendig	möglich
Via ADB	Bequemes Arbeiten vom PC aus, einfaches Multitasking	PC mit installiertem SDK (bzw. Teilen davon) notwendig
Via SSH	Bequemes Arbeiten vom PC aus, einfaches Multitasking	benötigt SSH-Server auf dem Androiden

Zugriff per App



Ab und an benötigt auch der einfache Anwender einen Auszug aus den System-Logs. Etwa wenn eine App sich „seltsam verhält“ oder gar nachvollziehbar abstürzt, und man dies dem Entwickler mitteilen möchte: Der nämlich freut sich über derartige Details. Weiter oben habe ich bereits aufgezeigt, dass einige Apps diese Informationen sehr wohl anzeigen – nur wie bekommt man selbige dann zum Entwickler?

Die bekannteste Antwort darauf heißt sicher aLogcat, und ist mit einer Bewertung von über 4 Sternen recht beliebt. Diese App startet man, leert die Einträge kurz über den abgebildeten „Clear“ Eintrag im Kontext-Menü, provoziert den Fehler in der fraglichen App, kehrt zu *aLogcat* zurück, drückt auf den „Pause“ Button – und nun hat man die Wahl, den Log-Ausschnitt mit „Save“ auf die SD-Karte zu befördern, oder über „Share“ per Mail direkt zum Entwickler zu befördern. Das schaffen auch technisch weniger versierte!



Alternativ dazu wäre noch der Log Collector zu nennen, der ähnlich funktioniert. Oder auch Easy Bug Report, das zusätzlich zum Auszug aus den Systemlogs auch noch Informationen zum verwendeten Android-System sammelt, optional das Einschließen eines „Screen Recordings“ (Aufzeichnung von Bildschirm-Aktivitäten; ab Android 5.0) ermöglicht, und in die Mail an den Entwickler einschließt – das erspart das mühsame manuelle Zusammensuchen und stellt sicher, dass man kein wesentliches diesbezügliches Detail vergessen hat.

Weitere mögliche Alternativen finden sich u. a. in der passenden Übersicht bei *IzzyOnDroid*.

Zugriff via ADB oder SSH

Doch das ist bei Weitem nicht alles, was das System an Logs zu bieten hat – sondern lediglich das, was dem Anwender recht bequem zur Verfügung steht. Wer gern noch etwas tiefer graben möchte, hat dazu durchaus die Möglichkeit: Entweder mit einer Terminal-App direkt auf dem Androiden – oder aber bequemer vom PC aus, was jedoch die Installation der Android Entwicklungs-Umgebung (SDK; wie in obiger Tabelle beschrieben, genügt ein kleiner Teil davon) – oder alternativ den Einsatz eines SSH-Servers auf dem Androiden – erforderlich macht. Bevor ich also auf die einzelnen Logs eingehe, kurz ein paar Worte dazu.

Bei auf dem Arbeitsrechner installiertem SDK, und mit aktiviertem USB-Debugging (dies lässt sich im Systemmenü des Androiden unter *Anwendungen → Entwicklung* ein- und ausschalten) per USB-Kabel an selbigem angeschlossenen Android-Gerät lässt sich über ADB auf letzteres zugreifen. Der Name „Android Debug Bridge“ (sowie die Stelle, an der die Einstellung am Android-Gerät erfolgen muss) legen bereits nahe, dass dies hauptsächlich zur Entwicklung gedacht ist. Um Befehle vom Arbeitsrechner aus direkt auf dem Androiden ausführen zu können, stellt das SDK den Befehl `adb shell` zur Verfügung. Wer also folgendes gern bequem über die ADB ausprobieren möchte, setzt jedem Befehl einfach ein `adb shell` voran – etwa `adb shell logcat`. Die Ausgabe erfolgt dann natürlich im Terminal-Fenster des Arbeitsrechners.

dmesg

Was wird nun vom Android-System eigentlich alles protokolliert? So allerhand. Nur wird es nicht lange aufgehoben. Die meisten Protokolle liegen auf einem temporären Dateisystem (`tmpfs`), das bei Systemstart im RAM angelegt wird. Nach einem Neustart des Androiden sind also alle zuvor angelegten Einträge weg. Dafür werden aber neue geschrieben – und zwar bereits wenn das System hochfährt. Da Android auf einem Linux-Kernel läuft, kommt Linux-Anwendern natürlich sofort der Befehl `dmesg` in den Sinn – und tatsächlich findet sich das auch hier wieder. Die Ausgaben sind hier recht umfangreich; es empfiehlt sich also die Umleitung in eine Datei, etwa wie folgt: `dmesg > /mnt/sdcard/dmesg.log`.

```
$ dmesg
<6>[82839.126586] PM: Syncing filesystems … done.
<7>[82839.189056] PM: Preparing system for mem sleep
<4>[82839.189361] Freezing user space processes … (elapsed 0.05 seconds)
done.
<4>[82839.240661] Freezing remaining freezable tasks … (elapsed 0.00
seconds) done.
<7>[82839.242279] PM: Entering mem sleep
<4>[82839.242889] Suspending console(s) (use no_console_suspend to debug)
<7>[82839.252410] vfp_pm_save_context: saving vfp state
<6>[82839.252716] PM: Resume timer in 26 secs (864747 ticks at 32768
ticks/sec.)
<6>[82842.091369] Successfully put all powerdomains to target state
<6>[82842.092468] wakeup wake lock: wifi_wake
```

```
<snip>
```

Wie die meisten anderen Protokoll-Dateien, so handelt es sich auch bei `dmesg` um einen „Ringpuffer“ mit fester Größe: Ist er voll, und muss etwas neues geschrieben werden – dann fliegt etwas altes raus. `dmesg` ist ja keinesfalls ein „Boot-Log“, sondern vielmehr die Log-Datei des Kernels – und der läuft ständig im Hintergrund. Je länger also der letzte Bootvorgang her ist, desto wahrscheinlicher sind auch die entsprechenden Einträge bereits durch neuere Ereignisse überschrieben worden.

logcat

Bereits im Zusammenhang mit einigen grafischen Tools wurde `logcat` genannt. An der Kommandozeile lässt sich diesem noch einiges und viel präziser entlocken. Eine ausführliche Beschreibung dieses Befehls ist auf der Hilfe-Seite der Android-Entwickler-Website zu finden – dennoch hier ein paar kurze Hinweise.

Ein kleines Problem zeigt sich nämlich gleich beim ersten Aufruf:

```
$ logcat
Unable to open log device ‘/dev/log/main’: Permission denied
```

Anders als mit den oben genannten Apps, scheint der `shell` User also nicht so ohne weiteres Zugriff auf `logcat` zu erhalten – zumindest bei Zugriff über SSH benötigt man also root-Rechte. Dann geht es aber (und via `adb logcat` auch ohne root):

```
$ su
# logcat
--- beginning of /dev/log/system
<snip>
D/WifiWatchdogService( 3457): (android.server.ServerThread) IzzySoft
(24:65:11:67:d7:6d) does not re
quire the watchdog
I/ActivityManager( 3457): Starting: Intent {
act=android.settings.WIFI_SETTINGS cat=[android.intent.
category.DEFAULT] flg=0x10000000
cmp=com.android.settings/.wifi.WifiSettings } from pid 3529
W/InputManagerService( 3457): Starting input on non-focused client
com.android.internal.view.IInputM
ethodClient$Stub$Proxy@409f6128 (uid=1000 pid=2760)
<snip>
```

Über den Parameter `-b` lässt sich der Puffer angeben, aus dem man Informationen erhalten möchte. So liefert beispielsweise ein `logcat -b events` Daten aus dem Ereignis-Protokoll, während `logcat -b radio` Details zum „Radio“ (also dem Mobilfunk-Element) preisgibt. Auch diverse Einstellungen lassen sich mit diesem Befehl steuern – etwa die Größe des zu verwendenden Ringpuffers. Diese gelten natürlich nur bis zum nächsten Gerätestart.

```
$ su
```

```
# logcat -b events
I/am_create_service( 3457):
[1085416560,nitro.phonestats/.widget.WidgetProvider4x1$WidgetUpdateService4x1

I/am_destroy_service( 3457):
[1085416560,nitro.phonestats/.widget.WidgetProvider4x1$WidgetUpdateService4x1

I/notification_cancel( 3457): [nitro.phonestats,4,0]
I/db_sample( 4002):
[/data/data/com.lbe.security.lite/databases/lbe_trafficmon.db,INSERT OR
REPLACE INTO
 uid_traffic(wifitx, wifirx, date_uid, ce,0,com.lbe.security.lite,1]
I/force_gc( 3721): bg
I/dvm_gc_info( 3721):
[8099846449757718054,-9031259007116011474,-4008901822536468439,0]
<snip>
```

Jede Zeile der Logcat-Ausgabe beginnt mit einem Loglevel-Hinweis, gefolgt von einem Schrägstrich, gefolgt vom Ursprung des Eintrags – beispielsweise `D/WifiWatchdogService( 3457)`. Daran schließt sich sodann die eigentliche Meldung an. Folgende Log-Level gibt es: **V**erbose, **D**ebug, **I**nfo, **W**arning, **E**rror, **F**atal, **S**ilent. Der Wert in Klammern gibt die Prozess-ID an, sodass auch Querverweise zu anderen Protokollen möglich sind. Ein weiterer hilfreicher Parameter ist `-v time`, der jeder Zeile Datum und Uhrzeit des zugehörigen Ereignisses voranstellt.

dumpsys und dumpstate

Diese beiden Befehle geben ausführliche Informationen zum System: Dienste, Speicher, Broadcasts, anstehende Intents, Activities, Prozesse, Account-Informationen; Informationen über das Gerät, Build, Radio, Netzwerk, Kernel-Details … Auch hier empfiehlt sich, die Ausgaben in eine Datei schreiben zu lassen, da sie überaus umfangreich sind.

Für beide Befehle werden keine root-Rechte benötigt – jedoch fehlen bei Ausführung ohne selbige einige Details:

```
$ dumpsys
Currently running services:
  LocationProxyService
  SurfaceFlinger
  accessibility
  account
  activity
<snip>
DUMP OF SERVICE account:
Accounts: 1
  Account {name=xxxxxxx@googlemail.com, type=com.google}
<snip>
DUMP OF SERVICE alarm:
Permission Denial: can’t dump AlarmManager from from pid=7274, uid=10038
<snip>
DUMP OF SERVICE batteryinfo:
Battery History:
```

```
      -10h32m25s939ms 099 23c30040 status=discharging health=good
plug=none temp=280 volt=4117 +scre
en +wifi +wifi_running +wifi_full_lock +wifi_scan_lock +wake_lock +sensor
signal_strength=great
      -10h32m19s441ms 099 03c10040 -screen -wake_lock
<snip>

$ dumpstate
============================================================
== dumpstate: 2012-08-18 23:39:53
============================================================

Build: Gingerbread GWK74 - CyanogenMilestone2
Bootloader: 0x0000
Radio: unknown
Network: E-Plus
Kernel: Linux version 2.6.32.9-g5db7937 (a17140@zch68lnxdroid63) (gcc
version 4.4.0 (GCC) ) #1 PREEM
PT Fri Sep 16 17:57:00 CST 2011
Command line: (unknown)

-- MEMORY INFO (/proc/meminfo) --
MemTotal:         487344 kB
MemFree:           10436 kB
Buffers:           14136 kB
Cached:           145460 kB
<snip>
```

Unschwer zu erkennen, dass sich hier allerhand über System, Konfiguration, Dienste, und vieles mehr in Erfahrung bringen lässt.

bugreport

Was mag sich wohl hinter einem Befehl mit diesem Namen verbergen? Große Preisfrage. Antwort: Alles obige für Tippfaule. Gibt man `bugreport` im Terminal (oder `adb shell bugreport` via ADB) ein, flattern die Zeilen nur so über den Bildschirm – hier sollte also auf jeden Fall die Ausgabe in eine Datei umgeleitet werden (`bugreport > /mnt/sdcard/bugreport.txt`). Das Ergebnis sind dann alle von `logcat`, `dumpsys` und `dumpstate` gesammelten Informationen in einem schönen Bugreport für den Entwickler – also im Prinzip das, was auch die oben beschriebene App *Easy Bug Report* erzeugt (ohne das Screen-Recording natürlich).

Andere Logdateien

Außer den bislang genannten Standard-Logs gibt es noch einige Verzeichnisse, in denen sich Log-Dateien finden lassen. Da diese durchaus von Gerät zu Gerät (und auch zwischen verschiedenen Android-Versionen) unterschiedlich sein können, sind nicht alle hier genannten Verzeichnisse auf jedem Gerät vorhanden – die mit „AOSP“ gekennzeichneten sollten jedoch zum Standard gehören. Aber

selbst wenn das Verzeichnis existiert, kann es durchaus auch leer sein…

- `/data/anr`: Einige Trace-Logs landen u. U. hier (bei **A**pplication **N**ot **R**esponding, auch als „Force-Close“ oder „Schließen erzwingen“ bekannt)
- `/data/dontpanic` (AOSP): Crash-Logs mit Traces. Im Falle einer auftretenden Kernel-Panik finden sich hier zwei Dateien: `apanic_threads` verzeichnet den zu diesem Zeitpunkt aktiven Prozess/Thread, `apanic_console` enthält zusätzliche Informationen wie Traces und Register
- `/data/kernelpanics` (AOSP): Log-Einträge bei sog. Kernel-Panic
- `/data/panic/panic_daemon.config`: Konfigurations-Datei für den Panic-Daemon. Wenn unterstützt, lässt sich hier festlegen, wohin Panic-Logs bei manueller Auslösung (über Geräte-spezifische Tastenkombinationen) geschrieben werden (oft `/sdcard/panic_data/`).
- `/data/panicreports`: Weitere Panik-Protokolle werden bei einem „Not-Halt“ hier abgelegt (AOSP).
- `/data/tombstones`: Lustig sind sie ja, die AOSP Entwickler: Auf das Grab (englisch „tomb“) kommt ein Stein (englisch „stone“), also ein Grabstein („tombstone“), damit man sich an den Toten erinnert. Stirbt also ein Prozess unerwartet (auch als „Absturz“ oder „Crash“ bekannt), kann er sich auch so einen „Grabstein bestellen“ – der dann Informationen zu seinem Ableben enthält. Auf Linux/Unix Systemen nennt man dies „Core Dumps“.

Monitoring

Über 6.000 Nutzer bewerteten es mit durchschnittlich 4,7 Sternen, es kostet gut zwei Euro: SystemPanel scheint mir hier die App der Wahl, um einen „Bösewicht“ aufzuspüren. Beispielsweise wenn man wissen möchte, wer einem da gerade den Akku leer genuckelt hat: Vor vier Stunden war er noch randvoll, und jetzt ist die Hälfte verschwunden? *SystemPanel* hat da sicher die passenden Details zur Hand, wie im (schon etwas älteren) Screenshot zu sehen. Die Auswertung von oben nach unten: Zwischen 18 und 20 Uhr hat Google Maps die CPU mit ca. 5% Beschlag belegt – davor und danach gar nicht („Process CPU Activity“). Eine externe Stromversorgung (Ladekabel) war in diesem Zeitraum nicht angeschlossen, sondern erst wieder ab ca. 21 Uhr („Charging“). Ebenfalls im Zeitraum von 18 bis 20 Uhr verlor der Akku etwa die Hälfte seiner Ladung („Battery Charge“), was sich so ziemlich mit der Aktivität von Maps deckt. Das Ganze scheint außerdem größtenteils „im Hintergrund“ geschehen zu sein, denn laut „Device Usage“ war der Anwender selbst im

gleichen Zeitraum weniger aktiv. Da hätten wir also den Verursacher so ziemlich genau identifiziert, oder? Doch halt: Die gesamte CPU Auslastung („CPU Activity“) lag im fraglichen Zeitraum bei über 10%, also muss wohl noch jemand beteiligt sein.

Eine derartige Auswertung ist nur in der Vollversion möglich, die das „Protokollieren“ erlaubt (also im Hintergrund die Daten bis zu einer Woche lang festhält). Neben diesem System-Monitor bietet *SystemPanel* noch Geräte-Information, einen Task-Manager sowie einen App-(Un-)Installer. Einen ersten Eindruck kann man sich natürlich mit der Lite-Version verschaffen.

Doch auch mit der Lite-Version lässt sich bereits einiges anstellen. Auf der Suche nach Ressourcen-Fressern fällt gleich in der App-Liste der kleine senkrechte Balken links der App-Namen auf: Je höher dieser gefüllt ist, desto mehr CPU-Zyklen hat dieser Prozess während seiner Laufzeit beansprucht. Ist der Balken besonders hoch, und dies durch Art und Einsatz der App nicht gerechtfertigt? Dann wird es vielleicht Zeit, sich nach einer Alternative umzusehen.

Der nächste Blick gilt der Tatsache, welche App die CPU besonders lange mit Beschlag belegt hat. Dazu gibt die Übersicht der Top-Apps nach CPU Auskunft. Eine App, die man nur gelegentlich verwendet, und die (eigentlich) in der restlichen Zeit auch keine Hintergrund-Aufgaben durchzuführen hat, sollte hier nicht auftauchen. Ansonsten gilt gerade gesagtes: Ist man als Anwender nicht selbst daran Schuld (es könnte ja auch eine Fehlkonfiguration vorliegen, oder die ungewünschte Hintergrund-Funktion lässt sich etwa in der App selbst deaktivieren): Der Playstore bietet in der Regel genügend ähnliche Apps an – sofern der Entwickler der App nicht Abhilfe schaffen kann.

Wonach könnte man noch schauen? Etwa nach dem Speicherverbrauch. Verbraucht eine App viel Speicher, heißt dies natürlich nicht automatisch, dass da etwas schief liegt: Die Sache kann durchaus einen guten Grund haben. Doch der von einer App belegte Speicher steht anderen Apps nicht mehr zur Verfügung – und wird der Speicher knapp, bremst dies das System aus. Die Relationen sollten also schon stimmen: Verbraucht etwa eine einfache Stoppuhr mehr Speicher als eine Office-Suite, passt da definitiv etwas nicht zusammen.

Ähnliches wie das gerade ausführlich behandelte *SystemPanel* möchte auch System Tuner auf die Beine stellen. Diese App kann ebenfalls Aktivitäten im Hintergrund aufzeichnen und für eine spätere Analyse zur Verfügung stellen, RAM und Speicherdetails anzeigen, hat einen Task-Manager an Bord und sogar einen Terminal-Emulator, sodass man Befehle direkt an der Kommandozeile absetzen kann. Letzteres klingt ein wenig „nerdy“? Auf gerooteten Geräten lässt sich mit *System Tuner* auch der Cache der SD-Karte anpassen, der OOM-Killer konfigurieren, die CPU tweaken – man kann gar System-Apps in den „User-Space“ verschieben oder gleich ganz deinstallieren. Das ist jetzt „nerdy“. Vielleicht.

Mit dabei ist hier auch ein Widget, welches die wesentlichsten Eckdaten (CPU, RAM, Speicher) anzeigen kann. Auch einige Aktivitäten (Monitoring starten/stoppen, Task-Manager/Analyzer/Einstellungen öffnen) lassen sich von diesem aus steuern.

Weitere interessante Daten stellt auch BetterBatteryStats bereit. So bringt diese App etwa die *WakeLock* Statistiken zurück, die bis einschließlich Froyo zum Bordgepäck gehörten. Der Name suggeriert es bereits: Hier wird jemand in den Wachzustand gesperrt – und zwar die CPU. Fordert also eine App ein *WakeLock* an (und erhält es zugeteilt), kann sich die CPU nicht schlafen legen, solange dieser aufrecht erhalten wird. So etwas schlägt sich natürlich nicht zuletzt im Akku-Verbrauch nieder.

BetterBatteryStats verrät uns also, welche Apps sich hier (aber auch in weiteren Dingen, die zum Akkuverbrauch beitragen) schuldig bekennen müssen. Dazu muss *BetterBatteryStats* selbst natürlich im Hintergrund mitlaufen – denn wie sonst soll die App an eben diese Daten kommen?

Auf zwei Dinge sollte hier besonders geschaut werden: Gleich auf der Startseite der App findet sich der Abschnitt *Deepsleep* („Tiefschlaf“). Wer hat hier besonders viel CPU verbraucht? Der ist ein Kandidat zum Löschen/Ersetzen, Deaktivieren des automatischen Startens, bzw. Abstellen/Herunterregeln der automatischen Synchronisation. Schwieriger wird es bei Einträgen wie *Moderate Signal* (kein optimaler Netzempfang), *Wifi On* oder *Wifi Running* (WLAN): Das lässt sich nicht eben mit einer anderen App ersetzen. Lösungen gibt es dafür trotzdem, wie unter Automatisch (De)Aktivieren beschrieben ist.

Ein weiterer Punkt sind die bereits erwähnten *Partial Wakelocks*, wie im

Screenshot zu sehen. Hier verbergen sich die Bösewichter, die der CPU schlaflose Tage und Nächte bereiten. Besonders interessant wird es, taucht dort der *NetworkLocationLocator* auf. Das ist der Dienst von *Google Maps*, der überwacht, wo sich der Androide (und somit i. d. R. sein Besitzer) wann aufhält – um diese Informationen später an Google zu übertragen. Der ist dann an kurzer Akku-Laufzeit nicht gerade unschuldig.

Da wir gerade beim Monitoring sind: Interessant wären sicher auch Informationen darüber, welche App wann im Internet wohin unterwegs war. Diese Funktionalität bietet unter anderem die App *AFWall+*, auf die ich bereits zuvor beim Thema Firewalls näher eingegangen bin. Für Bluetooth-Aktivitäten trifft das in gleicher Weise auf *Bluetooth Firewall* im selben Kapitel zu. Weitere Apps zum System-Monitoring finden sich u. a. in dieser Übersicht.

Konfiguration

Über sieben Brücken musst Du geh'n,
Und an mehr als sieben Schräubchen dreh'n…

Das Problem dabei ist nur häufig: Welche Schräubchen, und wo sind sie zu finden? Einige davon möchte ich im Folgenden kurz benennen:

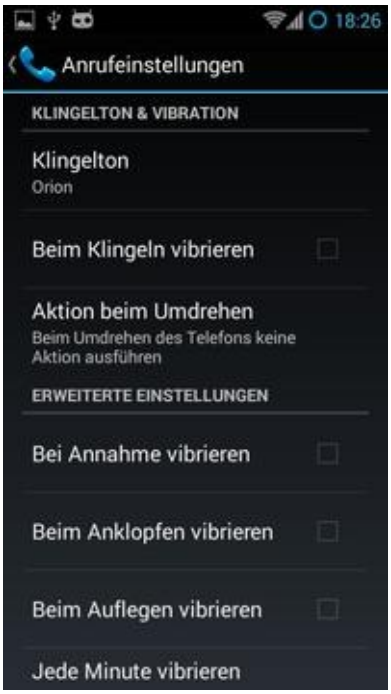
Konfiguration mit Bordmitteln

Für die grundlegenden Dinge muss man nicht gleich auf zusätzliche Apps zurückgreifen. Ein wenig „Graben" mit den Bordmitteln fördert so manches zu Tage, dessen Existenz man nicht einmal geahnt hätte. Eine gute Empfehlung bei Geräte- bzw. ROM-Wechsel ist daher: Erst einmal ganz systematisch das gesamte Einstellungs-Menü durchgehen und schauen, was so alles dazugekommen ist. Oder verschwunden, auch das kommt vor: So manche sinnvolle und wünschenswerte Einstellung wurde da mit der Zeit wieder „wegoptimiert" …

Alle Eventualitäten kann ich hier natürlich eben so wenig berücksichtigen, wie jedes kleine Detail. Die folgende Kurzübersicht orientiert sich daher an den Systemeinstellungen, die Android 4.4 anbietet – mit dem einen oder anderen Schwenk nach oben oder unten. In diese Einstellungen gelangt man, indem man vom Home-Screen ausgehend die „Menü-Taste" drückt, und im sich öffnenden Menü den Punkt „Einstellungen" (bzw. „Systemeinstellungen", „Settings", „System Settings" – verschiedene Hersteller bzw. Spracheinstellungen können natürlich leicht variieren) auswählt (alternativ findet sich der passende Eintrag auch im App-Drawer). Diese Stelle soll im Folgenden als Ausgangspunkt dienen – denn hier folgen die einzelnen „Sektionen", deren Name (jedenfalls meistens) aussagekräftig in Bezug auf die dahinter verborgenen Einstellungen ist:

Anrufeinstellungen

Nomen est omen: Hier geht es um unsere Anrufe – und dieses Menü versteckt sich (anders als die übrigen Einstellungen in den folgenden Abschnitten) in den Einstellungen der Telefon-App. Es lassen sich unter *Anrufbegrenzung* zugelassene Rufnummern verwalten, Einstellungen für die zu verwendende Mailbox treffen, Kurzantworten für ggf. abzuweisende Gespräche definieren (etwa „Bin im Meeting, rufe gleich zurück"). Man kann festlegen, ob bei abgehenden Gesprächen die eigene Nummer angezeigt werden soll, ein Profil für die Sprachqualität wählen. Auch um eventuelle Anrufweiterleitungen (ständig / bei besetzt / bei nicht-antworten / bei nicht-erreichbar) lassen sich hier einrichten bzw. verwalten – dazu gehört übrigens auch, ob und wann die Mailbox „rangehen" soll.



Zu guter Letzt wäre da noch die Sache mit den „Anklopfern“, die sich meist hinter dem Punkt *Zusätzliche Einstellungen* verbirgt. Mit etwas Glück findet sich sogar eine Möglichkeit, das Telefonklingeln durch Umdrehen des Telefons stumm zu schalten (bzw. den Anruf dadurch abzuweisen).

Ab Android 2.3 (aka „Gingerbread“) findet sich am Ende des Menüs ein zusätzlicher Konfigurations-Block: *Einstellungen für Internetanrufe*. Seitdem ist nämlich von Haus aus die Möglichkeit gegeben, mittels sogenannter SIP-Konten Telefonate über das Internet zu führen. Nachdem sich viele Netzanbieter dagegen gesperrt haben, bieten einige davon mittlerweile selbst entsprechende Dienste an. Zu den bekanntesten SIP-Anbietern zählen u. a. SIPGate, 1&1, Dus.NET und QSC. Im genannten Block lassen sich SIP-Konten verwalten und man kann festlegen, wie mit ihnen umgegangen werden soll: Auf eingehende Anrufe reagieren? Das verkürzt u. U. die Akku-Laufzeit. Für ausgehende Anrufe verwenden? Das geht, ohne WLAN, auf’s Datenvolumen. Daher ist es standardmäßig so konfiguriert, dass *nicht* auf eingehende Anrufe „gelauscht“ wird – und die SIP-Konten nur für ausgehende Telefonate an andere SIP-Nummern verwendet werden.

WLAN Einstellungen

Unter dem Punkt *WLAN* finden sich, sofern man den zugehörigen Schalter aktiviert hat, die in Reichweite befindlichen WLAN-Netze aufgelistet. Durch einfaches Antippen eines Eintrags kann man sich (ggf. nach Eingabe des WLAN-Schlüssels) mit selbigem verbinden.

Soweit ganz offensichtlich. Etliche zusätzliche Einstellungen verstecken sich hinter dem „Overflow-Menu“. Tippt man dieses an (bzw. drückt eine evtl. vorhandene Menü-Taste) und wählt den Punkt „Erweitert“, gelangt man zum rechts abgebildeten Bildschirm. Die meisten Punkte hier haben eine aussagekräftige Kurzbeschreibung – aber nicht alle. Aus der letzteren Gruppe möchte ich einige Kandidaten kurz vorstellen:

WLAN im Ruhemodus bezieht sich auf WLAN-Aktivität bei ausgeschaltetem Bildschirm. „Immer aktiviert lassen“ nuckelt vielleicht ein wenig mehr am Akku, aber dafür reißen etwaige längere Downloads auch nicht ab. „Nur wenn angeschlossen“ bezieht sich auf die Stromzufuhr. und „Nie“ erhöht

deshalb den Datenverbrauch, weil bei abgeschaltetem WLAN auf die mobile Datenverbindung zurückgegriffen wird.

Interessant ist auch der Eintrag zur *WLAN-Priorität*, sofern vorhanden. An dieser Stelle lässt sich festlegen, mit welchem Netz man bevorzugt verbunden werden möchte, sollten mehrere konfigurierte Kandidaten gleichzeitig in Reichweite sein. Nicht immer fällt die Wahl dabei auf das stärkste Signal: Manch einer bevorzugt ein „sichereres Netz“ gegenüber einem „offenen“ – oder hat andere Gründe, ein bestimmtes Netzwerk zu bevorzugen.

Datenverbrauch

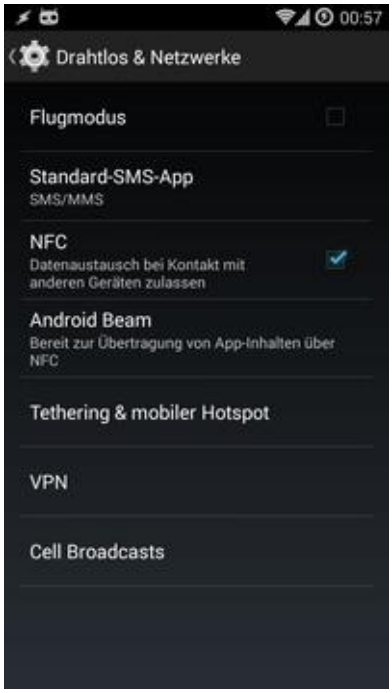
Wie zu sehen, lässt sich hier zunächst die Verwendung mobiler Daten generell ein- bzw. ausschalten. Auch wenn heutzutage fast jeder über eine Datenflat verfügt, ein sinnvoller Schalter – beispielsweise, wenn man kurzzeitig eine andere SIM-Karte einlegt, mit der keine Flat verbunden ist, bzw. einen seiner Androiden generell mit einer solchen benutzt (bei meinen Testgeräten der Fall).

Per Default nicht gesetzt ist der darauf folgende Haken, über den sich Grenzen für den mobilen Datenverbrauch aktivieren lassen. Obwohl ich selbige Grenzen auf Fantasiewerte gesetzt habe (der „rote Balken“ für „ganz Abschalten“ steht bei weit über zwei Gigabyte – ich werde im Traum nicht einmal die zwei Gigabyte für den „gelben Balken“ des Warnhinweises erreichen) ist diese Aktivierung nötig, damit man einzelne Apps „in die Schranken weisen“ kann. Dies tut man, indem man ihren Eintrag in der „Verbraucherliste“ am unteren Bildschirmrand antippt, nochmals ganz nach unten scrollt, und auch dort das mit „Hintergrunddaten einschränken“ beschriftete Kästchen markiert: Damit verbietet das System der betreffenden App, selbständig im Hintergrund ins Netz zu gehen. Dies betrifft etwa automatische Aktualisierungen von Newsreadern und Mailprogrammen – aber ebenso die Synchronisation von Kontakten und Terminen, sowie die Update-Prüfung etwa bei der *Google Play Store* App, die sich auf diese Weise ruhigstellen lässt.

Einen ähnlichen Graphen, wie er im Screenshot fürs Gesamtsystem zu sehen ist, findet man übrigens auch für jede der aufgeführten Apps separat (wenn man deren Eintrag antippt). Durch Verschieben der beiden senkrechten blauen Linien kann man den dargestellten Nutzungszeitraum anpassen – und so recht genau herausfinden, wann wieviel Datenvolumen angefallen ist, sowie wer selbiges maßgeblich verursacht hat.

Versteckte Goodies? Aber klar doch: Einfach einmal die Menü-Taste (bzw. das „Overflow-Icon“ mit den drei übereinander gestapelten Punkten) drücken. Dann lassen sich die gleichen Statistiken auch für’s WLAN anzeigen, man kann das Daten-Roaming (de-)aktivieren, sowie bekannte Hotspots als „Mobile Hotspots“ markieren. Letzteres bedeutet, dass eine Verbindung mit ihnen behandelt wird, als wäre man gar nicht im WLAN: Apps, welche die Datenverbindung nur im WLAN nutzen, halten dann also „die Füße still“ – was auch gut so ist, will man nicht das Datenvolumen des als mobilen Hotspots agierenden anderen Androiden zu stark belasten.

Drahtlos und Netzwerke



Eindeutig: Hier funkt es. Zumindest kann man an dieser Stelle (verborgen hinter dem aussagekräftigen Namen „Mehr…“) die entsprechenden Einstellungen tätigen. Mit den „Häkchen“ lassen sich einzelne Features (de-)aktivieren. So sorgt der *Flugmodus* für Funkstille: Keine mobilen Daten, kein Telefon – alles, was die empfindliche Bordelektronik stören könnte, wird damit auf einen Schlag abgeschaltet. Damit kann man den Androiden auch an Bord eines Flugzeugs nutzen – etwa um eBooks zu lesen. Handelt es sich um eine fortschrittliche Fluggesellschaft, die WLAN an Bord anbietet, so lässt sich letzteres auch im Flugmodus separat aktivieren.

Hinter Einträgen ohne „Häkchen“ verstecken sich i. d. R. weitere Konfigurations-Bildschirme: So kann man unter „WLAN“ (aus dem übergeordneten Menü) etwa in Reichweite befindliche WLAN-Netze anzeigen lassen, sich mit selbigen verbinden, gerade „abwesende“ Netze manuell hinzufügen – oder auch entfernen. Spezielle Einstellungen pro Netzwerk sind, abgesehen vom ggf. benötigten Zugangsschlüssel, hier in der Regel nicht möglich.

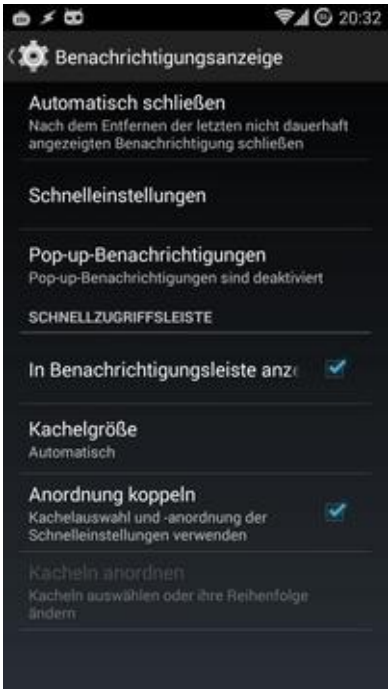
Die Bluetooth-Einstellungen (ebenfalls im übergeordneten Menü) erlauben die Konfiguration der Sichtbarkeit des eigenen Gerätes, die Vergabe eines Gerätenamens, sowie das Scannen nach in Reichweite befindlichen sichtbaren Geräten – natürlich nur, sofern Bluetooth auch aktiviert ist.

Über die VPN-Einstellungen lassen sich die entsprechenden sicheren Verbindungen ins Firmen-, Uni- oder heimische Netzwerk erstellen und verwalten.

Zu guter Letzt lassen sich über den Punkt *Mobilfunknetze* (wiederum aus dem übergeordneten Menü) das Daten-Roaming sowie die Zugangspunkte (APN) für das mobile Netzwerk verwalten, mobile Daten (de)aktivieren, die

Betreiberauswahl konfigurieren (automatisch aus verfügbaren Netzen auswählen, oder einen Betreiber fest einstellen), und mehr. Die im Screenshot außerdem sichtbaren Punkte für *NFC* und *Android Beam* stehen u. U. nicht zur Verfügung, wenn die Hardware dies nicht unterstützt.

Benachrichtigungsanzeige



In diesem Menü geht es in erster Linie um die Benachrichtigungsleiste (bei Telefonen i. d. R. am oberen Rand, bei Tablets unten) sowie den bei Öffnung derselben sichtbar werdenden Benachrichtigungsbereich. Hinter den Schnelleinstellungen verbergen sich die Einstellungen der unter Schnellumschalter beschriebenen *Quick Settings:* Kachelgröße, Anordnung, Modi der einzelnen Kacheln (z. B. zwischen welchen Klingelmodi gewechselt werden soll), und mehr. Der Menüpunkt „Pop-up-Benachrichtigungen“ ist je nach Gerät und ROM erst ab Android 5 verfügbar: Hier kann man ausgewählten Anwendungen erlauben bzw. verbieten, Benachrichtigungen in einem kleinen Popup-Fenster darzustellen.

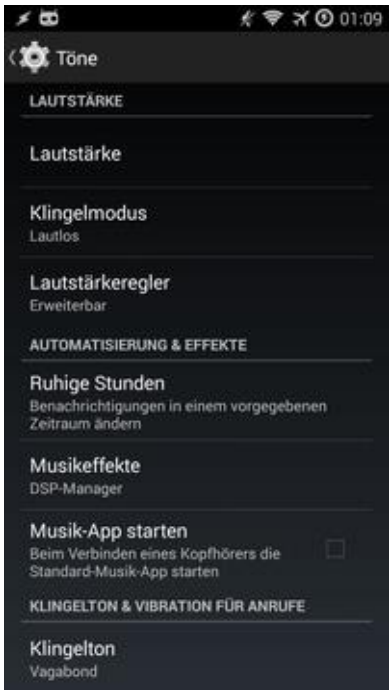
Erlauben Gerät und ROM dies, lassen sich die *Quick Settings* auch als Schnellzugriffs-Leiste direkt in den Benachrichtigungsbereich integrieren – dem widmet sich dann der entsprechende im Screenshot sichtbare Abschnitt.

Töne

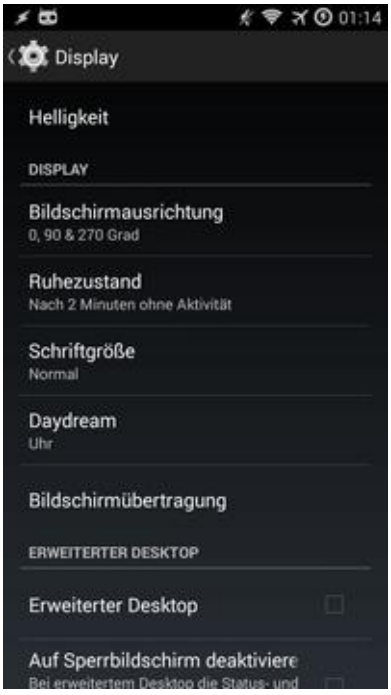
Ich höre schon die Beschwerden: „Das sieht bei mir aber ganz anders aus!“ Wahrscheinlich war das bereits bei mindestens einem der vorigen Screenshots so, und wird auch bei weiteren wieder auftreten. Was daran liegt, dass jeder Hersteller ein wenig sein eigenes Süppchen kocht – und verschiedene Dinge anders einsortiert, oder erfreulicherweise zusätzliche Dinge zur Verfügung stellt. Beschränken wir uns also auf Prinzipielles, und ignorieren Abweichungen gutmütig.

Bei den *Toneinstellungen* geht es im Großen und Ganzen also um das, was es auf die Ohren gibt (ja okay: Auf das „haptische Feedback“ trifft das natürlich nur zu, sofern das Gerät dabei direkt am Ohr befindlich ist). Separat lassen sich hier die verschiedenen Lautstärke-Einstellungen (Klingelton bei eingehenden Anrufen,

Benachrichtigungen, Medien, Wecker …) vornehmen, oder das Telefon auch einfach „stumm“ schalten. Der zu verwendende Sound bei eingehenden Anrufen lässt sich auswählen, man kann das „haptische Feedback“ (vibrieren bei Betätigung von „Soft-Keys“) sowie die Tastentöne bei der Telefonwahl einstellen. Ebenso lässt sich mittlerweile für „ruhige Stunden“ sorgen, in denen diverse Geräusche generell unterbleiben sollen – und mehr (oder auch nicht, je nach Hersteller). HTC bietet beispielsweise noch einen Taschenmodus – nicht für Billard, sondern für lauteres Klingeln, sollte sich der Androide in einer Tasche befinden (und dies auch bemerken). Motorola hat dafür einen Taschenmodus im Display-Menü (nächster Punkt), um selbiges beim Einstecken in die Hosentasche automatisch zu deaktivieren.



Display



Unter *Anzeige* (oder *Display*) finden sich die Einstellungen, die für den Bildschirm relevant sind – wie beispielsweise die Helligkeit, ob die Anzeige automatisch mit ins Querformat gedreht werden soll, wenn das Gerät gedreht wird, und ob Fenster-Animationen angezeigt werden sollen. Ebenso in diesem Menü zu finden ist das *Display-Timeout* (im Screenshot: *Ruhezustand*) – also der Zeitraum, nach dem der Bildschirm automatisch abgeschaltet werden soll (sofern der Benutzer „nichts gemacht“ hat).

Seit Android 4.2 mit dabei: Daydream. Hierbei handelt es sich um eine Art Bildschirmschoner für den stationären Betrieb (also bei angeschlossenem Ladekabel), der sich über diverse Module erweitern lässt. Von Haus aus dabei sind u. a. eine analoge und eine digitale Uhr, sowie ein „Fotorahmen“ für eine Diashow mit den eigenen Bildern. Diverse Apps bieten weitere Module.

Weiteres variiert wieder nach Hersteller, und verschiedene Custom ROMs ergänzen auch gern das eine oder andere: Motorola möchte (zumindest beim Milestone 2) automatisch erkennen, ob sich das Gerät gerade in einer Tasche befindet (und bietet für diesen Fall die Option einer automatischen Bildschirm-Abschaltung – funktioniert aber in meiner Hemdtasche schon Mal nicht). Dann

gibt es Hersteller, die durch eine automatische Anpassung der Bildschirm-Helligkeit Akku sparen wollen, und anderes mehr.

Die im Screenshot sichtbare Einstellung *Erweiterter Desktop* ist übrigens eine Spezialität von *CyanogenMod.* Sie deaktiviert auf Geräten mit virtuellen Navigationstasten dieselbigen, um mehr Platz auf dem Bildschirm zu schaffen. Und wer jetzt noch verwundert auf den Eintrag *Bildschirmübertragung* schielt: Dahinter verbirgt sich die Möglichkeit, den Bildschirm des Androiden per Miracast auf ein anderes, kompatibles Gerät zu übertragen – ein Feature, das mit Android 4.4 (Kitkat) ergänzt wurde.

SD-Karte und Telefonspeicher

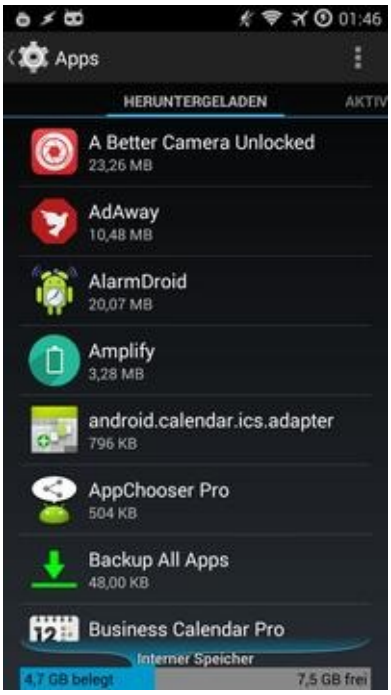
Wenn jemand seinen Speicherplatz sucht: Hier wird derjenige fündig. Angezeigt wird der insgesamt verfügbare Speicherplatz je Medium, und wie sich selbiger aufteilt. Im Kapitel Datenstrukturen habe ich ja bereits ausgeführt, dass unsere Androiden über mehrere „Datenträger“ verfügen; aufgeführt werden in diesem Menü jedoch lediglich die, welche für den Anwender auch von Interesse sind: Der Gerätespeicher (`/data`), sowie die interne und externe SD-Karte (so vorhanden).

Interessant ist an dieser Stelle auch zu sehen, wie sich der belegte Speicherplatz zusammensetzt: Wieviel entfällt etwa auf Apps und deren Daten? Oder auf den SD-Karten auf Bilder & Videos, Audio (Musik, Klingeltöne, etc.) und Downloads? Wo liegen die „dicken Brocken“? Für eine grobe Analyse ist das durchaus geeignet.

Darüber hinaus lässt sich die externe SD-Karte aus diesem Menü heraus formatieren, oder auch „entnehmen“ (unter Windows heißt das „sicher entfernen“; gemeint ist, das System auf die physikalische Entfernung aka Entnahme der SD-Karte vorzubereiten). Über das „Overflow Menu“ (die am oberen rechten Rand sichtbaren drei „übereinander gestapelten Punkte“) kann man darüber hinaus festlegen, wie der Speicher bei Verbindung mit dem PC freigegeben werden soll (z. B. per MTP).

Anwendungen

Unter *Anwendungen verwalten* (oft einfach nur „Anwendungen“ oder „Apps“)



werden alle installierten Apps aufgelistet. So kann man hier nachschauen, wie viel Platz die Apps selbst belegen, wie viele Daten sie verwalten, und wie groß der vorgehaltene Cache gerade ist. Und man kann all diese Dinge auf einen Schlag oder auch einzeln löschen – wobei das Löschen einer App auch automatisch das Löschen ihrer Daten und ihres Caches nach sich zieht, nicht aber umgekehrt. Dies ist bei der *Playstore* App manchmal ganz hilfreich, wenn die sich gerade wieder verhaspelt hat: Cache löschen, neu initialisieren lassen.

Organisiert ist dieser Bereich in verschiedenen „Tabs“: „Heruntergeladen“ für die Apps, welche man selbst installiert hat, „Auf SD-Karte“ für entsprechend verschobene Apps, und „Alle“ für eine komplette Liste, welche auch die vorinstallierten System-Apps mit einbezieht.

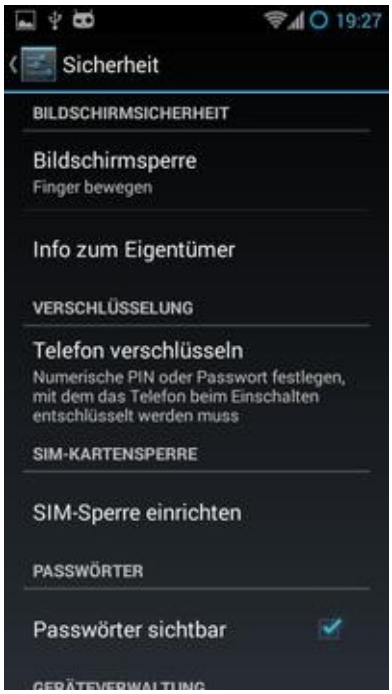
Einen Tab habe ich jetzt noch nicht erwähnt: Hinter den *Ausgeführten Diensten* (oder „Aktiv“) verbirgt sich ein Mini-Task-Manager. Dieser zeigt u. a. an, welche Apps gerade ausgeführt werden, ggf. seit wann, und wie viel Speicher sie belegen. Ebenfalls an dieser Stelle zu sehen: Welche Apps welche Services gestartet haben. Der bekannteste Übeltäter dürfte da Google Maps mit seinem *NetworkLocationService* sein – läuft ständig, auch wenn man Maps nie genutzt hat (wird vermutlich für die netzbasierte Standortbestimmung genutzt – doch sicher sagen kann das scheinbar niemand). Am unteren Rand sieht man hier auch die Gesamtbelegung des Arbeitsspeichers.

Durch Antippen eines App-Eintrags erhält man die Möglichkeit, einen Dienst anzuhalten – oder auch nicht. Das „oder auch nicht“ bezieht sich zum Einen auf die Möglichkeit, diese Aktion abzubrechen; zum Anderen aber auch auf den Fakt, dass die meisten Dienste ohnehin nach wenigen Sekunden wieder gestartet sind (worauf der Dialog auch hinweist).

Standort und Sicherheit

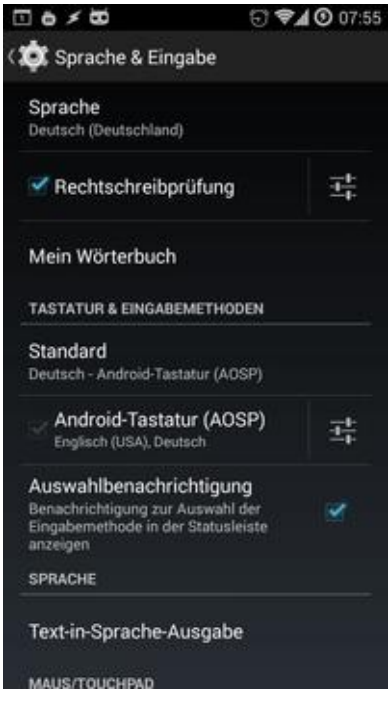
In älteren Android-Versionen war dies ein gemeinsamer Menüpunkt. Da aber die Überschrift „Standort *und* Sicherheit“ wohl ein wenig irreführend war, wurden zwei separate Einträge daraus. Richtig hätte es heißen müssen: Standort *oder* Sicherheit. Spätestens ab Android 2.3 (Gingerbread) gibt es nämlich nur noch eins von beidem: Wer die (energiesparendere) Standortbestimmung über Mobilfunkmasten und WLAN-Netzwerke nutzen möchte, muss Google seine Standortdaten zur Verfügung stellen – ein Opt-Out bei gleichzeitiger Nutzung des Services ist nicht (mehr) vorgesehen.

Daneben (oder der Anordnung nach besser „darunter“) gibt es natürlich die Möglichkeit, die Standort-Bestimmung über GPS ein- bzw. auszuschalten. Sofern die andere Ortsbestimmungsart nicht aktiviert wurde, sammelt hier (hoffentlich?) keiner Daten mit – und solange nicht tatsächlich eine Ortsbestimmung durchgeführt wird, kostet das auch nicht wirklich Energie. Wenn aber doch, dann gleich richtig: Da kann es durchaus passieren, dass nach ca. vier Stunden Navigation der Akku aufgibt.



Im Menü „Sicherheit“ (siehe Screenshot) finden sich nun die wirklich sicherheitsrelevanten Einstellungen: Bildschirmsperre (PIN, Password, Pattern; befindet sich ggf. in einem eigenen Haupt-Menü namens *Bildschirmsperre*), SIM-Sperre, Geräteverwaltung (Einrichten/Verwalten von Geräteadministratoren), sowie Zertifikate. Ebenso lässt sich von hier aus die Geräteverschlüsselung aktivieren, und auch die Freischaltung von „fremden Installationsquellen“ (alternative Märkte, Direkt-Installation von APK Dateien) findet sich in diesem Menü. Einige Hersteller bzw. Custom-ROMs können durchaus weitere Punkte integriert haben: Bei CyanogenMod findet sich hier u. a. auch die Rechteverwaltung für Apps – sofern sie nicht in ein eigenes Hauptmenü namens *Datenschutz* umgezogen ist (ab Kitkat).

Sprache und Eingabe



In diesem Menü geht es um die Lokalisierung. Das betrifft zum Einen Spracheinstellungen, nach denen sich auch die Apps (sofern entsprechend angepasst) richten sollen – damit man auch versteht, was man da liest. Durch das zugehörige Land werden darüber hinaus auch Dinge wie das Datums- und Uhrzeitformat, sowie das zu verwendende Einheiten-System (metrisch oder imperial) beeinflusst.

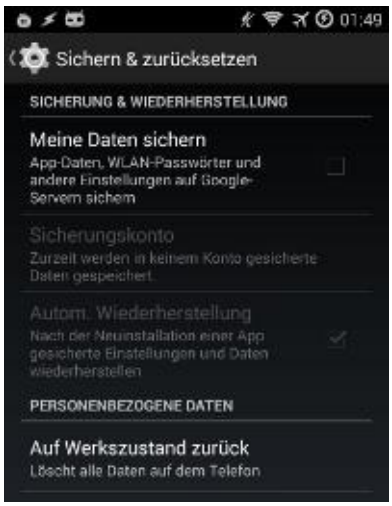
Zum Zweiten geht es aber auch um die zu verwendende Tastatur („Eingabe-Methode“) und deren Konfiguration. Auch selbst installierte zusätzliche Tastatur-Apps lassen sich hier konfigurieren. Und zwar inklusive der von ihr unterstützten Sprachen, die sich im laufenden Betrieb sodann meist durch ein Wischen über die Leertaste oder Antippen einer speziell dafür vorgesehenen Taste wechseln lassen. Verfügt das Gerät über ein „Hardware

Keyboard“ (was nur für wenige aktuelle Geräte zutrifft), taucht dieses als „physikalisches Keyboard“ hier ebenfalls auf.

Im Screenshot nicht mehr zu sehen, aber normalerweise ebenfalls in diesem Menü zu finden (ggf. verborgen hinter dem Settings-Icon der jeweiligen Tastatur) sind die Einstellungen für die automatische Korrektur: Sollen Tippfehler automatisch korrigiert werden, oder das zweimalige Drücken der Leertaste gleich das Satzzeichen ergänzen? Möchte man bei Auswahl eines Eingabefeldes ein Symbol zur Wahl der Eingabemethode in der Statusleiste sehen?

Schließlich finden sich hier auch noch die Einstellungen für „Text in Sprache“ – die Android Vorlese-Funktion, auch TTS (Text-To-Speech) genannt. Das Benutzerwörterbuch sucht man zunächst vergeblich, will man etwa ein falsch hinzugefügtes Wort wieder loswerden. Dazu muss man in die Einstellungen der jeweiligen Tastatur-App wechseln, wo sie sodann unter „Textkorrektur“ zu finden sein sollten.

Datenschutz



Schon wieder etwas Doppeldeutiges: Möchte Google hier unsere Daten schützen (indem es sie auf seine Server sichert) – oder sollen wir hier unsere Daten vor Google schützen, indem wir alle Häkchen entfernen? Sollte letztere Variante die richtige sein, ist die Umsetzung nur teilweise gelungen – denn K&K (Kalender und Kontakte) ignorieren diese Einstellung geflissentlich, und synchronisieren trotzdem. Wohl einer der Gründe, warum dieser Menüpunkt nunmehr *Sichern & Zurücksetzen* heißt.

Scherz beiseite: Gemeint ist, dass die eigenen Einstellungen sowie Anwendungsdaten auf die Server von *Google Inc.* gesichert werden bzw. nach einem Werksreset von dort wieder hergestellt werden sollten. Warum der Konjunktiv an dieser Stelle? Dieser Service funktioniert leider nicht immer so zuverlässig, wie man es sich wünschen würde. So liest man im Bug-Tracker von etlichen Fällen, in denen Apps und Daten nur teilweise oder überhaupt nicht wieder hergestellt wurden – von 2011 bis heute, und auch bei Nexus-Geräten.

Und was wird hier überhaupt gesichert? Für Geräte, auf denen Android 3.x oder höher eingesetzt wird, finden sich Details dazu im Benutzer-Handbuch. Demnach werden etwa folgende Daten gesichert (und hoffentlich bei Bedarf auch wieder hergestellt):

- Android-Einstellungen wie WLAN-Netzwerke mit ihren Passwörtern, das

Benutzer-Wörterbuch, etc.
- Einstellungen vieler Google-Apps, einschließlich der Lesezeichen des Web-Browsers
- Die aus dem Playstore installierten Apps
- Daten von anderen Apps, sofern die Entwickler dies explizit implementiert haben

Es handelt sich hierbei also mitnichten um ein vollständiges Backup: Daten von Apps werden nur dann mitgesichert, wenn diese Apps das auch explizit unterstützen – was für die meisten Apps nicht zutreffen dürfte. Und natürlich sind auch die auf der SD-Karte abgelegten Dateien nicht in diesem Backup enthalten. Was bedeutet, dass man sich zumindest nicht auf dieses Backup allein stützen sollte. Auch ist es nicht jedermanns Ding, seine Daten einer Cloud anzuvertrauen – selbst wenn dies (Achtung!) die Voreinstellung ist.

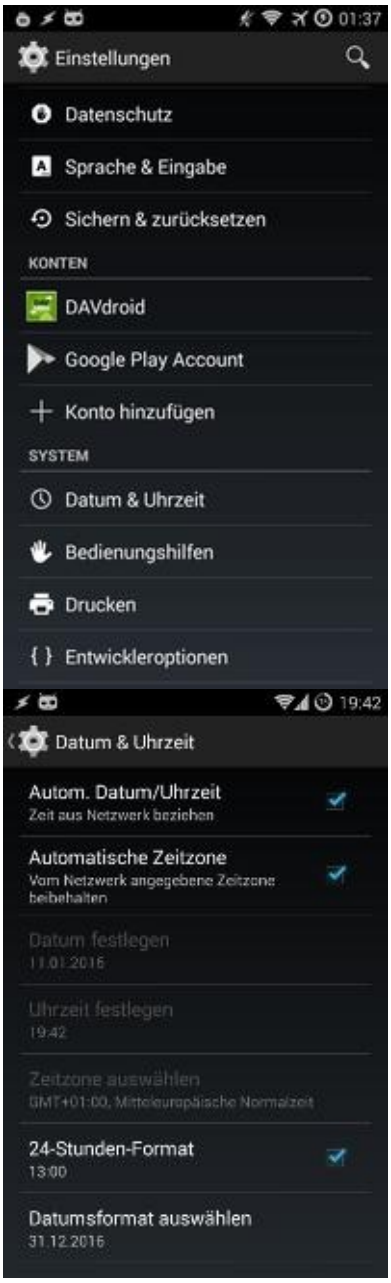
Ab Android 6.0 soll dies besser werden: Dann kommt eine neue Art von Google-Cloud-Backup, die keine explizite Unterstützung auf App-Seite benötigt – und alles zu *Google Drive* sichert. Bleibt abzuwarten, wie gut das dann funktioniert – und ob man unser Einverständnis wieder einmal einfach voraussetzt.

Der Werksreset ist übrigens ein weiterer Punkt, der sich auf vielen Geräten in diesem Menü findet: „Zurücksetzen auf Werkseinstellungen“ (oder einfach „Speicher zurücksetzen“). Entweder, wenn gar nichts mehr geht und alles nur noch verrückt spielt – oder wenn man das fragliche Gerät veräußern möchte, denn bei einem „Werksreset“ werden alle Anwenderdaten einschließlich der vom Anwender installierten Apps gelöscht. Lediglich die eventuell vorhandene SD-Karte bleibt dabei i. d. R. völlig unberührt (abgesehen von App2SD).

Konten

Dieser Punkt ist mittlerweile im Hauptmenü selbst integriert (siehe Screenshot). Hier fällt die Länge der Liste zum Teil sehr unterschiedlich aus. Was zum Einen vom Hersteller und seiner „Zwangsbeglückung“ zu tun haben kann – zum Anderen aber nicht zuletzt davon abhängt, wie viele „Konten“ (z. B. Google-Accounts) man eingebunden hat. Natürlich spielt es auch eine Rolle, ob der Hersteller einen Geräte-Spion (HTC: Sense.COM, Motorola: Blur, etc.) zur Auffindung eines „verlorenen“ Gerätes integriert, und der Anwender diesen eingerichtet hat und nutzt.

Nach Auswahl des Google- oder in meinem Falle DAVDroid-Kontos lässt sich hier auch konfigurieren, ob Kontakte und Kalender automatisch synchronisiert werden sollen (ebenso kann man an dieser Stelle die Synchronisation auch manuell anstoßen). Gut versteckt – und bei Google-Konten natürlich per Default

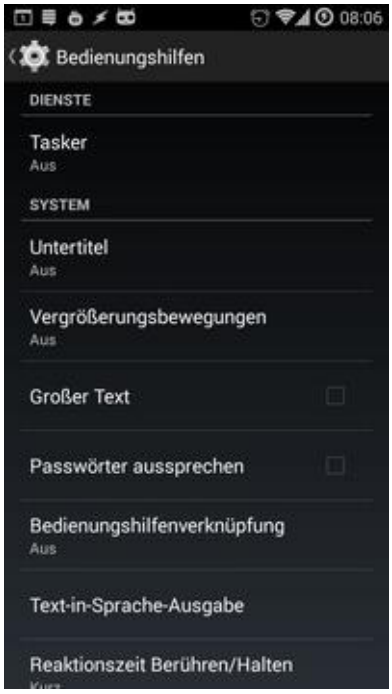


aktiviert. Wer ein „zentrales Häkchen“ für die automatische Synchronisation vermisst, findet dies in den „Quick Settings“ des Benachrichtigungsbereichs.

Datum und Uhrzeit

Dieses Menü sollte einfach und verständlich sein – eigentlich erübrigt sich dazu fast jede Erklärung. Hier lässt sich bei Bedarf die Zeit manuell korrigieren, falls das Netzwerk des Betreibers das nicht hinbekommt (oder das Gerät ohne SIM-Karte/Netzzugang betrieben wird); normalerweise sollte dies recht selten bis nie nötig werden. Einzig die Umschaltung vom 12- auf den 24-Stunden-Modus (oder umgekehrt) sowie die Anpassung des Datumsformates kann, je nach Vorliebe, für den Einen oder die Andere interessant werden – oder bei Reisen in entferntere Gefilde das manuelle Setzen der Zeitzone (etwa, weil man die „heimatliche Zeit“ beibehalten will).

Bedienungshilfen



Was zum Geier sind denn nun Bedienungshilfen – Tastatur und Spracheingabe hatten wir doch schon? Derartige „Eingabehilfen“ (alternativer Name dieses Menüs) sind hier auch nicht gemeint; dieser Punkt richtet sich vielmehr an Menschen mit Behinderungen. Da wäre zuerst die *Sprachwiedergabe*: Hat der Benutzer so schlechte Augen, dass er zwar den Text sieht, ihn aber nicht lesen kann, kann er ihn sich durch „Antippen“ vorlesen lassen. Ähnlich verhält es sich mit dem *Zoom-Modus* zur Vergrößerung von Elementen.

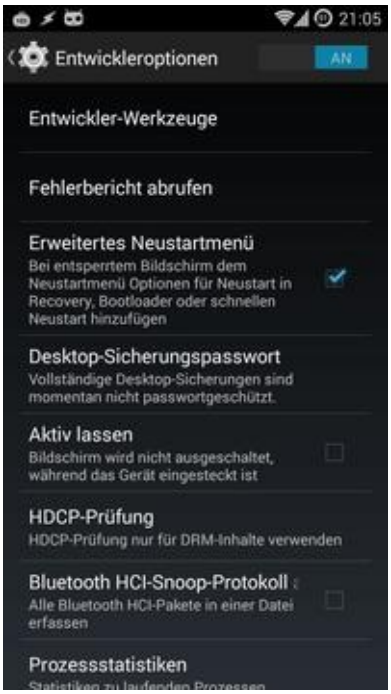
Sofern installiert, taucht auch Tasker hier auf – was aber eine andere Bewandnis hat: Diese App braucht die entsprechenden Berechtigungen, um im „Benutzer-Interface“ (also auf dem Bildschirm) dargestellte Elemente auswerten zu können.

Einfach und verständlich hingegen ist der Punkt *Ein/Aus beendet Anruf* (sofern vorhanden). Gut für alle die, die den entsprechenden Button auf dem Bildschirm

nicht finden können: Statt das Display auszuschalten, wird dann per Ein/Aus-Taste ein laufendes Gespräch beendet. Natürlich nur, wenn eines läuft. Das hat manchmal den dummen Nebeneffekt, dass man mitten im Telefonat versehentlich auflegt – weil man zum „eben Mal etwas nachschauen“ das Display anschalten wollte … Das Gegenstück dazu wäre *Anruf mit Home-Taste annehmen* – eine Erklärung dafür dürfte sich erübrigen.

Über die *Bedienungshilfenverknüpfung* lässt sich eine Funktion zum schnellen Einschalten derselben aktivieren – eine detailliertere Beschreibung findet sich direkt hinter dem Menüpunkt.

Entwickleroptionen



Wie der Name es bereits andeutet, sind diese in erster Linie für Entwickler gedacht. Doch auch für „Otto Normalanwender“ finden sich hier sinnvolle Schalterchen. Andere wiederum können unerwartete Auswirkungen haben – weshalb man beim „Spielen“ Vorsicht walten lassen sollte.

Ist dieser Menüpunkt (aus genanntem Grunde) nicht sichtbar, muss er zunächst freigeschaltet werden. Wie das geht, ist im Punkt Telefoninfo beschrieben. Von den für Endanwender nützlichen Einstellungen möchte ich nun einige kurz vorstellen. Die übliche Anmerkungen: Nicht alle davon sind auf jedem Gerät/ROM verfügbar. Und allesamt kann man erst verwenden, nachdem der „Hauptschalter“ umgelegt wurde.

Erweitertes Neustartmenü: Dies sollte man auf jeden Fall aktivieren – selbst wenn man mit Begriffen wie „Bootloader“ oder „Recovery“ nichts anfangen kann: Ein schneller Neustart mit zwei Fingertipps erweist sich öfter nützlich, als einem lieb ist! Das *Desktop-Sicherungspasswort* bezieht sich auf per ADB vom Desktop-Rechner aus angestoßene Sicherungen bzw. deren Verschlüsselung. Wer diese Sicherungen häufiger erstellt, mag das zugehörige Passwort vielleicht gern hier hinterlegen.

Das *Aktiv lassen* des Bildschirms bei angeschlossener Stromquelle ist in erster Linie für Entwickler interessant. *Daydream* funktioniert auch ohne.

Die Einstellung *Android-Debugging* ist wiederum von zentralem Interesse: Dieser Hauptschalter wird für alle ADB-Tätigkeiten benötigt – sei es `adb backup` oder der Zugriff mittels `adb shell`. Für den Zugriff per USB-Kabel genügt das Häkchen an dieser Stelle; möchte man das Ganze auch drahtlos verwenden, muss man auch *ADB über Netzwerk* aktivieren – nach jedem Boot erneut.

Testet jemand häufiger verschiedene Apps, und möchte dem jeweiligen Entwickler bei Fehlverhalten gern aussagekräftige Daten zur Verfügung stellen, dürfte auch der Punkt „Fehlerberichte im Menü Ein/Aus“ die Augen leuchten lassen: Ein Bugreport ist damit schnell über das erweiterte “Power-Menü“ verfügbar.

Von allen anderen Einstellungen sollte man besser die Finger lassen – sofern man nicht weiß, was sich dahinter verbirgt. Eine Ausnahme stellt noch der letzte Punkt dar: „App Beenden“ lässt uns eine ungehorsame App durch langes Drücken der „Zurück-Taste“ zwangsbeenden. Eine Funktion, die ich ungern missen möchte.

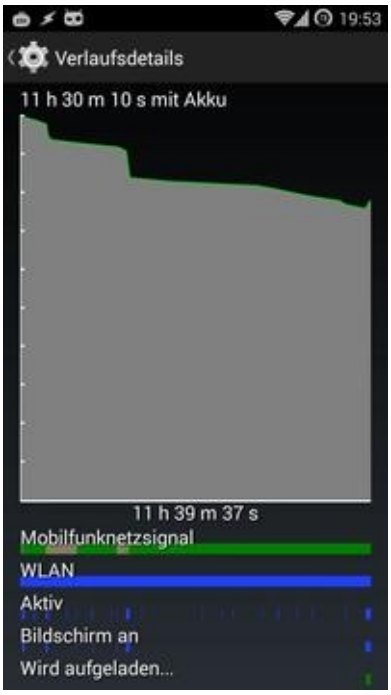
Telefoninfo

Das letzte Menü birgt eine Menge Potential. Zuallererst finden sich hier die vom Betreiber bereitgestellten Systemsoftware-Updates – für all die Ungeduldigen, die nicht auf die entsprechende automatische Benachrichtigung warten können oder wollen. Dies ist der einzige Punkt, in dem auch eine Aktivität ausgelöst wird: Das Einspielen eines Updates, so denn eines gefunden wird.

Naja, das stimmt nur so halb – denn traditionsgemäß verbergen sich in diesem Menü auch immer ein paar Ostereier („Easter Eggs“). So sollte man sich einmal den Spaß machen, bei den Software-Informationen wie wild auf die Android-Version zu tippen (Spoiler: Das löst eine Animation aus – und schaltet ab Android 4.2 gleichzeitig ein zusätzliches Daydream-Modul frei). Ab Android 4.2 wird übrigens die gleiche Aktion auf dem Feld der Build-Nummer mit dem Freischalten der Entwickler-Optionen belohnt.

Alle anderen Punkte spucken hingegen jede Menge nützlicher Informationen aus – und sind auf verschiedenen Geräten unterschiedlich zusammengefasst. Sie informieren über die Android-Version, Kernel & Baseband, CPU, RAM, die eigene Telefonnummer, das aktive Netzwerk (Provider) sowie dessen Signalstärke und Typ, Roaming, Mobilnetzstatus, IMEI, MAC-Adressen und Betriebszeit und geben schließlich diverse weitere Informationen zum verwendeten System.

Unter dem Punkt *Akku* (der gelegentlich in einem eigenen Hauptmenü-Punkt zu finden ist) findet man in der Regel auch eine Liste der größten Verbraucher. Mit Gingerbread kam hier ein neues Feature hinzu: die Statistiken zum Batterieverbrauch (siehe Screenshot). Zu diesen gelangt man, wenn man im



Menü *Akku* auf den „Balken“ am oberen Bildschirmrand tippt – der sich bei genauerem Hinschauen als Mini-Graph entpuppt, wie der „große Graph“ auf dem Screenshot deutlich macht.

Während dieser „große Graph“ veranschaulicht, wie (un)gleichmäßig der Akku entladen wurde, finden sich unmittelbar darunter einige wertvolle Details. So beispielsweise zum Telefonsignal (die einzelnen Farben sind im Anhang erklärt): Ein sattes Grün steht für eine sehr gute Signalstärke; wird es heller, war das Signal schwach. Kritisch wird es bei roten Stellen: Hier war gar kein Signal vorhanden – das Smartphone hat aber mit aller Gewalt versucht, eines zu finden. Im Luftschutzbunker relativ sinnlos: Wenn es kein Signal gibt (etwa aufgrund starker Stahlbeton-Wände, oder weil man sich etwa irgendwo abseits jeglicher Zivilisation befindet), bedeutet dies nur eins: Hoher Akku-Verbrauch, da das Gerät mit voller Leistung auf (nutzlose) Suche geht. Tritt dies jeden Tag zur gleichen Zeit auf, sollte man die zugehörige Örtlichkeit prüfen – und ggf. per Automat hier in den Flugmodus wechseln.

Die übrigen vier Graphen sind eigentlich selbsterklärend: Wann war WLAN aktiviert, wann das Gerät beschäftigt, wann der Bildschirm eingeschaltet, und wann das Ladekabel angeschlossen (natürlich beidseitig).

Ausgewählte bzw. häufig genutzte Einstellungen

Schnell und ohne große Umstände an die wichtigsten Einstellungen gelangen – darum geht es hier. Sofern ein pures Umschalten im Sinne von „An/Aus“ gefragt ist, haben wir dieses Thema ja bereits bei den Schnellumschaltern abgehandelt. Doch nicht alles ist dort enthalten. Daher schauen wir nach, was es noch so gibt.

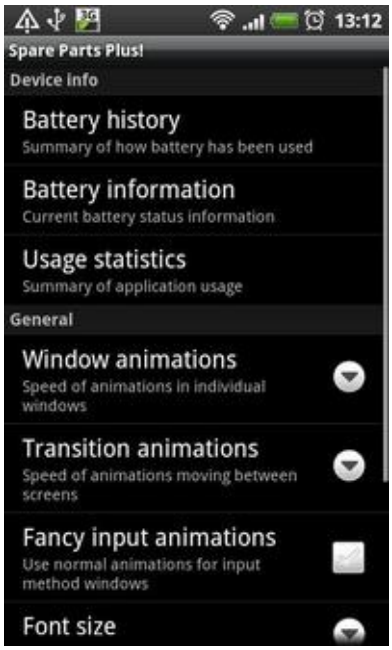
„Schnelle Einstellungen“ – auf Englisch hieße das „QuickSettings“. Und mindestens vier Apps tragen genau diesen Namen. Seit jedoch Android selbst ein Feature dieses Namens mitbringt, haben die meisten davon keine neuen Versionen mehr gesehen – dabei weisen Androids „Quick Settings“ bei weitem nicht alle ihre Features auf; Schieberegler für Lautstärke oder Helligkeit fehlen beispielsweise völlig.

Diese und mehr bietet z. B. die App Quick Control Panel. Wie bei den nativen

„Quick Settings“ sind auch hier Toggle-Switches mit an Bord, die bei langem Drücken in die zugehörigen Systemeinstellungen führen. Dazu kommen – neben den genannten Schiebereglern – auch App-Shortcuts sowie eine „Music Section“ (siehe Screenshot). Die Anordnung lässt sich dabei frei konfigurieren.

Wer sich bei den anderen Alternativen umschauen möchte, findet diese in der passenden Übersicht.

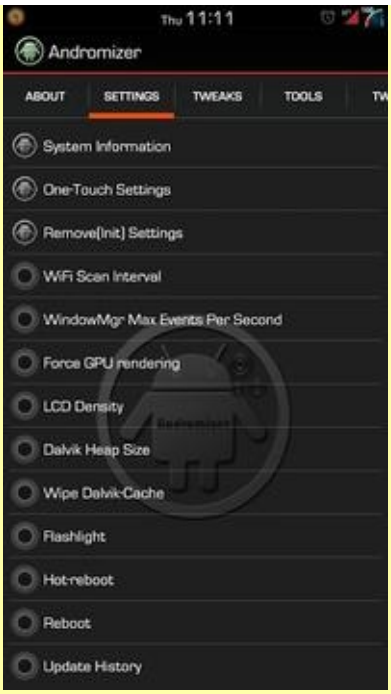
Zusätzliche bzw. versteckte Einstellungen



Wenn es um zusätzliche Konfigurations-Möglichkeiten geht, die das System von sich aus in der Form nicht anbietet, fällt vielen älteren Semestern sicher als erstes *Spare Parts* ein. Oder besser noch, da aktueller und ohne Werbung: Spare Parts Plus. Diese App stellt nicht nur einzelne zusätzliche Einstellungen zur Verfügung, die nicht selten wesentlich mehr ins Detail gehen als die Bordmittel (beispielsweise Animationen, siehe Screenshot: Nicht nur ein simples globales „An/Aus“, sondern explizite Einzelheiten). Anwender mit „schlechten Augen“ werden sich zudem über die Möglichkeit freuen, die Größe der Systemschrift anpassen zu können. Und darüber hinaus versorgt uns *Spare Parts Plus* auch noch mit etlichen detaillierten System-Statistiken, wie ebenfalls auf dem Screenshot erkennbar.

Die App hat den Machern des bekannten Custom-ROMs CyanogenMod so gut gefallen, dass sie sie lange Zeit in ihre ROM eingebaut hatten – mittlerweile sind jedoch die einzelnen Einstellungen an den entsprechenden Stellen direkt integriert. Fehlen sie, lässt sich die App gratis aus dem Playstore laden – und wem sie gut gefällt, der kann für knapp anderthalb Euro auch zur Pro-Version greifen. Anders als die Screenshots auf der App-Seite es vermuten lassen, wird die App offensichtlich noch aktiv gepflegt, und der Entwickler stellt weiterhin Updates zur Verfügung.

Wer seinen Androiden mit root-Rechten ausgestattet hat, greift (zusätzlich? stattdessen?) vielleicht eher zu Blade Buddy oder dessen Pro-Version, beide aus dem gleichen Hause wie *Spare Parts*. Die App integriert sich in *Spare Parts* (bzw. lässt sich aus *Spare Parts* heraus aufrufen), und erlaubt die Konfiguration etlicher systeminterner Schalterchen. So lässt sich das Auslösegeräusch der Kamera abschalten, die Dalvik-VM detailliert anpassen, oder auch die Boot-Animation deaktivieren. Aktionen wie ein schneller Reboot, oder das beschreibbare Einhängen der Systempartition stellen die App ebenfalls vor



keinerlei Probleme. Allerdings wurde sie seit 2013 nicht mehr aktualisiert – sodass einige der Einstellungen spätestens mit Android 5.0 nicht mehr greifen dürften – insbesondere für die Laufzeit-Umgebung Dalvik, die seit Lollipop von ART abgelöst wurde.

Noch umfangreichere Anpassungen sind etwa mit Andromizer möglich, wie der Screenshot andeutet. Mit dieser App kommen u. a. noch CPU-Settings, ein Build.prop Manager, diverse Optimizer (CPU, RAM, etc.) – aber auch diverse Tools wie Kernel-/Recovery-Updater, ROM-Cleaner, und Backup hinzu. Sogar das TWRP-Recovery wird hier mit integriert.

Natürlich wird vor all zu wildem „Gespiele“ gewarnt. Einige dieser Einstellungen hat das Android-Team nicht umsonst vor dem „Otto Normalanwender“ versteckt, und für nicht-rootler unzugänglich gemacht: Das falsche Schräubchen zu weit gedreht, und da fliegt das Blech weg. Man kann sich sein System auch kaputt-konfigurieren – also bitte mit Bedacht vorgehen!

Aber benötigt man zum Sichtbarmachen „versteckter“ Einstellungen nun wirklich immer extra Apps? Oder gibt es da auch andere Möglichkeiten, an diese zu gelangen? In der Tat gibt es für so manche „Secret Settings“ auch passende „Secret Codes“ – wie beispielsweise `*#*#4636#*#*`. Diese gibt man so ein, als wollte man bei der entsprechenden Nummer anrufen. Und in diesem Beispiel öffnet sich dann ein recht informatives Menü, welches auch *Spare Parts* und Secret Phone Settings nutzen. Wobei bei letzterer App das dauernde „Info“ auf den Buttons leicht irritiert: Ja, es werden etliche hilfreiche Informationen angezeigt – dabei lässt sich aber durchaus auch die eine oder andere Einstellung anpassen.

Neben diesen Tools für vielfältige allgemeine Einstellungen gibt es jedoch auch noch einige, die sich auf Einzelbereiche spezialisiert haben. So lassen sich etwa mit dem WiFi Advanced Config Editor für jedes konfigurierte WLAN erweiterte Einstellungen wie der Index des WEP-Schlüssels oder WPA Enterprise Parameter (z. B. `wpa_supplicant`) anpassen, mit Set DNS zu verwendende NameServer und andere DNS-Parameter konfigurieren, und (root-Rechte vorausgesetzt) sogar mit ProxyDroid die Verwendung eines Proxy-Servers (auch abhängig von der Verbindungsart: mobil oder WLAN) erzwingen.

Nicht ganz uninteressant wäre auch die Möglichkeit, die Synchronisation von Kalender, Kontakten, Google Mail & Co. auf die Zeiten zu beschränken, zu denen man mit einem WLAN und/oder einer Stromquelle verbunden ist – AutoSync

macht dies möglich. Root-Usern vorbehalten bleiben allerdings Dinge wie die Anpassung der CPU-Taktung (das bekannteste Tool hierfür wäre sicherlich SetCPU, aber auch der CPU-tuner ist definitiv einen Blick wert!) oder der Bildschirm-Auflösung (z. B. mittels Screen Size and Density – hier sieht man an den Beispiel-Screenshots auch wunderbar die Auswirkungen).

Und wer jetzt noch mehr Möglichkeiten oder Alternativen sucht, wird in dieser Übersicht fündig.

Automatisierung

Einstellungen sind ja gut und schön. Einmal gemacht, merkt sie sich unser Android auch brav. Aber passen sie auch zu jeder Gelegenheit? Tagsüber auf der Baustelle muss ein eingehender Anruf möglichst laut signalisiert werden, damit man ihn auch mitbekommt. Doch wer lässt sich schon gern nach Feierabend etwa beim Schmusen von einem Donnerwetter in der Hose (oder wo immer das Gerät gerade steckt) aufschrecken? Genauso unpassend ist ein lautes Klingeln etwa während eines Seminars, oder im Konzertsaal (auch wenn bei heutiger „moderner Musik“ das halbe Publikum denken könnte, das gehöre dazu).

Und das ist erst der Anfang: Das Umschalten zwischen verschiedenen Klingeltönen und Lautstärken vergessen wir eben so leicht, wie das Aktivieren von GPS vor dem Start der Kartensoftware – oder das Hochschrauben des Display-Timeouts vor Aktivierung des RSS- bzw. eBook-Readers. Wie viele andere Szenarien ich jetzt hier nicht einmal angeschnitten habe, will ich dabei gar nicht mutmaßen.

Für all diese Dinge muss es doch auch Helferlein geben? Wir sind doch sicher nicht die Ersten, die vor diesem Problem stehen! – Das ist beides korrekt. Und so gibt es Apps, die uns gewisse Dinge abnehmen können: Zeitgesteuert, abhängig vom aktuellen Standort („location-based“ oder „ortsbasiert“ genannt), oder auch kombiniert bzw. bei anderen Events. Und es gibt Tasker. Der passt in keine dieser Kategorien hinein: Tasker ist eher ein „Wundertool“, „Schweizer Offiziersmesser“, oder einfach Das Ultimative Werkzeug (Englisch: „TUT“, **T**he **U**ltimate **T**ool). Braucht aber für Einsteiger auch mindestens ein „Tut“ (Tutorial) – doch dazu später.

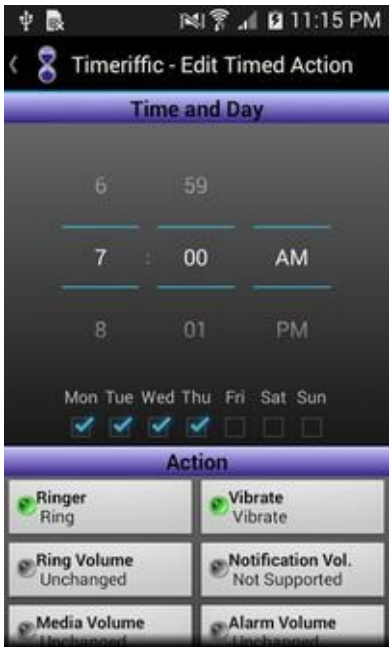
Ein Tipp noch vorweg: Automatisierungs-Tools (die übrigens in dieser Übersicht zusammengefasst sind) sollten möglichst nicht auf der SD-Karte installiert werden. Sonst werden sie u. U. beendet, wenn man seinen Androiden per USB mit dem Computer verbindet – oder nach dem Booten gar nicht erst gestartet.

Zeitschaltuhren

Geht es um reines zeitbasiertes Schalten, ist Timeriffic nicht nur das bekannteste, sondern auch das am besten bewertete Tool. Ein Großteil der oben genannten Aufgaben lässt sich damit erledigen – sofern es mit klassischen Profilen zu tun hat: Lautstärken, Stummschaltung, Vibrator. Auch Flugmodus, WLAN, und Bluetooth beherrscht die App. All diese Dinge lassen sich zu beliebigen Zeitpunkten auf beliebige Werte stellen – was übrigens auch für die Bildschirm-Helligkeit gilt.

Übrigens nicht nur Uhrzeit-basiert: Auch die Wochentage lassen sich dazu auswählen. Der Screenshot zeigt das ganz gut: So lassen sich etwa für das

Wochenende andere Regeln erstellen, als man sie werktags einsetzt – denn wer möchte schon sonntags früh um sieben Uhr von einem lauten Klingeln geweckt werden.



Das war es aber auch bereits – mehr ist nicht drin. Phone Schedule bietet ähnliches an. Und damit ist dann – für die reinen Zeitschaltuhren, die aktuell noch gepflegt sind – auch schon das Ende der Fahnenstange erreicht.

Ortsbasiertes Schalten

Die mir bekannten Apps für reines ortsbasiertes Schalten werden offensichtlich allesamt nicht mehr länger gepflegt: Die letzten Updates stammen von 2013, oder sind gar noch älter. Was nicht unbedingt tragisch ist, da viele Apps aus dem nächsten Abschnitt diese Funktionalität mit abdecken – wie beispielsweise Llama, welches ursprünglich eine rein ortsbasierte App war (was der volle Name auch noch andeutet: *Llama Location Profiles*).

Abhängig von den jeweiligen Standorten (der Reiter *Areas* im Screenshot), kann sich *Llama* neben genannten klassischen Audio-Profilen auch um Dinge wie APNs, Hintergrundbilder, WLAN, und den Flugmodus kümmern – oder aber den Androiden neu starten, Apps starten/beenden, und mehr. Und eigentlich muss sich *Llama* llangsam einmal Umbenennen, und gehört sodann in die nächste Kategorie verschoben. Denn mittlerweile lassen sich auch andere Events zum Schalten heranziehen: Angeschlossene Ladekabel, Docking-Stations, Kalender-Ereignisse, in der Nähe befindliche Bluetooth-Geräte …

Womit die App fast mit *Locale* (siehe unten) gleichauf zieht – nur dass es die entsprechenden Plugins nicht unterstützt.

Zwei Besonderheiten dieser App verdienen noch Erwähnung. So können ausgewählte Kontakte das Telefon auch zum Klingeln bringen, wenn *Llama* selbiges eigentlich „stumm“ geschaltet hat. Und obwohl die App völlig gratis ist, verlangt sie keine Berechtigung für den Internet-Zugang – was einerseits Werbefreiheit verspricht, und andererseits bedeutet, dass keine Daten irgendwohin abwandern. Die Ortsbestimmung erfolgt übrigens nicht über GPS, sondern über Mobilfunkzellen-IDs: Das spart Akku (die Zellen muss das Telefon ja ohnehin kontaktieren).

Zeit-, Orts- und Eventbasiertes Schalten

Der Name verrät bereits, was sich mit MyProfiles schalten lässt: Die klassischen Telefon-Profil Eigenschaften. Hinzu kommen noch WLAN, Bluetooth, der Flugmodus, sowie Bildschirm-Helligkeit und Ausrichtungsmodus. Auch ortsbasiertes Schalten unterstützt die App (wie bei *Llama* erfolgt die Erkennung dabei über Mobilfunkzellen) – und neben zeitbasiertem Schalten eine ganze Reihe Events: Etwa NFC-Tags, oder ein angeschlossenes Ladekabel bzw. ein bestimmter Akkustand. Damit lassen sich energiehungrige Aktionen einschränken – wobei ich beispielsweise an ein Dimmen des Displays bei niedrigem Akkustand, oder an das Abschalten der WLAN-Funktionalität denke.

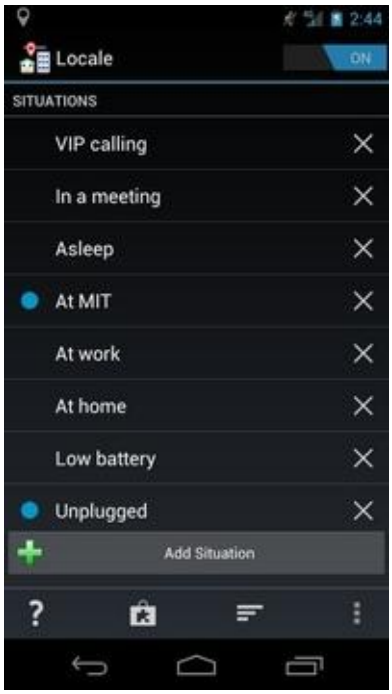
Mindestens genauso praktisch ist das Schalten anhand bestimmter Kalendereinträge: Bei Sitzungen klingelt es nicht mehr, und auch beim Theaterbesuch ist es ruhig. Ebenfalls möglich sind Aktionen bei Anschließen eines Headsets (etwa das automatische Starten des Music-Players) oder Einstecken des Gerätes in eine Docking-Station (so vorhanden).

Einen besonderen Hype genießt IF (ehemals *IFTTT* für „IF This Then That“) – hier lässt sich nahezu alles mit nahezu allem verbinden (Stichwort: Internet der Dinge). Das Besondere: Man benötigt zunächst einen Account, dann lassen sich „Rezepte“ über das Internet austauschen; eine Anbindung an die entsprechende Datenbank ist in der App integriert. Somit lassen sich nicht nur „lokale Ereignisse“ als Auslöser nutzen – sondern auch Dinge, die im Netz passieren. Details beschreibt etwa die zugehörige Wikipedia-Seite:

IFTTT ermöglicht Benutzern „Rezepte“ nach dem Motto „If this then that“ („Wenn dies dann das“) zu erstellen. Der „this“-Teil eines Rezepts wird „Trigger“ genannt. Beispiele für einen Trigger sind „Ich wurde auf einem Foto bei Facebook markiert“ oder „Ich habe bei Foursquare eingecheckt”. Der „that”-Teil eines Rezepts wird Aktion genannt. Beispiel für Aktionen sind “sende mir eine Textnachricht” oder „Erzeuge eine Statusmeldung auf Facebook”. Die Kombination aus Trigger und Aktion wird Rezept genannt. IFTTT bietet Trigger und Aktionen für mehr als 100 Dienste

wie Twitter, Foursquare, Flickr, und Facebook.

Lokale Dinge sind natürlich ebenso möglich – etwa das Telefon leiser zu stellen, wenn man im Büro ist. Außerdem besteht die Möglichkeit, Elemente der Hausautomatisierung zu integrieren – um beispielsweise auf die Alarmanlage oder das Thermostat zu reagieren, und vieles mehr. Eine Auflistung unterstützter Dinge findet sich auf der englischen Wikipedia-Seite.



Nicht unerwähnt bleiben darf natürlich der Klassiker in dieser Rubrik: Lange Zeit galt Locale als die einzige echte Lösung. Und was war davor?

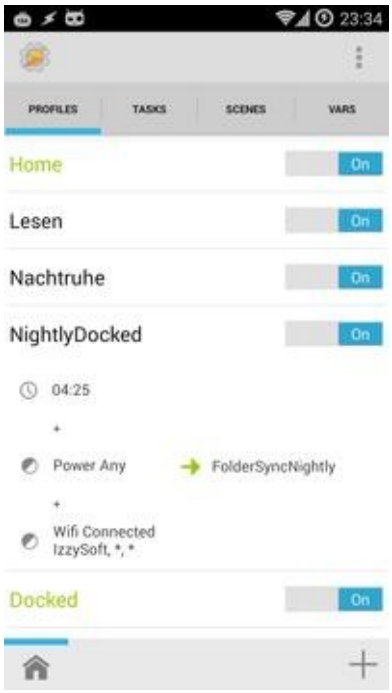
Judge Robert Restaino jailed 46 people when a mobile phone rang in his New York courtroom and no one would admit responsibility. So we invented ***Locale****. Problem solved.* (*Im Mai 05 hat Richter Restaino 46 Leuten eine Haftstrafe verordnet, da keiner zugeben wollte, dass es sein Telefon war, welches im New Yorker Gerichtssaal klingelte. Da erfanden wir* ***Locale****. Problem gelöst.*) So ist es auf der Website zu lesen. Vergleichsweise einfach zu erfassen und bedienen, wäre *Locale* auch heute noch im Ranking weiter vorn – gäbe es da nicht das Handicap des Preises: Gut sieben (bzw. mittlerweile zehn) Euro sind im App-Market einfach ein wenig viel, da schaltet so mancher bereits ab, ohne weiter zu lesen.

Locale bietet im Prinzip alles, was auch *MyProfiles* im Portfolio hat. Ob die Bedienung hier intuitiver oder einfach nur vergleichbar ist, muss jeder für sich entscheiden. Ich habe *Locale* eine Zeit lang benutzt und sehe eigentlich wenig, was *MyProfiles* zu weniger als dem halben Preis nicht bieten würde – abgesehen vielleicht von der längeren Erfahrung, die das Locale-Team mitbringen dürfte – und den zahlreichen Plugins, die es für *Locale* (sowie *Tasker*) gibt. Wobei ich nicht sagen kann, ob das geniale Prinzip des „Schichten-Aufbaus“ dort auch übernommen wurde: Bei *Locale* bauen die Profile nämlich aufeinander auf. Das „unterste“ Profil bildet die „Basis“, und die „darüber liegenden“ Profile wenden lediglich davon abweichende Einstellungen an. Wird also „weiter oben“ ein Profil inaktiv, tritt automatisch der aus den „darunter liegenden“ aktiven Profilen bestimmte Zustand ein – wobei die Eigenschaften der „weiter oben liegenden“ jeweils Vorrang haben. So lässt sich jederzeit leicht erkennen, was Sache ist.

Tasker

Ein eigenes Kapitel für eine einzige App sei ungerechtfertigt? Dem kann ich in sofern zustimmen, als dass man Tasker ein eigenes Buch widmen könnte:

„Android-Automation mit Tasker“. Vielleicht mache ich das ja sogar einmal. *Tasker* ist nicht irgend *eine* Automatisierungs-App – es ist *die* Automatisierungs-App schlechthin. Ist etwas mit Tasker nicht hinzubekommen, dann geht es schlicht und ergreifend nicht. Auch nicht mit einer anderen App. Selbst in Foren findet sich kaum eine Frage, auf die nicht irgend jemand mit „Tasker“ geantwortet hätte …



Tasker kann alles das, was auch *Locale* kann, einschließlich der Verwendung der Plugins. Ach was: *Tasker* kann alles, was irgend eine der vorgenannten Apps kann. Und das in beliebiger Kombination. Und mehr. Viel mehr!

Auch bei *Tasker* erstellt man „Profile“. Dargestellt werden diese in der Übersicht etwa so, wie man es auf dem Screenshot erkennen kann – zweispaltig mit dem Namen als Überschrift: Links der (bzw. die) Auslöser („Kontext“ genannt), rechts das, was passieren soll („Tasks“, also Aufgaben). Bereits an diesem einen Screenshot lässt sich erahnen, wie vielfältig *Tasker* einsetzbar ist: Als Auslöser lassen sich neben den bisher besprochenen „gewöhnlichen“ Dingen wie Zeit, Tag und Ort etwa Apps im Vordergrund, Hardware/Software Status, Ereignisse, Widgets, oder Timer einsetzen. Auch der Erhalt einer SMS mit einem bestimmten Text, oder der Anruf von einer bestimmten Telefonnummer sind möglich.

Ebenso vielfältig schaut es bei den Möglichkeiten für Tasks aus. Aufgrund ihres Umfangs mussten diese bereits in grauer Vorzeit auf Kategorien verteilt werden. Außerdem lassen sich in Tasks auch Schleifen, Variablen und Bedingungen verwenden.

Im „Tasks“-Screenshot sieht man bereits einen Eintrag „Plugins“: Ja, auch Tasker ist in der Lage, mit den zahlreichen *Locale*-Plugins zu arbeiten. Darüber hinaus bieten auch immer mehr Apps direkte Schnittstellen für Tasker: So lässt sich etwa bei *FolderSync* (im Kapitel Dateisynchronisation vorgestellt) eine Synchronisation anstoßen oder eine Einstellung ändern. Auch für *Dropbox* gibt es diverse Plugins. Und nicht zuletzt lassen sich bei Bedarf sehr maßgeschneiderte Lösungen wie etwa ein Diebstahlschutz oder ähnliches realisieren.

Wenn *Tasker* also das Non-plus-Ultra ist, warum nutzt es dann nicht jeder? Dafür gibt es sicher mehrere Gründe. Die Einen schreckt der Preis ab: Mit etwa fünf Euro ist die App nicht ganz billig – wenn auch, an Funktionsumfang und gebotenen Möglichkeiten gemessen, absolut nicht teuer. Wer jedoch nur einen kleinen Teil dieser Fähigkeiten benötigt, die zudem noch von einem anderen

(vielleicht gar kostenlosen) Tool abgedeckt werden – dem kann man es ja wohl kaum verdenken, wenn er sich dann auch für letzteres entscheidet.

Ein weiterer Punkt ist die mit der App verbundene Komplexität: Wenn jemand behauptet, er hätte sich „mal eben *Tasker* installiert und vollständig eingerichet“ – so hat dieser Mensch glatt gelogen. Selbst jemand, der Tasker bereits „kennt wie seine Westentasche“, macht dies nicht „mal eben so nebenbei“. Zugegeben: Je besser man sich damit auskennt, je mehr setzt man damit auch um. Und hätten unsere Androiden einen Temperatur-, Wasser- *und* einen Kaffeesensor, könnte *Tasker* mit gerooteten Geräten auch Kaffee kochen (Kontext: Wasser < 90°C mit Kaffeepulver; Task: Übertakte CPU, starte alle Benchmarks. Exit-Task: Benchmarks stoppen, CPU wieder normalisieren) …

Zugegeben: *Tasker* mag ein wenig gewöhnungsbedürftig sein, und fordert zumindest ein wenig Einarbeitung. Aber auch wenn gelegentlich behauptet wird, man bräuchte dafür ein Diplom: So kompliziert ist es nun wirklich nicht. Und mir ist niemand bekannt, der den kleinen Aufwand zu Beginn später bereut hätte. Um neuen Anwendern den Einstieg zu erleichtern, habe ich bei *IzzyOnDroid* eine kleine Sammlung der wichtigsten Ressourcen erstellt – hier finden sich Links zu Tutorials, Wikis und weiteren Hilfsquellen.

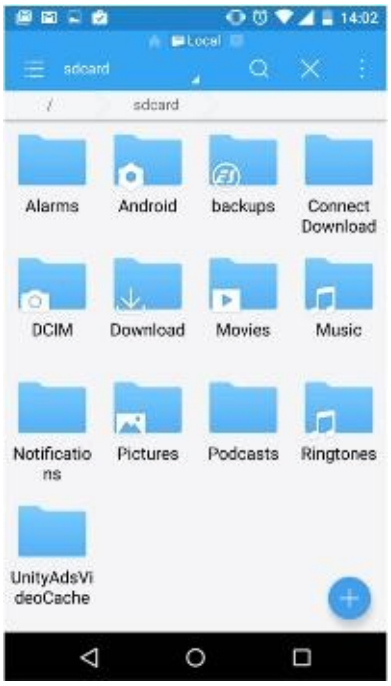
Übrigens: *Tasker* gibt es auf der Homepage in einer gratis-Version für sieben Tage zum Testen. Die Vollversion sollte man anschließend auch hier erwerben: Gegenüber der Playstore-Version kommen dann noch einige Features wie Verschlüsselung hinzu, und außerdem sind die Kosten hier geringfügig niedriger.

Dateimanager

Gibt es ihn, den ultimativen Dateimanager? Viele Anwender werden auf diese Frage ein klares „Ja“ zurückgeben. Nur der Rest der Antwort – nämlich das nebensächliche Detail, um welche App es sich dabei wohl handeln möge – dürfte dabei sehr unterschiedlich ausfallen. Denn was für den einen „ultimativ“ ist, mag für jemand anderen völlig uninteressant sein – das Umgekehrte gilt natürlich ebenso. Doch zum Glück ist auch in diesem Bereich die Auswahl an für Android verfügbaren Apps umfangreich: Angefangen bei solchen, die sich ausschließlich um die SD-Karte kümmern – bis hin zu denen, die nebenbei auch noch den ganzen Rest des Androiden (einschließlich der Hardware) verwalten wollen, ist alles dabei.

Bei einem derart großen Umfang kann ich natürlich nicht jede App vorstellen. Daher werde ich im Folgenden für die mir wichtig erscheinenden Bereiche die aus meiner Sicht besten Kandidaten herausgreifen – eine Auswahl, die natürlich sehr subjektiv ist. Dennoch hoffe ich, damit einen recht guten Überblick zu geben – und niemandem auf die Füße zu treten. Und gleich vorab: Eine Übersicht zu diesem Thema befindet sich an diesem Ort.

Für den “normalen Anwender”



Mein persönlicher Favorit war lange Zeit ES Datei Explorer. Dieser bringt alles mit, was man im Alltag wirklich benötigt: In Listen- oder Icon-Ansicht lässt sich das komplette lokale Dateisystem durchforsten. Bilder, Videos, und auch verschiedene Dokumente kann die App dabei gleich anzeigen, ZIP-Dateien ein- und auspacken, RAR-Dateien auspacken, und mehr. Zugriff auf den heimischen PC aus dem heimischen WLAN heraus? Über SMB gar kein Problem, sogar den Server findet *ES* selbst. Mal eben etwas vom FTP-Server holen? Ebenfalls kein Thema. Sogar SFTP (SSH), Bluetooth und Dropbox werden unterstützt.

Aber auch an all die, die gern mehr wollen, ist gedacht: Seit einer Weile beinhaltet die App auch Unterstützung für Wurzelmenschen. Ein minimaler Lesezeichen-Manager hilft, wichtige Orte schnell wiederzufinden, und installierte Apps lassen sich auflisten, sichern oder deinstallieren (für Details wird auf das App-Management des Android-Systems zurückgegriffen). Und wem das noch immer nicht reicht, der kann die App mittels weiterer AddOns erweitern: Um einen „großen“ Lesezeichen-Manager, einen Task-Manager, einen Sicherheits-Manager …

Eine der wenigen Sachen, die ich beim *ES Datei Explorer* erst lange suchen

musste, ist die Anzeige von Details zu den Dateien in der Listenansicht. Dies scheint für den ASTRO Datei-Manager selbstverständlich zu sein. Davon abgesehen deckt sich der Funktionsumfang dieser beiden Apps offenbar annähernd: Auch *ASTRO* versteht sich auf die Navigation durch das komplette Dateisystem, einschließlich dem Kopieren, Verschieben, oder Löschen von Dateien. Mittels Modulen lässt er sich erweitern, so dass gleiches auch für Bluetooth, SMB und SFTP gilt. Ein Task-Manager scheint bereits an Bord, und für das Betrachten von Bildern sowie die Verwaltung installierter Apps ist ebenfalls gesorgt. Sogar der Zugriff auf die Cloud ist damit möglich.

„Wurzeltools“ bringt *ASTRO* keine mit. Auf der Beliebtheits-Skala liegen die beiden genannten Apps relativ dicht beisammen, was die Bewertungen im Playstore betrifft.

Mittlerweile ebenfalls für Android zu haben ist einer der bekanntesten Dateimanager aus der Windows-Welt. Die Rede ist vom Total Commander – und der kann sich offensichtlich in Sachen Funktionsumfang mit den beiden anderen vorgestellten Kandidaten durchaus messen. Wie der Screenshot erkennen lässt, kann man durch das lokale Dateisystem ebenso navigieren wie durch SMB-Freigaben – wobei das Kopieren einzelner Dateien oder auch ganzer Verzeichnisse möglich ist. Ein Lesezeichen- sowie ein App-Manager sind vorhanden, die Archiv-Unterstützung (ZIP, RAR) entspricht der des ES Datei Explorers; an Netzwerk-Protokollen stehen neben SMB offensichtlich noch FTP und SFTP (aka SCP/SSH; via Plugin) zur Verfügung. Auch ist ein Text-Editor direkt integriert.

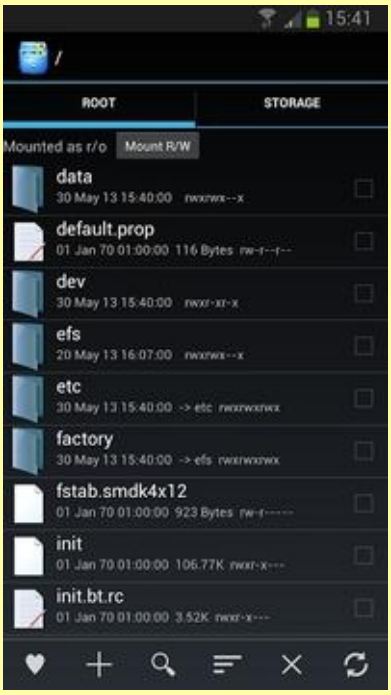
Der Funktionsumfang lässt sich darüber hinaus mittels Plugins noch erweitern: Etwa um WebDAV- und Cloud-Zugriff (Google Drive, Microsoft Live, Dropbox). Optionale root-Unterstützung ist hingegen von Haus aus dabei. Wer mag, kann auch im „guten alten Notron-Commander-Stil“ zwei Panels nebeneinander anzeigen lassen – was besonders Tablet-Anwender begeistern dürfte (bei „zu wenig Platz“ gibt es diesen Modus alternativ auch „virtuell“).

Seine gute Bewertung im *Google Play Store* legt nahe, dass der *Total Commander* sicher keine schlechte Wahl darstellt.

Und wer sich jetzt fragt, wohin ich mich nach meinem Abschied vom *ES Datei Explorer* gewendet habe: Open Explorer. Open Source, benötigt nur halb so viele Permissions, und hat einen vergleichbaren Funktionsumfang. Darüber hinaus ist

er auch bei F-Droid zu haben, von wo ich nach „Entgooglifizierung“ meiner Geräte bevorzugt meine Apps beziehe.

Spezielles für „Wurzelmenschen“ (root)



Wie bereits gezeigt, lässt sich der *ES Datei Explorer* auch als „root-Explorer“ nutzen (*Open Explorer* ebenso). Der Ghost Commander ist dazu ebenfalls in der Lage. Dennoch gibt es natürlich Apps, die sich genau auf dieses Gebiet spezialisiert haben.

Eine dieser Apps trägt den Namen Root Explorer. Selbstredend lässt sich damit das gesamte Dateisystem erkunden – und was nicht schon beschreibbar ist, wird beschreibbar gemacht (wie der Screenshot zeigt, wenn man genau hinschaut).

Doch damit ist das Thema noch lange nicht erschöpft: Mit dabei ist auch ein SQLite Database Viewer, damit man Einblick in die Datenbanken des Systems nehmen kann. Denn davon ist vieles in derartigen Datenbank-Dateien gespeichert – beispielsweise die Anruflisten, die Kurznachrichten, und natürlich auch die Kontaktliste sowie der Kalender. Da sind Dinge wie der integrierte Text Editor, die Möglichkeit zum Erstellen und Entpacken von ZIP sowie Tar/GZip Dateien bzw. das Auspacken von RAR-Archiven schon fast normal.

Ferner mit dabei: Cloud-Support (Google Drive, Box, Dropbox), Unterstützung von Windows-Freigaben (SMB), ein App-Manager, und mehr. Als spezielle root-Features kann man das Ausführen von Skripten, (Neu-) Einbinden von Dateisystemen, Ändern von Dateiberechtigungen und -eigentümern, sowie den binären APK XML-Viewer betrachten. Lesezeichen und die Möglichkeit, Dateien via Mail, Bluetooth, etc. zu versenden, runden das Ganze schließlich ab.

Toolboxen mit integriertem Dateimanager

Um die Sache am anderen Ende zu beginnen: Der bereits zuvor genannte *ES Datei Explorer* lässt sich mit AddOns zu einer derartigen Toolbox ausbauen. Ein Lesezeichen-Manager, mit dem sich Bookmarks für Dateien, Verzeichnisse, etc. verwalten lassen, ist bereits integriert; ein Task-Manager kommt gleich mit einem praktischen Widget daher, welches wahlweise auch Auskunft über Resourcen geben kann; einen App-Locker namens ES App Locker – und sogar ein

ChromeCast-Plugin. Abgesehen einmal von diversen Themes, sollte jemand die Optik pimpen wollen.

Prinzipiell interessant klingt Advanced Users Toolbox: ZIP-Support, App Installer/Uninstaller, Akku-Manager, Market-History-Cleaner, generischer Cache-Cleaner und mehr gesellen sich hier zum integrierten Dateimanager. Wer sein Gerät gerootet hat, kommt in den Genuss weiterer Features – wie etwa einem Cleaner für den Dalvik-Cache. Der Screenshot lässt es allerdings bereits vermuten: Diese App hat schon länger kein Update mehr gesehen – seit Mitte 2012, um genau zu sein.

Noch mehr zu bieten scheint etwa Advanced Tools (und ist dabei auch noch aktuell): An Bord sind u. a. ein Dateimanager mit ZIP-Support sowie Unterstützung für Bluetooth und FTP, ein System-Manager mit umfangreichen System-Infos und der Möglichkeit, etwa CarrierIQ zu entdecken (falls dieser Spion auf dem Gerät vorhanden ist), App-Manager, Terminal, Sensor-Analyzer, GPS Status & Fix, und mehr. Auch hier gibt es auf gerooteten Geräten erweiterte Funktionalitäten – etwa zur Anpassung der Pixeldichte des Displays, einfrieren von Bloatware, und mehr.

Weitere Kandidaten dieser Kategorie finden sich noch in der eingangs genannten Übersicht. Alternativ ist es sicher nicht verkehrt, sich für jeden benötigten Bereich eine App auszuwählen, die sich auf diesen spezialisiert hat – und ihre Aufgaben daher mit Bravour löst (insbesondere, wenn man nur ausgewählte Features benötigt). Einige davon werde ich im vierten Teil dieses Buches zum Thema Tuning noch benennen.

Androiden vom PC aus verwalten

Eine Übersicht zu diesem Thema findet sich an dieser Stelle bei *IzzyOnDroid*.

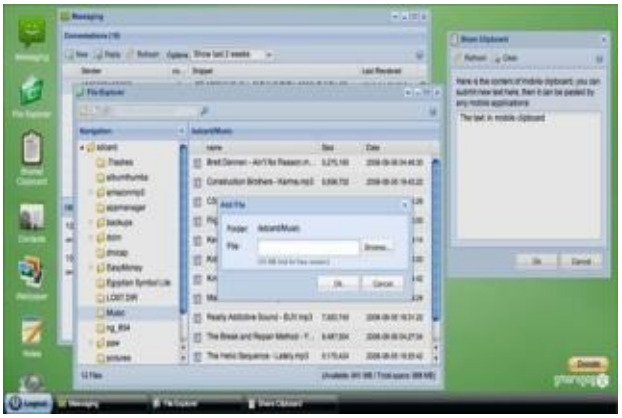
Für den „normalen Anwender“

MyPhoneExplorer ist in diesem Bereich sicher die bekannteste und beliebteste App. Ihre größte Einschränkung ist jedoch, dass sie nur mit einem Windows-Betriebssystem kommunizieren kann, auf dem überdies dafür auch eine spezielle Anwendung installiert sein muss. Mac und Linux-Nutzer stehen also außen vor. In dieser Richtung ist auch nichts geplant; es wird jedoch darauf verwiesen, dass die Software mit einigen Einschränkungen auch unter Wine eingesetzt werden kann (eine Anleitung dafür findet sich u. a. hier).

Die meisten Leser wird diese Einschränkung nicht weiter stören, da sie auf ihren PCs das entsprechende Microsoft-Betriebssystem im Einsatz haben. Und für diese ist *MyPhoneExplorer* sicher eine gute Wahl. Sehr bequem in der Handhabung, deckt er vieles ab: Verwaltung der Kontakte, Telefonlisten, Kalender und auch SMS sind möglich, wobei sich auch Anrufe initiieren und SMS lesen/schreiben lassen. Dateien und Medien (Fotos, Videos) lassen sich natürlich ebenso verwalten.

Neben der „Verwaltung“ ist sicher noch ein weiterer Bereich interessant: Der Datenabgleich, auch Synchronisation genannt. Und in diesem Umfeld bietet *MyPhoneExplorer* weitreichende Unterstützung: Ein Abgleich ist mit Outlook, Thunderbird, Sunbird, Lotus Notes, Tobit David, Windows Kontakte, Windows Kalender, und weiteren Kandidaten möglich.

Sowohl die App selbst, als auch die zugehörige PC-Anwendung sind kostenlos erhältlich: Erste ganz normal im Playstore, und letztere auf der Homepage des Anbieters.



Wer sich nicht gern auf ein Desktop-Betriebssystem einschließlich zugehöriger PC-Applikation festlegen lässt, es aber dennoch grafisch nett haben möchte, der sollte einen Blick auf 3CX DroidDesktop (ehemals *Remote Web Desktop*) werfen. Nomen est omen: Man hat bei dieser App tatsächlich den Eindruck, einen vollständigen Desktop vor sich zu haben.

Und das ganz einfach im Web-Browser, ohne Bedarf an zusätzlicher Software! Auch ein Datenkabel ist recht überflüssig, da alles über WLAN ablaufen kann.

Die Dateiverwaltung ist wahlweise über den integrierten Dateimanager (im Browser) – oder aber auch mit beliebigem Client unter Nutzung des ebenfalls integrierten FTP-Servers möglich. Noch eins oben drauf gesetzt: Auch ein VNC Server und Client sind mit an Bord. Ein WiFi-Keyboard registriert sich auf dem Androiden als Eingabe-Methode, und lässt sich so als alternative Tastatur verwenden.

Des Weiteren bietet die App die Möglichkeit, vom PC aus Kurznachrichten zu lesen/schreiben, die Kontakte zu verwalten, Screenshots zu erstellen, die WebCam zu nutzen, Apps zu sichern, und vieles mehr. Sogar ein persönlicher Webserver lässt sich damit auf dem Androiden realisieren – oder der Zugriff für einen irgendwo in der Ferne sitzenden Spezialisten via Netzwerk-Brücke umsetzen, während man gerade im mobilen Netz unterwegs ist. Beinahe vergessen hätte ich jetzt die Möglichkeit, dass sich mit Hilfe von *Remote Web Desktop* PC und Androide eine gemeinsame Zwischenablage teilen können …

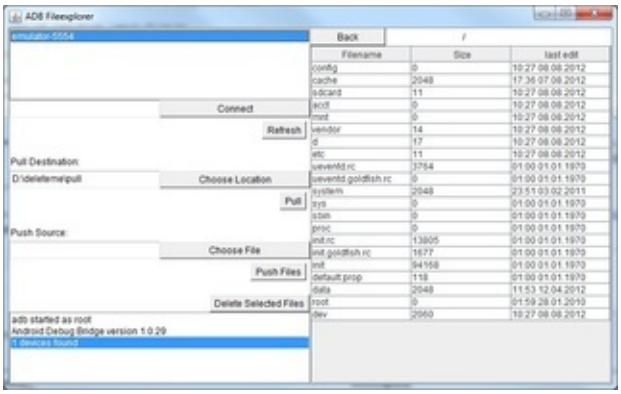
Auch der PAW Server funktioniert unabhängig von einem speziellen PC, und wird einfach im Browser benutzt. Der große Vorteil dieser Tatsache: Man kann ihn überall verwenden – beispielsweise, wenn man bei Freunden/Verwandten zu Besuch ist. Einzig ein funktionierendes WLAN und ein ebenfalls darin eingebuchter Rechner mit Web-Browser werden benötigt. Und da es wohl kaum einen aktuellen PC ohne Webbrowser drauf gibt, wäre eigentlich ausschließlich WLAN als Voraussetzung zu nennen. Im heimischen Netzwerk eingebucht, und auf dem Router eine entsprechende Port-Freigabe eingerichtet, kann auf diese Weise gar ein Experte aus der Ferne hilfreich unter die Arme greifen – vorausgesetzt, ihm wurden die IP-Adresse und die Zugangsdaten zum PAW Server mitgeteilt.

Wie auch bei den zuvor genannten Kandidaten lassen sich hier Anruflisten, SMS, Kontakte etc. einsehen, Anrufe initialisieren, SMS schreiben, etc. Und wenn der Hund sich „den Knochen“ geschnappt und verschleppt hat, selbiger per Knopfdruck zum Klingeln bringen (der „Knochen“, nicht der Hund!) um festzustellen, wo er denn nun abgeblieben ist. Vorausgesetzt, der Hund hat dabei nicht das WLAN-Signal verloren (oder war’s der Knochen?).

Natürlich ist auch ein Dateimanager enthalten. Fotos lassen sich ebenfalls durchstöbern (auf Wunsch sogar eines davon als neues Hintergrundbild festlegen), der Androide als Diktier- oder Vorlesegerät oder auch Musik-Player oder WebCam nutzen, und vieles mehr.

Für den „Power-User“

Die letzten Absätze müsste ich an dieser Stelle nun wiederholen – denn der *PAW Server* ist definitiv etwas für „Power-User“. Besonders interessant für Tüftler: Der integrierte Web-Server kann mit eigenen Skripten ergänzt werden. Dazu bietet sich die *PAW* eigene Skriptsprache an, aber auch ein PHP-Plugin ist verfügbar. Des Weiteren unterstützt *PAW* auch die im Kapitel Automatisierung genannten Tools *Locale* und Tasker.



Kommen wir zu einem kleinen Schmankerl, welches besonders Android-Entwicklern gefallen dürfte. Sie sind zwar überwiegend im Umgang mit der ADB gewöhnt, und setzen sicher so manchen Befehl darüber ab – insbesondere, um Dateien mit `adb push` auf das Gerät, bzw. mit `adb pull` vom Gerät herunter zu kopieren. Wer vorwiegend an der Shell unterwegs ist, ist damit sicher durchaus zufrieden – doch so manch einer hat sich dafür vielleicht schon eine kleine grafische Oberfläche gewünscht.

Diesem Wunsch ist XDA Forums-Mitglied DareTOBe nachgekommen, und hat den *ADB FileExplorer* bereitgestellt (siehe XDA-Developers Forum). In Java geschrieben, lässt sich dieses Programm unter nahezu jedem Betriebssystem einsetzen. Es ermöglicht das Kopieren von Dateien zwischen Gerät und Arbeitsrechner, das Löschen von Dateien sowie das Browsen durch das Dateisystem auf dem Androiden – und auch das Aufbauen einer ADB Verbindung über TCP/IP.

Darf es ein wenig mehr sein? Dann wäre vielleicht ADBBrowser einen Blick Wert. Verfügbar für Windows und Linux (32-Bit sowie 64-Bit), bietet dieser einen Datei-Browser – aber ebenso auch einen App-Browser. Reicht nicht? Na gut, einen noch: QtADB (Bild unten). Verfügbar für Windows, Mac und auch Linux (sowohl 32 als auch 64-Bit), ist dieser mit tollen Features nur so vollgestopft: Dateimanager, AppManager, Geräte-Informationen, Verwaltung der SMS, Shell-Zugriff, Screenshots, Logcat, Backup (Nandroid – aber auch einzelne Apps mit und ohne Daten). Und wem es dann wirklich reicht, der findet hier auch eine Möglichkeit zum Reboot. Das echte Schmankerl bleibt aber wohl noch ein Weilchen den Windows-Nutzern vorbehalten: DroidExplorer …


QtADB
Connection
Tools
Help
Files
Apps
Phone info
Screenshot
Settings
Shell
Applications
App name
Version
Size
3b Gallery
2.0
688.41 KiB
adbWireless
1.4.1
132.01 KiB
Angry Birds
AppControl
ASTRO
aTrackDog
AudioManager
2.1.5
372.14 KiB
Bankomaty
1.0
1.91 MiB
Barcode Scanner
3.5
456.83 KiB
BatteryLife
1.3.3
78.66 KiB
Bebbled
2.14
2.33 MiB
Bluetooth File Transfer
3.80
1.38 MiB
Backup
Uninstall
Copy to PC
Get apps info
Get app version from internet
Application
Data
Application and data
1/33 selected
Application name
Angry Birds
Application version
1.3.5
Package name
Location
internal
Latest version
Uninstall
Backup
Refresh
Open in Market

QtADB

Netzwerk

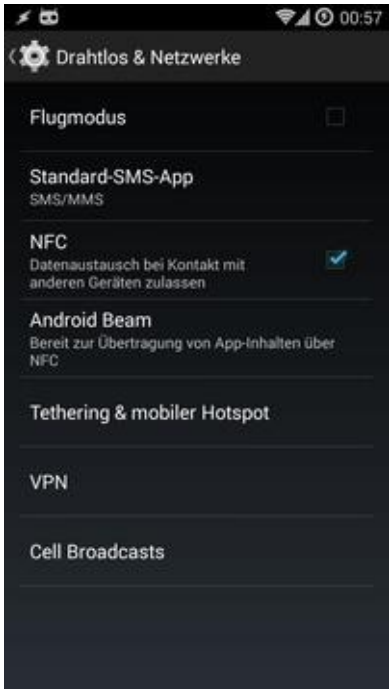
Der dritte Teil des Buches beschäftigt sich intensiver mit Netzwerk-spezifischen Dingen.

Netzwerk-Konfiguration

Ganz allgemein sind einige der hier relevanten Themen ja bereits im zweiten Teil dieses Buches (Konfiguration) aufgeführt worden – ohne jedoch auf die Details näher einzugehen. Letzteres soll nun an dieser Stelle nachgeholt werden. Soweit möglich, sollen dabei Bordmittel zum Einsatz kommen. Wo es sich anbietet (etwa, weil es kein Bordmittel gibt, oder sich etwas auf andere Weise viel bequemer erledigen lässt), kommen jedoch auch spezielle Apps zur Sprache.

An und Aus

Nein, die Rede ist hier nicht vom „großen roten Knopf“. Und auch nicht von dem speziellen Button, der als „Shutdown the Internet“ bekannt ist. Ich rede hier vielmehr davon, dass man nicht alle Netz-Dienste ständig benötigt – manchmal sind sie sogar explizit unerwünscht, etwa weil sie wie das Roaming im Ausland zusätzliche Kosten verursachen, oder bei gerade nicht verfügbarer Lademöglichkeit zu sehr am Akku nuckeln. Speziell haben daher vier Dinge unsere spezielle Aufmerksamkeit: WLAN, mobiles Netz (ganz allgemein, und Roaming im Speziellen), Bluetooth, und – nicht Lachen – das Telefonnetz.

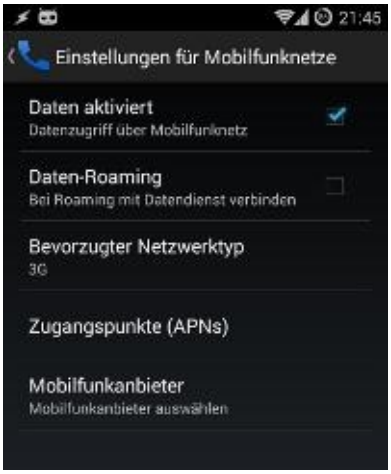


Wie war das jetzt? Frage an Radio Eriwan: „Warum sollte jemand das Telefonnetz abschalten wollen? Das ist doch bei einem Telefon die zentrale Eigenschaft!“ – „Im Prinzip schon, aber …“ War da was? Eingehende Telefonate im Ausland? Ein eBook im Flugzeug lesen? Ungestörte Nachtruhe bei Verfügbarkeit des morgendlichen Weckers? Also gut: Es gibt auch für die Deaktivierung der telefonisch-kommunikativen Eigenschaften des Schlaufons offensichtlich hin und wieder vernünftige Gründe. Schauen wir also einmal nach, wo sich bei Verwendung von Bordmitteln die nötigen Schalterchen versteckt haben. Dafür habe ich noch einmal den Screenshot eingebunden, der sich bereits im Kapitel Drahtlos und Netzwerke findet.

In diesem Menü, welches sich (wie beschrieben) vom androidischen Home-Screen über *Menü › Einstellungen › Drahtlos › Netzwerke* erreichen lässt, finden wir gleich mehrere Kandidaten. Ganz zu Oberst wäre da der *Flugmodus*. Bei diesem handelt es sich um einen „Sammelschalter“, der gleich mehrere Komponenten lahmlegt: Telefon und WLAN im Wesentlichen. Wobei sich das WLAN im Flugmodus meist wieder separat aktivieren lässt. Der Flugmodus versetzt den Androiden in einen sehr stromsparenden Modus: Bei abgeschaltetem Bildschirm wird fast kein Strom mehr verbraucht. Nur ganz abschalten wäre da noch sparsamer. Daher bietet sich dies für eine ungestörte

Nachtruhe an, aus welcher der androidische Wecker den Schlummernden allerdings morgens wieder zurückholen soll.

Ein weiterer Kandidat wäre König Blauzahn (eigener Punkt im Hauptmenü). Wer kein passendes Headset sein eigen nennt, und auch sonst keine Bluetooth-Geräte einsetzt, kann den Haken an dieser Stelle gleich permanent entfernen.



Hinter dem Punkt „Mobilfunknetze“ verbirgt sich nicht nur der globale „Knopf“ für das mobile Datennetz – es finden sich auch noch weitere Schalter. So lassen sich u. a. auch Einstellungen für das Daten-Roaming vornehmen: Soll es verwendet werden, oder besser nicht? Die meisten von uns werden es sicher eher abschalten, und nur im Ausnahmefall aktivieren. Bei Geschäftsreisenden mag das naturgemäß etwas anders aussehen. Aktiviertes Roaming erkennt man im Ausland übrigens an einem „R“ in der Notification-Bar. Der Punkt für das „Nationale Roaming“ dürfte hingegen (sofern überhaupt vorhanden) nur für die Wenigsten interessant sein, da die Netzbetreiber hierzulande so etwas nicht anbieten.

Da gerade das Thema „Roaming“ fällt: Für die Telefonie ist der Haken, dass es da keinen „Haken“ gibt. Dennoch existieren natürlich Möglichkeiten, sich auch um dieses zu kümmern: Entweder indem man im Ausland einfach den Flugmodus aktiviert, und bei Bedarf WLAN anschaltet – oder *vor* der Abreise (also noch zu Hause) die automatische Betreiberwahl deaktiviert, und stattdessen den eigenen Anbieter fest einstellt. Dies geht ebenfalls im zuletzt genannten Menü, unter dem Punkt „Netzbetreiber“ bzw. „Mobilfunkanbieter“. Auch die *Zugangspunkte* (APNs) haben gewissermaßen einen „An-und-Aus“ Effekt: Falsch eingestellt, bleibt das Datennetz nämlich üblicherweise aus. Anpassungen hier sind jedoch selten nötig, da in den meisten Fällen die automatisch ermittelten Einstellungen passen.

Eine andere Art von „Netzwerk“ stellt das GPS dar, welches bereits unter Standort und Sicherheit beschrieben wurde – und daher hier lediglich der Vollständigkeit halber noch einmal erwähnt werden soll.

Wer sich nun nicht ständig durch die vielen Menüs der „Bordmittel“ hangeln möchte, um „eben einmal schnell“ etwas an- oder auszuschalten: Natürlich gibt es dafür bequemere Wege. Eine Anlaufstelle sind die mit Android 4.2 eingeführten „Quick Settings“ im Benachrichtigungsbereich, die auf langes und kurzes Drücken unterschiedlich reagieren (Umschalten/Konfigurationsseite aufrufen). Weitere Kandidaten sind im Abschnitt Apps mit Schnellzugriffen enthalten – und vor allem gibt es auch Schnellumschalter, sofern es nur um ein „Umschalten“ (an/aus) geht.

Datensynchronisation im Hintergrund

Ein Smartphone hat man schließlich nicht zuletzt, um auch unterwegs immer auf dem aktuellen Stand zu sein – viele Apps synchronisieren dafür Daten im Hintergrund. Das ist zwar einerseits recht nützlich, aber auch nicht unbedingt immer erwünscht. Für einige Apps lässt sich diese Synchronisation daher im Menü Konten & Synchronisierung abschalten (per Default ist sie für jede App angeschaltet). An dieser Stelle verewigen sich vor allem die Google-Apps wie *Google Mail* oder der Kalender, aber auch die Hersteller-Apps wie *HTC Sync* (ebenso deren Nachrichten und Wetter App sowie die dusseligen Aktien, die kaum jemand braucht) oder Motorolas *MotoBlur*.

Doch nicht alles, was im Hintergrund Daten austauscht, ist an dieser Stelle auch aufgeführt. Ein klassisches Beispiel sind die meisten RSS-Reader. Wer also Einfluss auf den gesamten „Hintergrund-Verkehr“ nehmen möchte, sollte die Einstellungen der verbleibenden Apps separat durchgehen: Bei vielen lässt sich die Häufigkeit der Aktualisierungen einstellen; bei manchen darüber hinaus auch auf ein verfügbares WLAN einschränken. Weitere Möglichkeiten haben wir uns ja bereits beim Thema Datenverbrauch angesehen.

Mehr zum Thema „Datensynchronisation im Hintergrund“ findet sich ebenso im Bereich Tuning unter Hintergrunddaten und Synchronisierung.

WLAN Konfiguration

Besonders viel lässt sich mit den unter Drahtlos und Netzwerke verfügbaren Menü der Bordmittel nicht einstellen. Allenfalls die „Basics“ sind hier verfügbar: WLAN an aktivieren? Bei verfügbaren offene WLANs benachrichtigen? Dazu bei aktiviertem WLAN noch eine Liste der in Reichweite befindlichen Netze mit der Möglichkeit, sich zu verbinden, und allenfalls noch den ggf. notwendigen Netzwerkschlüssel einzugeben. Unter „Netzwerke verwalten“ lassen sich auch gerade einmal nicht mehr benötigte, aber konfigurierte Netzwerke (also die, zu denen man bereits einmal verbunden war) entfernen. Mehr ist auf Anhieb nicht ersichtlich. Haben wir da etwas übersehen?

In der Tat. Da war doch am Androiden noch so eine Menü-Taste? Richtig: Dort findet sich ein Punkt „Erweitert“. Und was sich dahinter verbirgt, zeigt der zugehörige Screenshot. In der „Rechtlichen Domain“ (hier: „Ländercode“) gibt man an, wie viele Kanäle benutzt werden dürfen – dies unterscheidet sich nach den lokalen Gesetzen, in Deutschland sind es 13 Kanäle. Interessant ist ferner die „WLAN Standby-Richtlinie“ (oft auch mit „WLAN im Ruhemodus“ o. ä. beschriftet), mit der man festlegt, wann das aktivierte WLAN temporär deaktiviert werden soll. Zur Auswahl

stehen hier je nach Gerät/ROM Dinge wie „bei Display-Aus“, „nie wenn im Netzbetrieb“, oder „niemals“. Wer vermeiden möchte, dass das Internet-Radio im Hintergrund ausgeht, findet hier den richtigen Schalter.

Letztendlich können auch IP-bezogene Einstellungen getroffen werden. Voreingestellt ist hier die Verwendung von DHCP, was in den meisten Fällen auch genau das richtige ist. In manchen Umgebungen ist jedoch eine feste IP notwendig, sowie die damit verbundene manuelle Konfiguration von Dingen wie „Default-Gateway“ und „DNS Server“ (wer mit diesen Begriffen nichts anfangen kann, braucht das i. d. R. auch nicht). Diese Dinge stellt man jedoch pro Zugangspunkt ein – indem man in der Netzwerk-Liste lange auf den entsprechenden Namen drückt, und dort „Netzwerk ändern“ auswählt. Jetzt noch am Ende des sich öffnenden Formulars die Box für „Erweiterte Optionen“ aktiviert, und die genannten Punkte stehen zur Verfügung.

Wer spezifischere Anforderungen hat und daher Apps für weitere Einstellungen in diesem Bereich sucht, mag sich vielleicht für diese Übersicht interessieren.

Administration

Genau genommen, gehört die Konfiguration natürlich auch zu den Aufgaben eines Administrators – doch meist wird erst nach ihm geschrien, wenn es irgendwo klemmt. Genau da soll dieses Kapitel ansetzen: Was tun, wenn etwas nicht tut? Welche Hilfsmittel gibt es, um die Ursache(n) eines Netzwerk-Problems zu finden?

Geräte-Informationen abfragen

Zuallererst wären da die Geräte-Informationen, von denen sich einige mit Bordmitteln auslesen lassen – an gefühlten zehn verschiedenen Stellen verteilt: Zunächst unter *Einstellungen › Telefoninfo › Status* MAC-Adressen, Mobilnetz-Typ und Signalstärke. Dann unter *Einstellungen › Drahtlos › Netzwerke* das verbundene WLAN-Netzwerk und ggf. auch IP-Adressen. Weitere Details lassen sich dem Gerät eventuell noch mit diversen „Geheimen“ Zugriffs-Codes entlocken (die so geheim nicht sein können, wie das verlinkte Kapitel zeigt). Und bevor die letzten Notizen auf dem Zettel angekommen sind, hat der Nachwuchs selbigen bereits zum Malen mit den Buntstiften entführt …

Zum Glück gibt es auch in diesem Bereich wieder eine ganze Reihe Helferlein, von denen ich zwei kurz vorstellen möchte. Das Meiste des soeben aufgeführten fasst beispielsweise Network Info for Android zusammen: MAC-Adresse, Signalstärke, verbundenes Netz, zugehörige IP-Adressen, und sogar die Signalstärke für das WLAN, ebenso entsprechende Details für das mobile Datennetz. Bluetooth-Details sind ebenfalls verfügbar, auch wenn die Informationen hier spärlicher ausfallen.

Ein paar zusätzliche Informationen bietet Network Info II, welches jedoch leider seit 2013 nicht mehr aktualisiert wurde. Oh – wenn das jetzt nicht die Untertreibung des Tages war, dann weiß ich auch nicht! Nicht nur sind hier WLAN und die mobile Verbindung voll abgedeckt – der Entwickler hat auch daran gedacht, dass Bluetooth ebenfalls ein Netzwerk-Interface ist, dass es IPv6 gibt, und das mobile Netz darüber hinaus auch Informationen über Mobilfunkzellen bereitstellen kann. Das alles ist sauber auf verschiedene Reiter aufgeteilt, was einen gezielten Zugriff auf die Informationen möglich macht. Wer die gesammelten Informationen gern in Textform exportieren möchte, erhält dazu natürlich die Möglichkeit – an alles ist gedacht, und ich habe mit dieser Beschreibung wohl nur an der Oberfläche gekratzt … Kurz: Das Tool der Wahl, wie es scheint, um sich einen umfassenden Überblick über alle relevanten Informationen zu verschaffen.

Derart gewappnet, kann man sodann weiterschreiten zum nächsten Punkt – sofern die gewonnenen Erkenntnisse das Problem nicht bereits gelöst haben:

Diagnose: Liegt was schief?

In welcher Richtung an dieser Stelle geforscht werden muss, hängt natürlich ganz vom Problem ab: Ist es einfach nur langsam, oder kann gar keine Gegenseite erreicht werden (obwohl eine Netzwerk-Verbindung besteht)? Noch anders sieht es aus, wenn gar keine Netzwerkverbindung aufgebaut werden kann. Aber eines nach dem anderen.

Keine Netzwerk-Verbindung möglich

In diesem Fall greifen wir zunächst auf die Geräte-Informationen zurück – und schauen erst einmal nach, ob überhaupt eine Gegenseite gefunden wurde. Für WLAN eignen sich die Bordmittel dafür recht gut: Unter *Einstellungen › [Drahtlos & Netzwerke ›] WLAN* sollten im Abschnitt *WLAN-Netzwerke* die verfügbaren (d. h. in Reichweite befindlichen) WLAN-Netze aufgelistet sein. Zusammen mit Signalstärke und ggf. Verschlüsselungstyp.

Bleibt diese Liste leer, kann es dafür mehrere Gründe geben:

- Es sind schlicht und ergreifend keine WLAN-Netze in Reichweite
- WLAN ist nicht aktiviert (das Häkchen weiter oben auf der Seite – oder auch der Netzschalter des heimischen Routers)
- Das Netzwerk-Interface ist defekt (entweder im Androiden, oder auch im Router)

Der erste Punkt lässt sich am Besten mit einem „Zweitgerät“ verifizieren: Zeigt dieses verfügbare Netze an, muss die Ursache eine andere sein. Auch das Häkchen auf der Einstellungsseite des Androiden ist schnell geprüft, ebenso der Netzschalter des Routers. Dann bleibt nach dem Ausschlussverfahren noch der letzte Punkt übrig.

Und der Fall, in dem verfügbare Netze angezeigt werden – jedoch der Androide sich nicht zu diesen verbinden möchte. Auch das kann sich auf verschiedene Weise äußern:

- Es sieht zunächst danach aus, als würde die Verbindung aufgebaut – doch plötzlich stehen alle Netze auf „außer Reichweite“

- Eine Fehlermeldung zeigt den fehlgeschlagenen Verbindungsaufbau an

Der erste Fall ist aus dem Leben gegriffen (trat bei mir beispielsweise häufiger nach Installation eines (inoffiziellen) Custom-ROMs von CyanogenMod auf dem *HTC Wildfire* auf). In diesem Beispiel liegt das Problem häufig im Radio-Image begründet; sofern man also seinen Androiden gerootet und mit einer Custom-Firmware versehen hat, hilft meist ein erneutes Flashen des originalen Radio-Images; doch zuvor erkundigt man sich besser im Forum, da weitere Details entscheidend für den Erfolg sein können.

Der zweite Fall kann mehrere Ursachen haben. Besonders häufig sind:

- falscher Netzwerk-Schlüssel (also so etwas wie ein „falsches Passwort“)
- richtiger Netzwerk-Schlüssel, aber beim falschen Netzwerk eingetragen (läuft auf das Gleiche hinaus)
- ein „MAC-Filter“ beim Zielnetzwerk/Router („Betriebsfremden ist der Zutritt verboten!“)

Am einfachsten lässt sich der zweite Punkt ausschließen, indem man bei der Auswahl des Zielnetzes im zweiten Anlauf besonders genau aufpasst. Sowohl für den ersten als auch für den dritten Punkt muss man sich an den Admin des Routers wenden: Dieser sollte sowohl den korrekten Schlüssel benennen/mitteilen, als auch ggf. die MAC-Adresse des Androiden in die Liste zugelassener Geräte aufnehmen können.

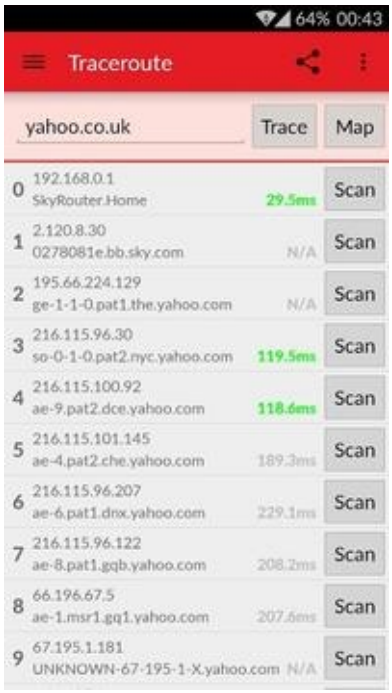
Minimal anders sieht es aus, wenn es sich um das mobile Datennetz handelt: Hier finden sich die Einstellungen unter *Einstellungen › Drahtlos & Netzwerke › Mobilnetze* – und man kann keinen falschen Schlüssel beim richtigen Netzwerk (oder umgekehrt) eingeben, da die Verifikation über die SIM-Karte erfolgt. Lässt sich kein Kontakt herstellen, hilft neben einer Überprüfung des Häkchens für die Aktivierung des mobilen Datennetzes noch die Verifikation der APN-Daten. Schafft auch das keine Lösung, bleibt der Anruf beim Anbieter der SIM-Karte übrig.

Server können nicht erreicht werden

Ist ein Server nicht zu erreichen, kann dies mehrere Ursachen haben. Möglich wären unter anderem folgende Probleme:

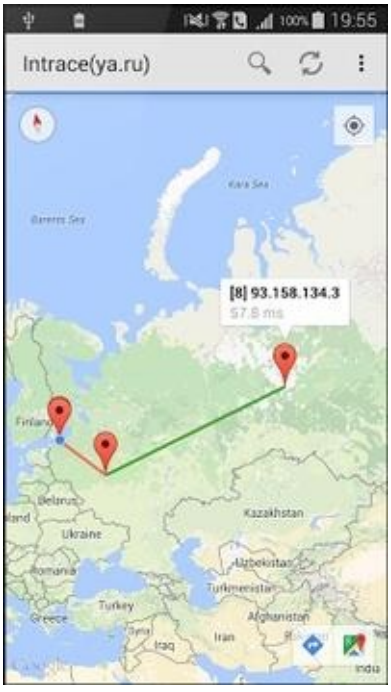
- Der Androide selbst hat keine Netzwerk-Verbindung (siehe oben)
- Der betreffende Server wurde (z. B. für Wartungsarbeiten vorübergehend) heruntergefahren oder gar permanent abgeschaltet
- Der Server an und für sich ist schon erreichbar, nur nicht der gewünschte Service (Beispiel: Beim Zugriff auf eine Webseite kommt der Fehler, der Server sei nicht verfügbar – Grund ist jedoch nicht ein abgeschalteter

Rechner, sondern lediglich die Webserver-Applikation wurde beendet)
- Der Server läuft wahrscheinlich noch – aber das Netz, in dem er steht, ist nicht erreichbar (defekter Router auf der Strecke)



Während sich der erste Punkt noch lokal klären lässt, schaut der Rest zunächst komplizierter aus: Nicht immer kennt man den Betreiber persönlich, um eben einmal kurz anzurufen und nachzufragen. Wie kommt man der Sache aber sonst auf die Spur? Einige passende Tools sind wiederum in einer Übersicht zusammengefasst, eine kleine Auswahl davon kommt hier zur Sprache.

So lässt sich der Weg zum Zielrechner beispielsweise mit *Traceroute* nachverfolgen – einem Feature, das u. a. PortDroid bietet. Endet die angezeigte Liste mit der IP-Adresse des gewünschten Ziels, so ist der Server selbst am Netz – und vermutlich der Service (also Web-App, Mail, oder was immer man benutzen wollte) das Problem. Nicht selten jedoch bleibt der Zug auf der Strecke stehen: Plötzlich dauert es länger, bis die nächste Zeile erscheint, und/oder es kommen nur noch Sternchen. In dem Fall ist das Routing (und wahrscheinlich in irgend einem Rechenzentrum ein Router) defekt; da gibt es nicht viel, was man tun kann. Außer die „defekte Stelle“ liegt noch im Bereich des eigenen Netzanbieters: Dann kann man diesen mit den ermittelten Details kontaktieren und um Hilfe bitten.



Zugegeben: Mit derartigen Zahlen- und Zeichenketten ist der Laie nicht selten überfordert. Es fällt einfach schwer, sich darunter etwas vorzustellen. Sicher lässt sich die Ausgabe-Syntax erklären: Die erste Zahl ist einfach ein Zähler (der wievielte Schritt auf dem Weg ist es?). Hinter diesem werden IP-Adresse und, sofern verfügbar, der zugehörige Rechnername angezeigt. Den Abschluss bietet schließlich die Angabe, wie schnell die Reaktionszeit war.

Weitaus anschaulicher stellt Visual Traceroute diese Informationen dar: Darunter kann sich auch ein Laie etwas vorstellen! Die Route wird bei dieser App auf einer Karte angezeigt – allerdings eher grob, indem die Standorte der Stationen mit direkten Linien, die nicht unbedingt der Kabelführung entsprechen, verbunden werden. Dazu gibt es die gleichen, zuvor genannten Details: Die Schrittnummer (in eckigen Klammern), daneben die zugehörige IP-Adresse und die Zeit, welche bis zum Eintreffen der Antwort von diesem Punkt verging (Round Trip Time).

Meist nutzen derartige Apps Maxminds GeoIP Datenbank um festzustellen, wo

sich die entsprechenden IP-Adressen befinden. So können sie die Standorte auf der Karte anzeigen, und auch in der Liste benennen. Ähnliches leistet übrigens auch eine App namens Visual Tracert. Diese kann die angezeigten Routen sogar als KML-Datei für Google Maps und Google Earth exportieren.

Jetzt wissen wir also, dass der Server als solches erreichbar ist. Von einem anderen Rechner aus ließe sich noch feststellen, ob jemand (vielleicht ein Freund, der gerade am PC sitzt) auch auf den gewünschten Service zugreifen kann. Warum können wir das dann nicht?

Antwort auf diese Frage könnte u. a. MobiPerf geben. Neben umfangreichen Informationen zur Konfiguration unseres Androiden und seines Netzwerks kann diese App nämlich auch feststellen, welche Dienste der Netzwerk-Anbieter ggf. gesperrt hat. So zeigt das Beispiel des Screenshots (aus einer älteren Version), dass der SMTP-Port 25 nicht genutzt werden kann – hier ist es also auf normale Weise nicht möglich, eine Mail zu verschicken.

Mit dieser App lassen sich beispielsweise auch die Latenzzeiten für DNS-Lookup/Ping/Verbindungsaufbau, die Bandbreite für Up-/Downloads, Signalstärke und mehr ermitteln. Alle Testergebnisse werden dabei im Cache aufbewahrt. Damit lassen sie sich einerseits auch offline auslesen, anderseits aber ebenso mit späteren/früheren Ergebnissen (oder solchen unterschiedlicher Standorte) vergleichen. Zahlreiche Graphen ermöglichen auch eine „visuelle Analyse“.

Träges Netz

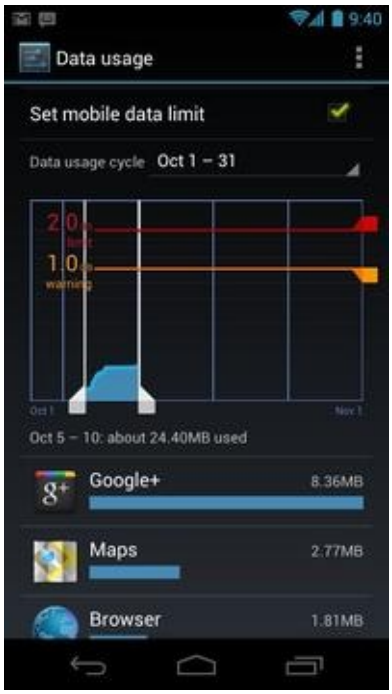
Sind alle netzwerk-abhängigen Apps durch die Bank grottig langsam, liegt dies sicher weniger an allen zuständigen Servern – sondern vermutlich eher an der eigenen Netzverbindung. Beziehungsweise an der Netzversorgung des Anbieters. Doch wie lässt sich so etwas belegen?

Eine mögliche Antwort nennt sich Speedtest.net Mobile. Die App eignet sich, um Schwachstellen beim Provider aufzudecken: Die Resultate periodischer Tests werden gespeichert, sodass man im Nachhinein eine Vergleichsmöglichkeit hat. Auf diese Weise ist leicht erkennbar, wo die lahme Ente sitzt – und wo hingegen „Schmitts Katze“ abgeht.

Wird hingegen ein echter Benchmark-Test gewünscht,

greift man beispielsweise zu 4Gmark SpeedTest. Mit dieser lassen sich Messreihen erstellen, die man auch mit der durchnittlichen Bitrate anderer User in der Gegend vergleichen kann. Allerdings erfolgt die Messung hier nicht „nebenbei“, sondern es wird separater Datentraffic verursacht: Wie viel das pro Messung ist, wird nicht angegeben; bei vergleichbaren Apps sind dies jedoch durchaus schon einmal 7 MB.

Wer verbrät mein Datenvolumen?



Seit Android 4.0 aka Ice-Cream-Sandwich, lässt sich diese Frage recht einfach beantworten – denn hier ist die entsprechende Funktionalität bereits von Haus aus im System integriert: Eine Überwachung des Datenverbrauchs bis auf App-Ebene herab (sogar getrennt nach Verbrauch bei Vorder- und Hintergrundaktivität), mit Statistik-Graph, und einschließlich der Möglichkeit zum Festlegen eines Warn- und eines „harten“ Limits. Natürlich getrennt nach WLAN und mobiler Datenverbindung, wobei man ersteres zunächst explizit aktivieren muss.

Doch auch für den kleinen Teil der Android-Nutzer, die noch nicht in den Genuss dieser Version gekommen sind, sind Hopfen und Malz noch lange nicht verloren – denn für diese gibt es Apps wie beispielsweise 3G Watchdog. Diese App überwacht die mobile Datenverbindung (3G/Edge/GPRS) und deren Trafficverbrauch, zeigt Benachrichtigungssymbol (Grün, Orange, Rot) in der Statuszeile und gibt eine detaillierte Übersicht zum Verbrauch. Zwei Widgets stehen auch zur Wahl.

Aber *3G Watchdog* kann noch mehr. Sicher: Es warnt vor und bei Erreichen der konfigurierten Limits – was für sich genommen schon eine gute Sache ist. So richtig

interessant wird es, wenn außerdem die App APNdroid installiert ist: Kurz vor Erreichen des eingestellten Limits dreht *3G Watchdog* dann nämlich einfach den Hahn zu! Der Zugangspunkt (in der Android-Konfiguration) wird dazu von *APNdroid* so verändert, dass er nicht mehr funktioniert. Und bevor jetzt Panik ausbricht: Selbstverständlich lässt sich diese Änderung rückgängig machen.

Sichere Übertragung

Für harmloses Surfen, ein paar News Lesen, und ähnliche Dinge – da muss man sich über sichere Übertragungswege sicher nicht den Kopf zerbrechen. Anders sieht es aus, wenn sensible Daten ins Spiel kommen: Betriebsgeheimnisse, Finanzdaten, neueste Entwicklungen, oder auch nur die privaten Zugangsdaten zu irgendwas. Beim Gedanken, dass da jemand mitlesen könnte, macht sich beim Anwender ein mulmiges Gefühl breit – während dem Vorstand die Haare zu Berge stehen, als hätte der gerade einen Zehnerpack Koppelzaun in der Hose.

Spätestens bei bildlicher Vorstellung dieses Sachverhaltes wird klar: Dafür muss eine Lösung her. Lässt sich Android tauglich machen für „corporate use“?

Proxy, Tunnel & Co

Manchmal scheint es, als wäre „Gefühl sein alles“, und Namen nur „Schall und Rauch“ – wie Doktor Faust sich so trefflich auszudrücken wusste (Faust I, Marthens Garten). Ist ein Proxy nun eine Art spezieller Tunnel – oder nutzt ein Tunnel eine spezielle Art von Proxy?

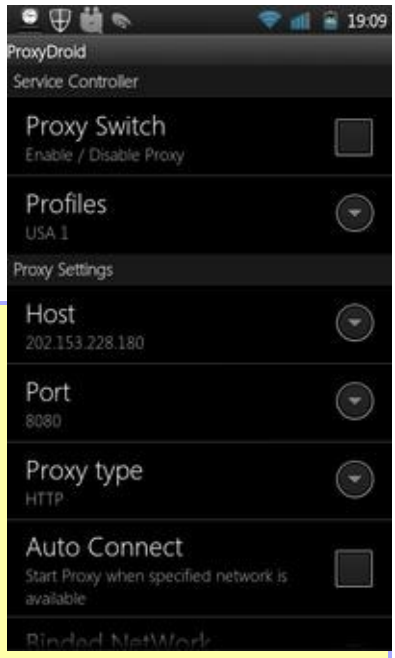
Praktisch gesehen, haben beide viel gemein – und so mancher Anwender weiß die beiden nicht recht auseinander zu halten. Manchem Entwickler geht es offenbar ähnlich. So liest man beispielsweise in der Beschreibung der App SSH Tunnel, die sich in erster Linie hinter der ganz speziellen, „chinesische Mauer“ genannten Firewall befindliche Anwender zur Überwindung derselbigen richtet, dass es als „globaler Proxy“ fungieren kann.

Gemeinsam ist allen beiden auf jeden Fall eines: Sie sind oftmals gleichermaßen geeignet, gewisse Sperren zu umgehen. So möchten viele Firmen gern vermeiden, dass ihre Angestellten sich auf dubiosen Webseiten ihre Arbeitszeit vertreiben – und haben daher die für das Web-Surfing i. d. R. zuständigen Port 80 und 443 gesperrt. Der Zugriff muss stattdessen über einen im Haus befindlichen Proxy-Server (der als Vermittler fungiert, und als einziger auf den genannten Ports „rauswählen“ darf) erfolgen. Natürlich ist dabei auch Port 22 gesperrt, damit sich niemand auf fremde SSH-Server verbindet. Und auf dem Proxy sind wiederum eine ganze Reihe interessanter Webseiten gesperrt. Meister Schlauberger umgeht dies nun, indem er sich einen Tunnel baut: Daheim lauscht ein SSH-Server auf Port 443, und dahinter ist der Weg frei. Da Port 443 vom Firmen-Proxy für sichere (und verschlüsselte) Web-Verbindungen akzeptiert wird, lässt dieser unseren Meister Schlauberger brav durch.

Und wie geht man das Ganze nun unter Android an?

Proxies

Grundlegende Proxy-Einstellungen sind unter Android vorhanden – nur nicht immer für den Anwender einfach erkennbar. Und leider wohl auch nur für das WLAN vorgesehen, wo es sich in den erweiterten Einstellungen des jeweiligen Zugangspunktes befindet (siehe WLAN-Konfiguration).



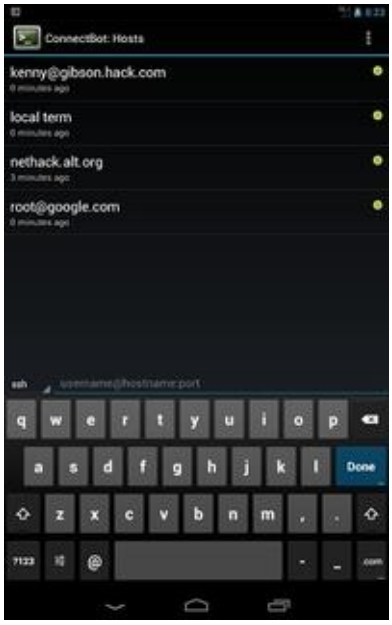
Wem diese Grundfunktionalität nicht ausreicht, der greift beispielsweise zu ProxyDroid. Diese App will helfen, mit allen Apps auf dem Androiden auch über Proxies auf das Internet zuzugreifen (benötigt dazu allerdings root). Dies ist z. B. häufig in Firmen nötig, wo eine Firewall den direkten Zugriff unterbindet. *ProxyDroid* unterstützt dabei HTTP / SOCKS4 / SOCKS5 Proxies, basic / NTLM / NTLMv2 Authentication, verschiedene Profile und mehr. Es lässt sich auswählen, welche Apps bei ihrem Zugriff auf das Internet über den Proxy geleitet werden sollen. Auch lässt sich ein konfigurierter Proxy für die Verwendung in einem bestimmten Netz (WLAN SSID / Mobiles Netzwork) einschränken, sowie bei Verfügbarkeit eines bestimmten Netzwerks automatisch aktivieren. Widgets ermöglichen ferner die schnelle (De-)Aktivierung von Proxies, und man kann sich per Klingelton und/oder Vibration sogar über Änderungen des Verbindungsstatus informieren lassen. Damit ist doch eigentlich an alles gedacht, oder?

Wenn das Wörtchen „eigentlich“ nicht wäre. Denn zumindest einen weiteren Punkt gäbe es noch: Anonymität. Ein Synonym dafür in der Netzwerk-Landschaft wäre TOR – nicht der nordische Donnergott, sondern der anonymisierende Routing-Dienst. Und damit kommt Orbot ins Spiel, der offizielle Android-Port von TOR für Android. Mit dieser App wird der Traffic anonymisiert – der Absender eines Netzwerk-Paketes ist also beim Empfänger nicht mehr erkennbar. Laut Beschreibung funktioniert das mit jeder App, die über Proxy-Einstellungen verfügt, mit Orweb: Proxy+Privacy Browser sowie mit Orbot-enabled Apps wie Gibberbot (secure chat). Transparenter Proxy-Support soll auf den meisten gerooteten Geräten verfügbar sein – wobei man einstellen kann, für welche Apps dieser dann verwendet wird.

Tunnel

Alles zappenduster? Oder was für ein Tunnel? Manchmal gibt es halt keinen

direkten Weg, und man muss Zwischenstation machen. Von A nach C geht es dann nur über B. Und das nennt sich Tunnel: „über B nach C“. Stark vereinfacht ausgedrückt, natürlich – denn häufig kommt hinzu, dass man dazu in einen speziellen „Schutzanzug“ muss. In der Regel hat das auch etwas mit Datensicherheit zu tun. und meistens auch mit SSH.



Als erste Wahl zu diesem Thema wäre sicherlich ConnectBot zu nennen. Bei der App handelt es sich um eine einfache, mächtige, open-source SSH-Client-Anwendung – die es ermöglicht, mehrere simultane SSH-Sessions zu betreiben, sichere Tunnel aufzubauen (ah!), und mehr – sogar Copy und Paste zwischen Anwendungen wird unterstützt. So kann man auch „mal eben kurz“ auf die Kommandozeile des Linux/Unix Servers zugreifen, der am anderen Ende der Welt steht.

Da sich die Einrichtung eines sicheren Tunnels ohne Vorkenntnisse nicht so ganz trivial gestaltet, und auch ganz allgemein die ersten Schritte so manchen Anwender vor das eine oder andere Problem stellen könnten, sei hier noch auf ein paar Artikel im Netz verwiesen (leider konnte ich keine aktuellen deutschsprachigen finden, daher sind die folgenden auf Englisch):

- Secure Android With SSH Tunnelling (Tutorial)
- SSH Proxy with ConnectBot and ProxyDroid (XDA)
- Rootless Android SSH Tunneling (Youtube Video, 2015; ab ca. 21:00) – Alternative im zweiten Teil ab ca. 14:25 mit SSHTunnel

ConnectBot ist schlichtweg der Android-Klassiker für SSH. Natürlich gibt es noch einige weitere Kandidaten – etwa meinen persönlichen Favoriten, JuiceSSH, der das Tunneln ebenfalls beherrscht. Die anderen finden sich wieder einmal in einer Übersicht bei *IzzyOnDroid*.

VPN

Ein VPN ist eigentlich so etwas wie ein „systemweiter Tunnel“, mit dem der Client (in unserem Fall der Androide) sich in das Netzwerk des Servers einbindet, als wäre er direkt dort vor Ort. Ich bin also zu Hause, und mein Androide ist im Office der Firma – obwohl ich ihn in der Hand halte. Und nein, meine Arme sind nicht mehrere Kilometer lang (ich kann auf Anfrage aber gern noch einmal nachmessen).

Eine Angelegenheit, die besonders im Unternehmens-Umfeld (und dort speziell für Außendienst-Mitarbeiter) interessant ist, wo man gewisse Dienste nur im

IntraNet bereit hält. Was sich aber gut auf den Privatbereich übertragen lässt: Auch im heimischen Netz gibt es einiges, was man nicht unbedingt der Öffentlichkeit zur Verfügung stellen, aber dennoch selbst von unterwegs verfügbar haben möchte. Etwa das Web-Interface des eigenen Routers, die Drucker, oder die P…hotosammlung. Welche Helferlein uns zum Thema VPN zur Verfügung stehen, zeigt diese Übersicht bei *IzzyOnDroid* auf.

Android bringt bereits von Haus aus einen VPN-Client mit, der auch mit den meisten VPN-Servern kompatibel ist. Trifft letzteres nicht zu, sind oft tiefer greifende Maßnahmen nötig – da die nötigen Tools der Drittanbieter in der Regel *root* voraussetzen. Aus diesem Grund wird der vorinstallierte Client als „erste Wahl“ angesehen: Ein super sicherer Spezial-VPN-Server erhöht die Sicherheit schließlich nicht, wenn im Gegenzug sämtliche mobilen Teilnehmer ihre Geräte rooten müssen.

Die Einrichtung eines VPN mit Android-Bordmitteln beschreibt ein sechs-seitiger Artikel bei TECChannel.DE im Detail. Und hieß es bei AVM zur Fritz!Box 2013 noch: *Für Apple iOS-Geräte ist die Nutzung per VPN aber grundsätzlich möglich. Eine VPN-Verbindung von Google Android zur FRITZ!Box wird hingegen nicht unterstützt.*, bietet die deutsche Vorzeige-Firma heute auch für Android ein Tutorial zur Einrichtung von VPN, das ganz ohne root und extra-Apps auskommt.

Sollte es mit Bordmitteln nicht hinzubekommen sein, könnte beispielsweise NeoRouter weiterhelfen: All die lästigen Voraussetzungen wie *root*, separates Kernel-Modul und aufwändige Installation scheinen bei dieser Lösung zu entfallen. Auf der Homepage werden Software-Pakete für alle möglichen Systeme (darunter Linux, Mac, BSD, Windows) angeboten, damit auch die Gegenstelle versorgt ist. Das Ganze verspricht eine einfache Umsetzung („einfacher als OpenVPN und schneller als LogMeIn Ignition“). Für den Einsatz im Unternehmensumfeld steht eine stabile, für Version-Junkies eine häufig aktualisierte Version zur Verfügung. Beide sind gratis.

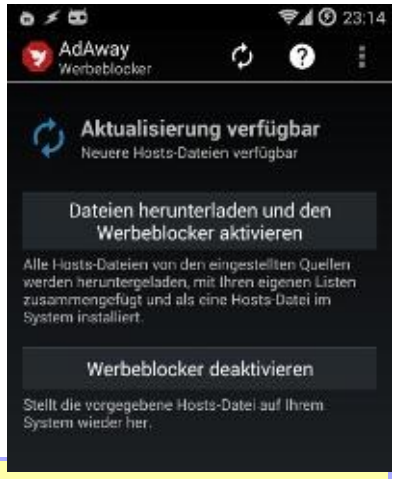
Eine Alternative für gerootete Geräte könnte auch OpenVPN sein – zumal es Open-Source, und die entsprechende App bei einigen Custom ROMs bereits vorinstalliert ist. Eine Anleitung für die Konfiguration unter Android findet sich bei Nodch.DE, für den notwendigen Server ist natürlich ebenfalls Sorge zu tragen.

AdBlock

Oh, ich höre schon das Geschrei: Die armen Programmierer, die am Hungertuch nagen müssen … Vielleicht sollte ich in meine Bücher auch ein wenig Werbung einbauen :wink:

Damit mich keiner falsch versteht: Ich möchte niemandem die Butter vom Brot nehmen (die Wurst gleich gar nicht). Eben so wenig soll dies ein Aufruf zur kompletten Reklame-Blockierung sein. Aber es gibt hin und wieder durchaus berechtigte Gründe, warum ein solcher „Ad-Blocker“ ans Werk gehen muss. Beispiele gefällig? Oh, „fällig“ ist da richtig gut: Abo-Fallen. Erwachsene sollten Dank vollständig aktualisiertem und mit allen nötigen Zusatz-Modulen versehenem GMV halbwegs darauf achten können, wo sie hintippen. Von einem nicht vollständig ausgewachsenen menschlichen Wesen (als Abkürzung für „Kann im Normalfall draufdrücken“ oftmals auch „Kind“ genannt), dem der Androide „eben einmal für ein Spiel“ überlassen wird, kann man dies hingegen nicht unbedingt erwarten.

Dann wären da ja auch noch die ganzen Web-Bugs und andere Tracker, die gern das Nutzer-Verhalten protokollieren wollen. Natürlich nur zu unserem Besten, versteht sich (gemeint ist damit jedoch, dass sie uns „zum Besten halten“ wollen). Spätestens hier fallen natürlich wieder die Worte „Datenschutz“ und „Privatsphäre“, und damit wird es Ernst.

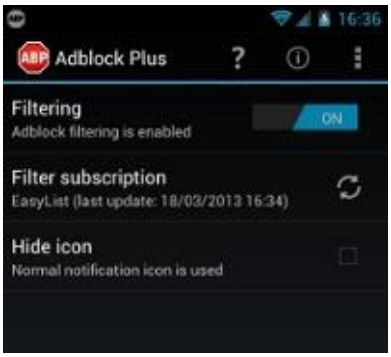


Zum Glück gibt es einige Möglichkeiten, sich zu wehren. Diese benötigen jedoch fast ausnahmslos root.

Am bekanntesten ist sicherlich AdAway. Diese App ersetzt die Hosts-Datei mit einer Version, in der die meisten bekannten Ad-Server ins Daten-Nirvana geschickt werden, indem ihnen eine „leere“ IP-Adresse zugewiesen wird. Dies blockt Werbe-Zugriffe effektiv – nicht nur im Browser, sondern in allen Apps.

(Hinweis: Zur Wiederherstellung des Ursprungs-Zustandes nicht einfach die App deinstallieren, sondern vorher mit Hilfe der App die ursprüngliche Host-Datei wieder herstellen („Werbeblocker deaktivieren“). *AdAway* macht ja nichts weiter, als diese Datei einmalig (oder einmal täglich) anzulegen – die Filterung der entsprechenden Netzzugriffe erledigt das Android-System anschließend selbst!)

Neben den bereitgestellten Hosts-Listen erlaubt die App auch, eigene Black- bzw. White-Listen sowie Umleitungen zu konfigurieren, weitere Quellen hinzuzufügen, und mehr. Ebenfalls mit an Bord ist ein Adware-Scanner.



Ohne root geht es zumindest im WLAN mit AdBlock Plus. Diese bereits vom gleichnamigen Firefox-AddOn bekannte App arbeitet nach einem bereits weiter vorn erklärten Prinzip: Sie wird als Proxy zwischengeschaltet – und muss dem System daher als solcher bekannt gemacht werden. Wie letzteres funktioniert, wurde ja bereits beschrieben. Übrigens benutzt *AdBlock Plus* die gleichen Filterlisten, die auch vom Firefox-Pendant bekannt sind.

Ebenfalls vom „normalen Computer“ bekannt ist eine weitere Filter-Proxy-App: Privoxy beschreibt sogar einen Weg, wie es als Proxy im mobilen Datennetz einzubinden sei.

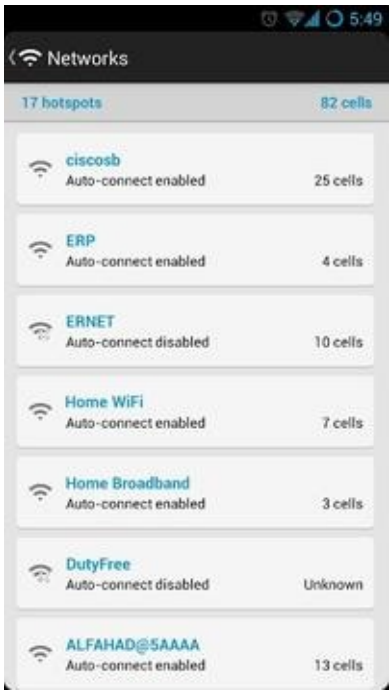
Eine weitere Variante wurde bereits beim Thema Firewalls genannt: Mit einer solchen ließe sich der Netzzugang einzelner Apps unterbinden. Eine komplexe Firewall könnte auch für bestimmte Apps lediglich bestimmte Ziele sperren. Dies würde jedoch auch eine komplexe Benutzer-Oberfläche erfordern, weshalb sich wohl noch niemand dieser Aufgabe gewidmet hat. Technisch versierte Anwender greifen daher bei Bedarf zu einer Terminal-App, und verwenden das *iptables* Binary direkt. Das setzt natürlich genaue Kenntnis der Syntax voraus – weshalb man sich zunächst zumindest mit den Grundlagen beschäftigt haben sollte.

WLAN

Netzwerk und Android – da ist WLAN sicher ein zentrales Thema. Und so soll sich dieses Kapitel einigen WLAN-spezifischen Dingen widmen.

Automatische (De-)Aktivierung

Mit diesem Thema greifen wir dem vierten Teil des Buches (Tuning) bereits ein wenig vor: Da WLAN schon im Stand-By Betrieb einen nicht unbedeutenden Stromverbrauch aufweist (etwa das zwanzigfache des GPS-Standby – was aber nur ca. 3% des Verbrauches eines auf niedrigste Stufe eingestellten Displays ausmacht), ist ein Abschalten bei fehlendem Bedarf sicher keine verkehrte Idee. Natürlich ließe sich das manuell erledigen; doch vergesslich, wie wir sind, überlassen wir das besser nicht dem Zufall. Eine Übersicht verfügbarer Werkzeuge für diesen Zweck findet sich natürlich wieder in einer Übersicht.



Die einfachste Variante lässt sich mit Bordmitteln regeln, und nennt sich „WLAN Standby Richtlinie“. Mittels dieser lässt sich WLAN abschalten, wenn man das Display abschaltet. Oder auch nicht. Oder auch nur dann, wenn kein Ladekabel angeschlossen ist. Für manchen Anwender mag das bereits genügen – doch hat es durchaus auch den einen oder anderen Haken: Etwa, wenn man im Hintergrund Musik von einem Server streamt. Oder periodisch Daten synchronisieren möchte (was dann über das mobile Netzwerk erfolgen würde). Für diese und ähnliche Fälle muss also eine Alternative her.

Jetzt könnte man natürlich auf eines der Multi-Talente aus dem beschriebenen Bereich der Automatisierung zurückgreifen (sowohl dort beschriebenes *Llama* als auch Tasker haben diese Rubrik in ihrem Repertoire). Sofern man jedoch außer dem WLAN nichts weiter steuern will, wäre das vielleicht ein wenig Overkill – und man greift lieber zu einem auf diesen Bereich spezialisiertes Tool.

Das könnte beispielsweise Smart WiFi Toggler heißen. Mit dieser App lässt sich festlegen, an welchen Orten man WLAN benutzen möchte: Betritt man die definierten Zonen, wird WLAN aktiviert – verlässt man sie, wird es wieder abgeschaltet; zeitgesteuertes Schalten ist ebenso möglich. Wer bereits Tasker im Einsatz hat, kann wie gesagt auf diese App natürlich verzichten – und setzt das dort mit entsprechenden Profilen um. Im Unterschied zu *Tasker* lernt *Smart WiFi Toggler* bei den Wunsch-Standorten jedoch automatisch mit. Für die Ortsbestimmung kommen Mobilfunkzellen zum Einsatz, sodass der Akku-Verbrauch auf einem Minimum gehalten wird.

Tethering

Da ist man also unterwegs, und würde gern mal schnell mit dem Tablet oder auch Notebook ins Netz der Netze. Natürlich kein WLAN weit und breit – zumindest keines, dem man traut. Da bietet sich doch förmlich das mobile Datennetz des Androiden-Telefons an, oder? Klar, mit Android 2.2 oder neuer gar kein Thema, da erstellt man sich einfach seinen eigenen Hotspot. Ist ja von Haus aus dabei.

Die zugehörigen Einstellungen finden sich unter *Einstellungen › Drahtlos & Netzwerke › Tethering & mobiler Hotspot*. Je nach Android-Version (und teilweise auch Geräte-Hersteller) heißt dieser Punkt u. U. leicht anders, und bietet auch unterschiedliche Möglichkeiten. So kann eventuell zusätzlich zum WLAN-Tethering auch noch USB-Tethering im Angebot sein.

Die Einrichtung ähnelt der eines Routers – was ganz logisch ist, denn schließlich möchte man seinen Androiden ja gerade in einen solchen verwandeln. Dazu müssen zunächst einige wenige Daten festgelegt werden, wie auf dem Screenshot erkennbar: Die zu verwendende SSID für das zu errichtende drahtlose Netzwerk, die Sicherheitsstufe („keine“/„open“ und „WPA2“ stehen dafür i. d. R. zur Auswahl, gelegentlich auch weitere), sowie ggf. ein „Passwort“ (auch als „WLAN-Schlüssel“ bekannt). Ist das geschehen, lässt sich der „mobile Hotspot“ jederzeit aktivieren und deaktivieren.

Einige WLAN-Tether-Apps bieten darüber hinaus zusätzliche Optionen, welche sich in den Bordmitteln nicht finden lassen. So verfügt etwa die App Portable WiFi Hotspot von *TNT Studio* eine Übersicht über verbundene Clients, eine andere App gleichen Namens Statistiken zum Datenverbrauch.

Fällt WLAN als Möglichkeit aus, kann man noch immer auf PdaNet (Screenshot) oder FoxFi ausweichen, um das Netz zumindest per USB oder Bluetooth zur Verfügung stellen. Die beiden Apps haben ihre Funktionalitäten verschmolzen, und sind daher weitgehend identisch. Ein WLAN-Hotspot wird von ihnen ebenfalls angeboten. Vorteil des „schurgebundenen Tethering via USB“: Bei einem Kabel ist klar erkennbar, wer dran hängt – ein „heimliches Einklinken“ wird somit deutlich erschwert.

Ist das Tethering einmal eingerichtet, könnte man es auf Knopfdruck aktivieren – wäre der Knopf nicht so versteckt, insbesondere bei den Bordmitteln. Abhilfe schaffen hier etliche Widgets für den Schnellzugriff, die teilweise das Schalten direkt übernehmen. Auch diese sind in der eingangs genannten Übersicht zu finden – zumindest eine Auswahl davon, bzw. auch in Android's hauseigenen „Quick Settings" enthalten.

Reverse Tether

Was passiert nun, wenn man das ganze umdreht? Klar: Man bekommt den Laptop-Deckel nicht mehr auf. Unsinn: Gemeint ist natürlich eine Umkehr der Verbindungsrichtung. Etwa, wenn am aktuellen Standort nur kabelgebundenes Netzwerk zur Verfügung steht. Dann ließe sich die am Laptop bestehende Internet-Verbindung auch vom Androiden aus nutzen. Per USB-Kabel ermöglicht dies beispielsweise die App Reverse Tether. Sie unterstützt eine automatische Konfiguration, lässt aber auch manuelle Anpassungen zu. Vorausgesetzt, das Android-Gerät wird unterstützt (leider funktioniert die App nicht auf allen Geräten – hier scheint es hin und wieder zu Kompatibilitäts-Problemen zu kommen), steht die Internet-Verbindung pronto zur Verfügung. Alles, was es dazu braucht, ist ein USB-Kabel – sowie ggf. (bei Einsatz von Windows) der entsprechende Geräte-Treiber für den Androiden. Optional kann man sich nun auch bei jedem Anstecken des USB-Kabels automatisch fragen lassen, ob die Internet-Verbindung hergestellt werden soll.

Es gibt eine gratis Testversion, die allerdings hinsichtlich der Verbindungszeit eingeschränkt ist. Damit lässt sich jedoch auf jeden Fall feststellen, ob die App mit dem eigenen Gerät überhaupt funktioniert. Die Vollversion kann man sodann für knapp vier Euro erwerben. Weiteres lässt sich auch (in englischer Sprache) bei AndroidAuthority nachlesen.

Sollte diese App nicht mit dem Wunschgerät kompatibel sein, gibt es noch ein paar Alternativen. Diese bestehen jedoch nicht aus einer einfachen App, sondern bedürfen oftmals ein wenig Handarbeit (etwa das Anpassen von Systemdateien auf dem Androiden – wie der Datei `/system/bin/wpa_supplicant` zur Ermöglichung des Zugriffs auf Ad-Hoc Netzwerke per WLAN (siehe XDA-Developers). Oder sie sind auf ein bestimmtes Betriebssystem auf dem PC/Laptop angewiesen – wie im Falle von Android Reverse Tethering for Windows users.

Eine Methode, die sich einfach mit Hilfe einer Terminal-App umsetzen lassen sollte, beschreibt Rotemmiz in einem Beitrag bei StackExchange:

```
ifconfig wlan0 up
iwconfig mode auto
iwconfig wlan0 essid “your SSID” channel 11 mode auto
ifconfig wlan0 10.0.0.x netmask 255.255.255.0
```

Diese Zeilen gilt es natürlich als root auszuführen werden – wobei “your SSID” durch die SSID des Access-Points, und 10.0.0.x durch die für den Androiden zuständige IP-Adresse ersetzt werden müssen.

Eine weitere Möglichkeit beschreibt GWLlosa, ebenfalls wieder bei StackExchange. In seinem Fall ermöglicht die manuelle Anpassung der Konfigurationsdatei `wpa_supplicant.conf` den Zugriff auf AdHoc-Netzwerke via WLAN. Glücklich, wer in den erweiterten WLAN-Einstellungen (siehe WLAN Konfiguration) den Kanal für „Peer-to-Peer Verbindungen“ einstellen kann: Ein solches Gerät unterstützt den Ad-Hoc Modus von Haus aus.

Wesentlich einfacher geht es natürlich, sofern der Laptop/PC ein vollwertiges „Infrastruktur“ WLAN (im Gegensatz zu „AdHoc“) bereitstellen kann, oder man über einen passenden WLAN Reiserouter (bereits für unter 50 Euro im Handel erhältlich) verfügt. Dann nutzt man das bereitgestellte WLAN nämlich einfach, wie jeden anderen Hotspot auch.

Hotspots & WLAN-Finder

Unterwegs und insbesondere im Ausland, wo man auf teure Roaming-Kosten gern verzichtet, ist so mancher scharf auf ein verfügbares WLAN. Viele Hotels und mittlerweile auch Cafés bieten ihren Gästen auf Anfrage einen Zugang auf diese Weise an. Auch die Telekom hält etliche Hotspots bereit. Die Frage, die sich stellt, ist nur: Wie bzw. wo findet man diese Zugangspunkte?

Natürlich gäbe es da zunächst den in Android integrierten WLAN-Scan, der sich unter *Einstellungen › WLAN* findet. Während sich dieser wunderbar eignet, so man das Netzwerk kennt, zu dem man sich verbinden will (also etwa das im eigenen Hotel, zu dem der Portier gerade die Zugangsdaten auf einem Kärtchen übergeben hat) – tauchen die ersten Probleme spätestens auf, wenn kein WLAN angezeigt wird. Schon fünfzig Meter weiter könnte eines sein – nur in welcher Richtung? Und ist ein als „offen“ gekennzeichnetes WLAN auch wirklich „offen“, oder kommt man nach dem Verbindungsaufbau lediglich mit dem Browser auf eine Anmeldeseite?

Eine Übersicht nennt eine Reihe von Tools, die genau diese Fragen beantworten wollen. So kann man beispielsweise Telekom-Hotspots relativ leicht aufspüren: Sie lassen sich an ihrer SSID erkennen. Da die zu verwendenden Zugangsdaten bei allen diesen Hotspots identisch sind, stünde einer automatischen Anmeldung eigentlich nichts im Weg. Zu dumm nur, dass diese

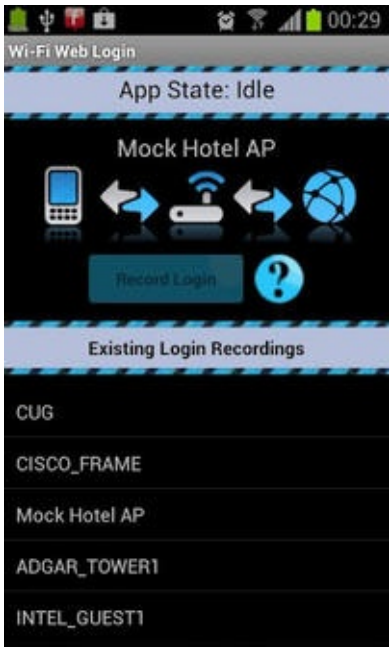
jeweils nach dem Aufbau der (unverschlüsselten) WLAN-Verbindung über eine Webseite erfolgt.

Die „originale App“ von der Telekom selbst müsste das Problem doch eigentlich lösen – zumindest für die zahlreichen eigenen Hotspots. Werfen wir daher einen Blick auf Online Manager: Diese App soll unabhängig von der SIM-Karte, oder sogar ohne selbige funktionieren – das wäre doch eine prima Sache für Tablet-Nutzer! Also flugs mal auf mein Testgerät (ohne SIM) gezogen und geschaut. Das Ergebnis ist ernüchternd: Im heimischen WLAN angemeldet, findet die App in München ganze 0 Hotspots, davon 0 in der näheren Umgebung. Wie kann das sein? Beschreibung nochmal lesen: Aha, die Suche funktioniert nur, wenn man an einem Telekom-Hotspot angemeldet ist. Tolle Logik: Um einen Telekom-Hotspot zu finden, muss man in einem solchen angemeldet sein. Dann allerdings braucht man ihn ja eigentlich nicht mehr suchen.

Ich teste ein wenig weiter. Am Hauptbahnhof gibt es ja einen Telekom-Hotspot. Und der wird von der App auch sogleich entdeckt. Die Anmeldung erfolgt *ohne Rückfrage, ob ich das überhaupt will* – und der Text auf dem Screen bestätigt, was wir schon immer ahnten: Bayern gehört nicht zu Deutschland („Sie nutzen einen Hotspot außerhalb Deutschlands – es können Roamingkosten entstehen“). Das sorgt natürlich zunächst für einen Lacher – und macht kurz darauf nachdenklich: Roamingkosten? Danke für den Hinweis! Wäre es tatsächlich so, hätte mir die App jetzt Kosten fabriziert, ohne mich zu fragen! Und überhaupt: Warum Roaming-Kosten? Geht es hier nicht um WLAN? Im Prinzip ja – aber die Abrechnung erfolgt ja über die Telekom, und die macht ja Telefon. Folglich: Befindet man sich aus ihrer Sicht im Ausland, fallen Roaming-Kosten an. Kann man verstehen – muss man aber nicht.

Die nächste Überraschung kommt Stunden später. Wieder zu Hause angekommen, verbindet sich das Gerät nicht mehr mit dem heimischen WLAN. Was ist denn jetzt los? In den Einstellungen nachgeschaut: Dankeschön! Die App hat alle gespeicherten WLANs *deaktiviert*! Man darf sich nun also zunächst wieder überall manuell verbinden. Außer beim nächsten Telekom-Hotspot – außerhalb Deutschlands …

Den gerade beschriebenen Test habe ich zwar 2013 durchgeführt (und die App gleich wieder deinstalliert) – die Fakten sind jedoch geblieben, wie Playstore-Kommentare u. a. von Oktober 2015 zeigen. Also selbst wenn andere Apps in diesem Sektor vielleicht nicht mehr ganz so aktuell sind, funktionieren sie wenigstens – und sorgen nicht für derartige Bauchschmerzen.



Müssen wir auf den entsprechenden Komfort nun also verzichten, und uns jeweils manuell über die entsprechende Webseite anmelden? Zum Glück gibt es noch Alternativen, die sogar weitere Anbieter abdecken können. Wie beispielsweise abgebildetes WiFi Web Login, welches ich seit Jahren erfolgreich verwende. Auch wenn sich die genannte Webseite wieder einmal ändern sollte, wird diese App dabei nicht unbrauchbar. Man muss sich dann lediglich wieder einmalig manuell anmelden. Denn *WiFi Web Login* schneidet diese manuelle Anmeldung mit (einschließlich etwaiger Eingabefehler), und spielt sie für die automatische Anmeldung wie ein Makro ab. Bei Änderung des Anmeldeformulars muss also lediglich das Makro erneut aufgezeichnet werden. Ein Vorgehen, welches bei jedem derartigen Netzwerk funktionieren sollte. Davon kann man sich drei Tage kostenlos überzeugen – anschließend muss die App für gut einen Euro käuflich erworben werden, damit sie weiterhin ihren Dienst tut.

Damit wäre geklärt, wie man sich automatisch anmelden kann. Um dies tun zu können, muss man jedoch zunächst wissen, wo sich ein passendes WLAN befindet – wobei uns wiederum eine ganze Reihe Apps behilflich sein wollen. Wie etwa WeFi, welches ich noch aus „Symbian-Zeiten“ kenne. Die Entwickler sind also bereits seit längerem in diesem Bereich aktiv, und verfügen über entsprechende Erfahrungen. Entsprechend groß ist auch die dahinter stehende Community.

Auf der Website kann man bereits vor Urlaubsantritt nachschauen, wo wohl die besten Hotspots am Ferienort zu finden sind. Inklusive Benutzer-Kommentaren natürlich – das macht eine Community aus. Und vor Ort gibt es dann zusätzlich den „On Demand Scanner“ – irgendwie müssen neue Spots ja auch in die Liste kommen. Als Mitglied der Community hat man nämlich die Möglichkeit, neu gefundene Hotspots in die Datenbank einzutragen. Da sich eine recht rege Community an dieser Aktivität beteiligt, wächst die Datenbank beständig – die Chancen sind also gut, für den Zielort eine Reihe passender Hotspots im Voraus zu finden.

Der On-Demand Scanner zeigt übrigens nicht nur an, dass da „ein WLAN“ verfügbar ist. Gefundene Netze werden in mehrere Kategorien unterteilt. Das wären zunächst verschlüsselte (also solche, für die man zunächst einen Netzwerk-Schlüssel benötigt), und offene (ohne Verschlüsselung). Doch letztere werden nochmals weiter unterteilt: *WeFi* erkennt automatisch, ob sich hinter einem vermeintlich „offenen Zugang“ noch eine Web-Anmeldung verbirgt. Außerdem werden die Netzwerke besonders hervorgehoben, bei denen eine gute

Internet-Verbindung bereits durch die Community bestätigt wurde. Und das sind nur einige der verfügbaren Details.

Einen anderen Weg beschreitet WiFi Manager, indem er gefundene Netze u. a. auch grafisch darstellt (siehe Screenshot). So lässt sich nicht nur einfach erkennen, welches das stärkste WLAN in Reichweite ist – sondern auch, ob es sich nicht eventuell mit einem anderen überlappt, was die Kanäle betrifft (und die Signalstärke damit wieder relativieren würde). Den einzelnen Zugangspunkten lässt sich auch eine Beschreibung hinzufügen, so dass man nicht auf den kryptischen Namen allein angewiesen ist. Alles genau beschrieben findet sich auf der Projektseite.

Durch In-App Payment über Google Checkout lässt sich ein Premium-Package hinzu erwerben, welches u. a. das Wechseln vorhandener Netze per Widget und das Ausschließen unerwünschter Netze beinhaltet.

War-Driving

Kriegsspiele? Panzerschlachten, Schiffe-Versenken? Mit all dem hat *War-Driving* herzlich wenig zu tun. Eher mit „da War doch was …“ Schauen wir also zunächst einmal nach, was genau es mit diesem Begriff auf sich hat, und konsultieren dazu die Wikipedia:

> *Wardriving ist das systematische Suchen nach Wireless Local Area Networks mit Hilfe eines Fahrzeugs. Der Begriff leitet sich von Wardialing ab, einer Methode, durch Durchprobieren vieler Telefonnummern offene Modem-Zugänge zu finden, wobei einige Wardriver die drei Anfangsbuchstaben als Backronym für „Wireless Access Revolution“ sehen (wohl nicht zuletzt, um dem Begriff den martialischen Klang zu nehmen).*

Im Prinzip geht es also darum: Wir machen das Gleiche wie auch Google mit seinen tollen Autos, nur in kleinerem Maßstab – und legen eine Karte verfügbarer WLAN-Netze der Umgebung an. Dabei helfen uns Tools wie beispielsweise Wigle Wifi Wardriving. Die App lässt sich sehr detailliert konfigurieren, und somit auf die eigenen Bedürfnisse anpassen. Gesammelte WLANs können, einen Account vorausgesetzt, zur zugehörigen Community hochgeladen werden, wo sie dann allen angemeldeten Mitgliedern (und nur diesen) zur Verfügung stehen. Die eigene Sammlung lässt sich auch gezielt durchsuchen, z. B. mittels regulären Ausdrücken über die SSID. Ebenso ist ein Export nach CSV und auch KML möglich.

Als weiteren Kandidaten möchte ich noch G-MoN für Android nennen. Diese deutsche App bezeichnet sich selbst als *WarDriving Scanner & GSM / UMTS Netmonitor und Messtool für Funknetzplaner und Optimierer*. Sie scannt alle in Reichweite befindlichen Netzwerke, und speichert deren Daten einschließlich der GPS-Informationen in einer internen SQLite-Datenbank auf dem Androiden ab. Gefundene Zugangspunkte lassen sich per KML-Export in *Google Earth* (und sicher auch anderen Anwendungen, welche dieses Format unterstützen) anzeigen, auch ein CSV-Export ist möglich.

In Reichweite befindliche WLANs zeigt die App auf Wunsch auf in einer integrierten Kartenansicht (siehe Screenshot) an. Wie man dort sieht, werden gefundene Mobilfunkzellen ebenfalls angezeigt; die zu sehende „rote Linie“ weist vermutlich vom eigenen Standort zum ausgewählten (und in der Kopfzeile angezeigten) WLAN.

Weitere und ausführlichere Informationen zu dieser App finden sich im Wiki des Projekts – weitere Apps hingegen in dieser Übersicht.

Dienste bereitstellen

Ists möglich, daß so harte Sinnen
Und ungemeine Sprödigkeit
Die zarten Glieder hegen können?
Die mir ein andres prophezeyt.
Hier trifft es nicht von Engeln ein,
daß sie dienstbare Geister seyn.

(Aus: Johann Christian Günthers, „Als er sich über ihren Eigensinn beschwehrte“)

In diesem Kapitel wollen wir – abschließend zum Thema Netzwerk – einmal betrachten, welche Dienste ein Androide „nach außen“ zur Verfügung stellen kann. Dass ich das Wort „Netzwerk“ hier ein wenig weiter fasse, lässt sich bereits an der Überschrift des ersten Abschnittes feststellen:

Kabelgebunden

Wie jetzt: Kabelgebunden? Der Androide hat doch gar keinen Anschluss für ein Netzwerkkabel! Oder vielleicht doch? Im weitesten Sinne kann man ein USB-Kabel ja auch als „Netzwerk-Verbindung“ betrachten. Spätestens beim Thema USB-Tethering wird da jeder zustimmen. Die einfachste Server-Funktion vergisst man jedoch ganz leicht:

Androide als USB-Stick

Mit dem Aufkommen von USB-Anschlüssen und passenden Speichermedien verlor die Floppy-Disk recht schnell ihren Reiz für den „täglichen Datentransport“. Dank ihrer einfacheren Handhabung und der größeren Kapazität überflügelten sie auch zügig die CD und DVD – woran auch die bald erschwinglicher werdenden „Rewritables“ (wiederbeschreibbaren CDs/DVDs) nichts mehr zu ändern vermochten. Dokumente und mehr wanderten auf USB-Sticks vom Büro nach Hause (und in die Gegenrichtung), Fotos zu Freunden und Verwandten, und so weiter.

Dann kamen die MP3-Player, die nicht viel größer waren als die genannten USB-Sticks – und erfüllten eine Doppelfunktion: Unterwegs konnten sie genutzt werden, um Musik zu hören. Den Datentransport übernahmen sie sozusagen nebenbei.

Heute werden beide gleichermaßen durch Smartphones verdrängt: Mir ist kein Smartphone bekannt, für das es keinen MP3-Player und keinen USB-Anschluss gäbe. Und was spricht dagegen, die eingelegte SD-Karte auch als „externe Festplatte“ zu nutzen? Der verfügbare Speicher kann mit dem von USB-Sticks durchaus mithalten, wenn die entsprechende Karte eingelegt ist. Extra Software

wird dafür nicht benötigt (zumindest nicht, sofern UMS verfügbar ist – ab Android 4.x ist dieses leider meist von MTP verdrängt worden), und das verwendete Dateisystem ist von Werk aus kompatibel zu allen gängigen Computern.

Eine einfache Sache also – und so mancher greift sich vielleicht jetzt an den Kopf, weil er nicht an diese Möglichkeit gedacht hat. Sein Smartphone hat man ohnehin dabei – warum also noch weitere Dinge einstecken, die man damit bereits abdecken kann? Das Kabel? Auch das ist bei den kurzen Akku-Laufzeiten überall vorhanden. Also hören wir doch auf Altmeister Goethe:

Willst du immer weiter schweifen?
Sieh, das Gute liegt so nah.
Lerne nur das Glück ergreifen,
Denn das Glück ist immer da.

(Aus: Johann Wolfgang von Goethe, „Erinnerungen“)

Wem das jedoch nicht „netzwerkig“ genug ist (bzw. wer mangels *UMS* auf Kompatibilitäts-Probleme stößt): Im kabellosen Bereich wird unter Datei-Server beschrieben, wie man den „Datenträger“ auch ohne Kabel-Einsatz als solchen vom PC aus nutzen kann.

USB-Tethering

Im Abschnitt WLAN Tethering habe ich bereits beschrieben, wie man Tethering drahtlos (per WLAN bzw. Bluetooth) bewerkstelligen kann. Welche Gründe könnte es geben, dass man stattdessen lieber ein Kabel nehmen möchte? Da wären tatsächlich ein paar nennenswerte verfügbar:

- Das ans Netz zu bringende Gerät verfügt weder über WLAN, noch über Bluetooth – wohl aber über einen USB-Anschluss (soll selten, aber vorkommen)
- Man möchte potentiellen Lauschern ihr „böses Tun“ nicht unnötig erleichtern – in eine drahtgebundene Verbindung können selbige sich wohl kaum einklinken
- Man möchte vermeiden, dass ein zufällig vorbeifahrendes Google-Auto das gerade als Hotspot fungierende Smartphone erfasst und in die Datenbank einträgt
- Die technischen Voraussetzungen für WLAN-Tethering / Hotspot Modus sind nicht gegeben (Android < 2.2, kein root, und keine der verbleibenden Apps funktioniert – unwahrscheinlich, aber möglich)

Für den letzten Punkt wurde im genannten Abschnitt ja bereits *PdaNet* genannt, welches ein Tethering per Bluetooth und USB bewerkstelligen kann. Auch das ebenfalls in der zuständigen Übersicht genannte EasyTether kümmert sich um dieses Problem, und zwar bereits ab Android 1.5 (Cupcake). Es wird dafür kein

root benötigt, und dennoch stehen mit dieser App umfangreiche Features zur Verfügung – wie der Screenshot es bereits erahnen lässt. Nicht umsonst ist diese App so gut bewertet (über 4 Sterne bei über 20.000 Bewertungen). Einziger Haken: Die freie Version ist etwas eingeschränkt. Kein HTTPS, und auch „Instant Messengers“ werden geblockt. Für die Vollversion werden aber eben einmal gut zehn Euro fällig …

Weitere Alternativen finden sich wieder einmal in der passenden Übersicht.

Sind es eher die anderen Punkte, und auf dem „Hotspot-Gerät“ läuft Android 2.3 oder höher (eher letzteres), lässt sich USB-Tethering mit Bordmitteln einrichten (siehe Screenshot unter Tethering): Der entsprechende Konfigurations-Bildschirm findet sich unter *Einstellungen › Drahtlos & Netzwerk › Tethering & mobiler Hotspot*.

Kabellos

Setz die Segel, mach die Leinen los,
da draußen warten deine Träume.
Am Horizont ist Gold, siehst du es scheinen?

(Aus: Peter Maffay, „Der Mensch auf den du wartest“)

Also dann: Die Leinen Los! Die Kabel weg! Und den Androiden zum Server umfunktionieren. Wozu das gut sein soll? Da fallen mir auf Anhieb viele tolle Dinge ein: Kabelloser USB-Stick, die Urlaubsfotos ohne Umweg auf einem größeren Bildschirm betrachten … Einige davon sollen im Folgenden vorgestellt werden.

Für all diejenigen, die nicht alles separat lesen – aber alles haben wollen, lohnt sicher ein Blick auf die App Servers Ultimate. Hier scheint wirklich nahezu alles integriert, was in diesem Bereich möglich ist:

Die gratis-Version ist auf maximal zwei gleichzeitig laufende Services beschränkt – was zumindest ausreichend ist, um die Funktionalität zu testen. Die volle Leistung lässt sich sodann mit der Vollversion für knapp 10 Euro ausreizen. Mit 4,3 bzw. sogar 4,8 Sternen sind die Bewertungen für beide Varianten weit über dem Durchschnitt.

Datei-Server

Das Thema „USB-Stick“ hatten wir ja bereits behandelt. Das einzige „störende Element“ dabei war das benötigte USB-Kabel (sowie ggf. die MTP-Software auf dem Computer). Wollen wir doch Mal schauen, ob es nicht verzichtbar ist! Gleich vorab: Mit Bordmitteln sicher nicht – zumindest habe ich bislang noch nichts entdecken können. Doch wie ich Murphy kenne: Unwahrscheinlich ist es nicht, dass eine neue Android-Version das genau dann anbietet, wenn dieses Buch frisch gedruckt ist …

Beim Thema „Datenserver“ denkt sicher so mancher zunächst an FTP. Dies ist auf jeden Fall die ressourcenschonendste Art, Dateien zu übertragen – und so kommt beispielsweise FTPServer auch als handliches Paket daher: Gerade einmal 100 kB Download sind es. Einmal installiert und gestartet, stellt die App einen kleinen FTP-Server auf dem Androiden zur Verfügung, auf den man dann von allen Geräten, die den Androiden erreichen können, Zugriff hat. Bei den meisten Systemen genügt dafür bereits ein Webbrowser – in dessen Adresszeile man das eingibt, was einem die App als URL anzeigt. Und schon lassen sich Dateien zumindest herunterladen. Bequemer und definitiv in beide Richtungen geht das natürlich mit einem „richtigen FTP-Client“ wie z. B. FileZilla (sowohl unter Windows, als auch unter Linux und auf dem Mac). Alternative FTP-Server sind zwar größer im Daten-, aber oft auch im Funktionsumfang.

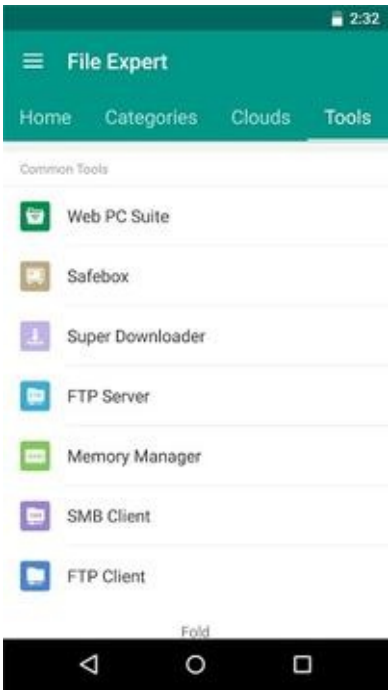
Wem FTP zu spartanisch ist, der kann z. B. zu Samba greifen. Der Name geht auf die Bezeichnung des verwendeten Protokolls zurück: SMB („a“ an passenden Stellen einfügen) steht für **S**erver **M**essage **B**lock, vielen auch einfach als „Windows Freigaben“ bekannt. Mit diesen „Laufwerken“ kann Windows also von Haus aus (und Linux Dank Samba ebenso) prima umgehen, der Androide wird einfach als Laufwerk in das lokale Dateisystem eingebunden. Allerdings hat die Sache einen kleinen Haken: Ohne root ist kein Zugriff auf die standardmäßig von SMB verwendeten Ports möglich – und auf „alternativen Ports“ findet Windows den

Server nicht. Bleiben im Falle von erforderlicher Windows-Kompatibilität also nur root-Apps, wie beispielsweise SambaDroid. Root vorausgesetzt, ließe sich unser eingangs gesetztes Ziel eines “drahtlosen USB-Sticks” hiermit umsetzen.

Eine weitere Möglichkeit der direkten Einbindung in das lokale Dateisystem ist mittels WebDAV gegeben. Von Haus aus unterstützen dies sowohl Linux als auch Mac OS, für Windows muss die entsprechende Client-Software separat besorgt werden. Wie die Einrichtung unter allen drei Systemen vonstatten geht, ist auf der Website von DavDrive beschrieben. Womit auch gleich der Name einer Server-App für Android benannt wäre: DavDrive stellt die SD-Karte (und in der Vollversion auch den internen Speicher, diesen allerdings nur im Lesezugriff) per WebDAV zur Verfügung. Die Vollversion lässt sich für knapp einen Euro im Playstore erwerben, und stellt zusätzlich zur Basis-Funktionalität verschlüsselte Verbindungen (HTTPS), Autostart-Funktionalität, Tasker/Locale Plugin, und einiges mehr zur Verfügung. Unter Windows 7/Vista lässt sich der Androide damit sogar als Laufwerk einbinden.

Steht die Sicherheit im Vordergrund, muss natürlich auch SSH Erwähnung finden. Mittels dieses Netzwerkprotokolls lassen sich nicht nur Dateien übertragen; die Möglichkeit eines Terminal-Zuganges (also zur Kommandozeile des Servers) ist ebenfalls gegeben.

Einen passenden Server stellt SimpleSSHD auf dem Androiden zur Verfügung. Die Implementierung dieser App basiert auf Dropbear, was als besonders ressourcenschonend gilt. Auch wenn root nicht zwingend vorausgesetzt wird, bringt dieser in diesem Zusammenhang mindestens einen Vorteil mit: Die Verwendung des Standard-Ports 22 wird dadurch möglich, so dass der entsprechende Parameter beim Zugriff (zur Angabe des zu verwendenden Ports) entfallen kann. Mit gerade einmal einem Megabyte fällt die App recht kompakt aus – und bietet neben reinem SSH auch SCP sowie RSync.



Zum Abschluss sei mit File Expert noch ein ganz spezieller Kandidat genannt. Dieser ist weit mehr als nur ein Dateiserver. Man muss sogar soweit gehen, zu sagen: Das macht er nebenbei. Hauptsächlich will die App ein vollwertiger Dateimanager sein – mit allem, was dazu gehört: Kopieren, Verschieben, umbenennen von Dateien und Ordnern, Zugriff auf Daten im Netz per SMB, FTP, SFTP sowie FTPS, Unterstützung für Archive (ZIP/RAR/GZIP/TAR/TGZ/BZ können wie Ordner navigiert und betrachtet werden, ZIP-Dateien lassen sich auch erstellen), Anzeige von Texten und Bildern … Sogar ein App-Manager ist integriert.

Im Umfeld dieser Übersicht sind jedoch die Server-Eigenschaften hervorzuheben. Und hier punktet *File Expert* mit der Bereitstellung sowohl eines FTP- , eines Bluetooth- und eines Webservers. Über alle drei lässt sich im WLAN drahtlos auf die Daten des Androiden zugreifen. Es können Dateien hoch- oder heruntergeladen, Dateien und Ordner umbenannt oder gelöscht werden – alles über ein einfach gehaltenes Interface.

Zu den meisten hier genannten Apps gibt es natürlich noch weitere Alternativen. Sie alle in diesem Kapitel zu beschreiben, würde ein wenig zu weit gehen – daher finden sie sich, wie gewohnt, in einer Übersicht bei *IzzyOnDroid*.

Web-Server

Die Webserver möchte ich grob in drei Kategorien einteilen. Die erste befasst sich, wie bereits die im vorigen Abschnitt beschriebenen Server, mit der Bereitstellung von Dateien – dient also in erster Linie dem Dateiaustausch. In der zweiten Kategorie finden sich sodann Webserver, die vorwiegend Inhalte (statische Webseiten) bereitstellen. Und schließlich soll es auch noch solche geben, die Support für Skriptsprachen wie PHP bieten. Eine Übersicht zu diesen dreien findet sich wieder bei *IzzyOnDroid*.

WiFi File Transfer stellt das lokale Dateisystem ausschließlich per HTTP im WLAN zur Verfügung. Über eine ansprechende Web-Oberfläche lassen sich Dateien hoch- und herunterladen (auch im Batch-Betrieb, also mehrere auf einmal), umbenennen, kopieren, ZIP-Archive erstellen oder auspacken, und mehr. Für die Betrachtung von Fotos steht sogar

eine integrierte Thumbnail-Galerie zur Verfügung. Die Einschränkung der Gratis-Version besteht in einer Limitierung beim Upload: Dateien größer als 4MB werden hier nicht unterstützt (wohl aber in der Kaufversion – die mit über viereinhalb Sternen eine sehr gute Playstore-Bewertung vorweisen kann).

Mit kWS begeben wir uns zu den „Content Servern“ – also jenen Webservern, die sich um den kompletten Webservice kümmern. Unter Android beschränkt sich dies zumeist auf sogenannten „statischen Content“, also reine HTML-Seiten mit Bildern etc. Die Unterstützung von Skriptsprachen bildet da eher die Ausnahme. *kWS* bietet in dieser Hinsicht zumindest SSI für grundlegende Dinge. Dennoch ist die App mit weniger als 100kB Download ein Leichtgewicht. Und bringt in dem kleinen Paket noch einiges mehr unter: Verzeichnis-Indexe, Resumable File-Downloads, Zugriffsschutz (Basic und Digest Authentication) – sogar Verzeichnis-Downloads (.tar, .tgz und .zip) und ein integrierter DynDNS Updater sind mit dabei. Bis zu zwanzig parallele Verbindungen sind möglich, auch Protokolldateien (Log-Files) werden erstellt.

Noch mehr bietet die Pro-Version, die sich für ca. zwei Euro im Playstore erwerben lässt: Diese beherrscht außerdem SSL/TLS für gesicherte Verbindungen (HTTPS), komprimierte Übertragung (gzip), Verzeichnis-Indexe in den Formaten JSON und XML, und bietet weitere zusätzliche Features. Beide Apps wurden jedoch seit Dezember 2013 nicht mehr aktualisiert.

Darf es gern ein wenig mehr sein? Wer etwa zusätzlich gern mit PHP und einer Datenbank arbeiten möchte, sollte einen Blick auf den Palapa Web Server werfen. Mit gut 20 MB ist diese App zwar etwas umfangreicher – gleiches trifft aber auch auf die Feature-Liste zu: Lighttpd versieht als Webserver seinen Dienst, PHP sorgt in einer aktuellen Version für Scripting-Support, und eine ebenfalls relativ aktuelle MySQL Datenbank stellt Platz für dynamische Inhalte zur Verfügung. Darüber hinaus ist mit MSMTP auch gleich eine Möglichkeit zum Mail-Versand, und mit *Web Admin* eine Administrations-Oberfläche (für die Bedienung im Web-Browser) integriert; für die Datenbank besorgt phpMyAdmin letzteres. Der Autor wirbt mit Attributen wie „perfektionistisch und flexibel“, und preist den schonenden Umgang mit Ressourcen an. Kurzum: Ein durchaus interessantes „Rundum-sorglos-Paket“ mit positiver Bewertung im Playstore.

Zu guter Letzt noch ein Wort zu einem ganz speziellen Kandidaten, der bereits im Kapitel Androiden vom PC aus verwalten vorgestellt wurde: Der PAW Server ist ein mächtiges Instrument, der u. a. auch einen Webserver integriert hat. Und

selbiger bringt auch Support für Skriptsprachen mit, was zumindest für Webentwickler interessant sein dürfte. Da wäre zum Einen die eigene Skriptsprache, mit der sich auch zahlreiche Systeminformationen darstellen lassen. Zum Anderen wird über ein Plugin zusätzlich PHP unterstützt.

TUNING

Einleitung und Überblick

Im vierten Teil dieses Buches wenden wir uns schließlich dem Tuning zu: Wie lässt sich möglichst viel aus dem Androiden herausholen?

Wobei zuerst einmal geklärt werden sollte: Möglichst viel wovon? Altbekannt: Wenn ein Manta-Fahrer in den Supermarkt geht, will er Mantarinen und Tunefish. Und später dann die Kiste tiefer legen (ja, auch die auf dem Friedhof). Aber was wollen wir nun mit dem Tunen erreichen?

Prinzipiell gibt es da zwei Richtungen: „Schneller, weiter, höher“ (Performance/Geschwindigkeit), und „Meiner ~~ist~~ läuft länger als Deiner“ (Akku-Laufzeit); bis auf einige wenige Ausnahmefälle schließen sich diese Tuning-Ziele i. d. R. gegenseitig aus. Des Weiteren gilt es zu beachten: Was kann „Otto Normalanwender“ umsetzen, und was richtet sich eher an „Nerd Root“? Bevor wir uns auf die Details einlassen, zunächst eine Kurzübersicht der Möglichkeiten:

Generelle Maßnahmen

Toll, dass es Dinge gibt, die beides bedienen: Mehr Speed ohne zusätzliche Akku-Belastung nimmt man gern in Kauf, längere Akku-Laufzeiten ohne Performance-Einbußen ebenso. Utopisch? Zu wahr, um schön zu sein? Es gibt sie aber wirklich, die derartigen Tuning-Maßnahmen:

- Aufräumen:
  - Nicht mehr verwendete/benötigte Apps deinstallieren
  - Cache bereinigen
  - Selten genutzte Apps: Am automatischen Starten hindern (ggf. root benötigt)
  - Häufig genutzte Apps möglichst im internen Speicher installieren (auch an die ständig im Hintergrund laufenden Apps denken). Dieser hat einerseits kürzere Zugriffszeiten (ist also schneller), und andererseits weniger Energiebedarf bei Zugriffen.
- Live-Wallpaper und sonstige „animierte Dauerrenner“: Wer drauf verzichten kann, sollte das tun – denn die nuckeln anständig am Akku, und knabbern auch gern an der CPU!
- Auf Task-Killer weitgehend verzichten (Ausnahme: Wenn sich etwas „aufgehängt“ hat). Diese tragen i. d. R. weder zu verbesserter Performance noch längerer Akku-Laufzeit bei – sondern eher ganz im Gegenteil …

Ein netter Nebeneffekt der ersten beiden Aufräumaktionen: Es wird wieder

interner Speicher frei. Eine Nebenwirkung, die man auch gern mitnimmt.

Bessere Akku-Laufzeit

Warum nur ist am Ende des Akkus noch so viel Tag übrig? Eine Frage, die sich sicher manch einer (und nicht nur einmal) schon gestellt hat. Was hier Abhilfe schafft, geht nicht selten zu Lasten der Performance – aber häufig so unmerklich, dass man es durchaus in Erwägung ziehen kann:

- WLAN komplett abschalten, wenn's nicht gebraucht wird. Das spart teilweise enorm (siehe Anhang)! Und lässt sich auch automatisieren. Die Verzögerung bei der bedarfsmäßigen Aktivierung ist zu verschmerzen.
- Wer kein mobiles High-Speed benötigt: 3G (bzw. LTE) aus, das frisst auch ganz nett. Wer es hin und wieder benötigt, kann es bei Bedarf aktivieren (Youtube per mobilem Datennetz macht mit 2G natürlich weniger Spaß). Siehe dazu aber auch *2G versus 3G: Spart 2G wirklich so viel Akku?* im ersten Band um herauszufinden, wann sich das lohnt.
- GPS benötigt im Standby zwar so gut wie keinen Saft – aber wenn es deaktiviert ist, kann auch keine „böse App" für „Werbestandort" darauf zugreifen.
- Helligkeit des Displays weitmöglichst herunterregeln. „Aufdrehen" dann im Bedarfsfall. Das Display-Timeout möglichst kurz (aber nicht zu kurz) einstellen. Da das Display einer der größten Akku-Fresser ist, liegt hier auch großes Potential.
- Die Datensynchronisierung muss evtl. nicht (für alle Apps) ständig laufen (Hintergrunddaten deaktivieren, Intervalle anpassen, Apps ausnehmen)
- Verschiedene Apps helfen ggf. bei der Laufzeit-Verlängerung, indem sie gewisse Aufgaben automatisch ausführen (JuiceDefender, GreenPower, Greenify …)
- Gute Pflege nimmt der Akku gern an (Temperaturbereich, Ladezustand, Kalibrierung *hust, hust …*)
- root und Modder: CPU nicht über-, sondern ggf. eher untertakten. Ich weiß, das klingt nicht besonders „cool" – spart aber Akku. Und auch dies lässt sich automatisieren: So benötigt man beispielsweise bei ausgeschaltetem Display und zu nachtschlafender Stunde meist nicht die volle CPU-Last.

Mehr Speed

Speedy Gonzales überholen, mit einem Androiden der ersten Generation? Da stehen die Chancen eher schlecht. Aber neben den generellen Tipps gibt es auch hier noch die eine oder andere Spezialität:

- RAM Bereinigen (Android-interne Einstellungen optimieren). Die persönliche

richtige Mischung aus „verfügbarem RAM“ und „schnell verfügbaren Apps“ festlegen.

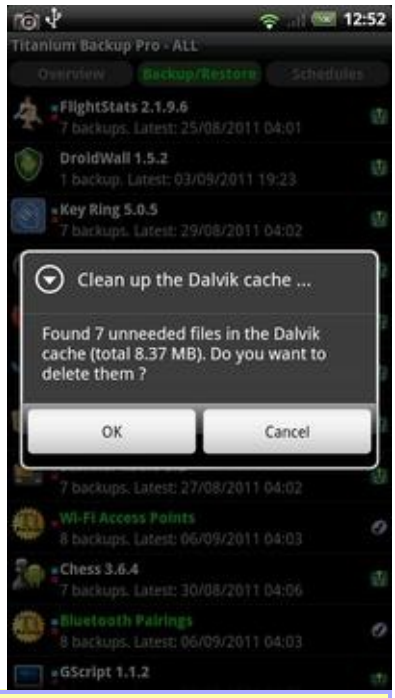
- Eventuell Swap-Space nutzen (braucht root)
- Nur für bestimmte Situationen, wo es darauf ankommt: CPU ggf. leicht übertakten. Frisst aber auch mehr Akku. Und benötigt root.

Generelle Tuning-Maßnahmen

Toll, dass es Dinge gibt … ach, das hatten wir schon? Trotzdem schön, dass all die in diesem Kapitel beschriebenen Maßnahmen von jedermann (und -frau, auch -innen bei Bedarf) eingesetzt werden können – von ein paar klitzekleinen Ausnahmen einmal abgesehen. Aber Schluß mit den ewigen Vorreden, und ran an die Bouletten!

Nicht mehr verwendete/benötigte Apps deinstallieren

Ein unnötiger Vorschlag? Verstehe: Ein HTC Wildfire, oder gar ein Gerät mit noch weniger internem Speicher im Einsatz. Trotzdem nicht für jeden selbstverständlich – aber: Alles raus, was keine Miete zahlt!



Und was soll das bringen? Mehrere Dinge. Zuerst einmal das offensichtliche: Es wird wieder interner Speicher frei, der zuvor von der App belegt wurde. Aber auch der, den die Daten der App mit Beschlag belegten. Und der Platz, den der Cache der App in Anspruch nahm (ha! Genau, da war auch noch was).

Gerade bei Geräten mit wenig internem Speicher kann das einiges bringen – denn dieser frei gewordene Platz steht nicht nur neuen Apps zur Verfügung, sondern kann auch als Cache genutzt werden (das geschieht automatisch). So muss beispielsweise eine noch aktuelle Datei nicht aus dem Netz neu geladen werden – was schnelleren Zugriff und geringeren Stromverbrauch bedeutet.

Hat man eine Reihe von Apps auf diese Weise ins Nirwana befördert, kann man sich als Wurzelmensch (also Android-User mit root) auch Gedanken um die Bereinigung des Dalvik-Cache machen. Hier sind ja ebenfalls Lücken entstanden, und eine Reorganisation würde für einen schnelleren (und kürzeren, also Akku-schonenderen) Zugriff sorgen. Eine Möglichkeit dazu bietet Titanium Backup Pro mit der Option „Cleanup Dalvik Cache“. Alternativ bieten die meisten Custom Recovery-Menüs eine entsprechende Option, den Dalvik-Cache komplett zu löschen – beim nächsten Boot wird er dann automatisch neu aufgebaut. Analoges gilt für den ART Cache (ab Android 5).

Ja toll – und wie wird man eine App nun los? Dazu kann man auf die Bordmittel zurückgreifen, von denen üblicherweise zwei zur Verfügung stehen. Auf jedem Androiden findet sich in den Einstellungen die Möglichkeit, die installierten Anwendungen zu verwalten (üblicherweise unter *Einstellungen › Apps*). Hier wählt man die zu deinstallierende App aus, und betätigt den passenden Button –

naheliegenderweise ist dieser meist mit *Deinstallieren* beschriftet. Ist er ausgegraut, und lässt sich nicht betätigen, muss man evtl. zunächst die App beenden (über den Button *Stoppen erzwingen*). Klappt es danach noch immer nicht, handelt es sich um eine „System-Applikation“ – ohne root lässt sich eine solche nicht entfernen.

Das zweite Bordmittel ist die Playstore-App. In dieser sollte sich die zu entfernende App unter *Meine Downloads* wiederfinden, und ebenfalls mit einem *Deinstallieren*-Button versehen sein. Ist dies nicht der Fall, wurde die App nicht über den Playstore installiert, und es handelt sich evtl. gar um eine System-App. Da hilft uns die Playstore-App dann nicht weiter.

Als alternative Uninstaller kommen zahlreiche Apps in Betracht. Da wäre zum Beispiel das bereits im Abschnitt Apps sichern genannte *AppMonster*, welches neben der Sicherung auch eine Deinstallation von Apps ermöglicht. Ebenso kommen spezielle Deinstallations-Apps in Frage.

Für die vom Hersteller vorinstallierte sogenannte Bloatware kann der Wurzelmensch wieder auf das bewährte *Titanium Backup* zurückgreifen. Um dabei nicht zu viel Schaden anzurichten (indem man beispielsweise eine vom System wirklich benötigte App versehentlich löscht), und sich ebenso wenig die Möglichkeit eines OTA-Updates nicht zu verbauen, lassen sich unerwünschte System-Apps damit auch einfach „einfrieren“: Sie werden dann nicht gelöscht, sondern lediglich „kalt gestellt“ – sind also schlicht „unsichtbar“, werden nicht mehr gestartet, und stören damit auch niemanden. Eine Möglichkeit, die übrigens auch das kostenlose App Quarantine anbietet.

Eine abschließende gute Nachricht: Ab *Ice Cream Sandwich* (Android 4.0) ist von Haus aus die Funktionalität zum „Einfrieren“ von Bloatware zur Nutzung durch den Endanwender ins System integriert – und findet sich an entsprechender Stelle beim beschriebenen „ersten Bordmittel“.

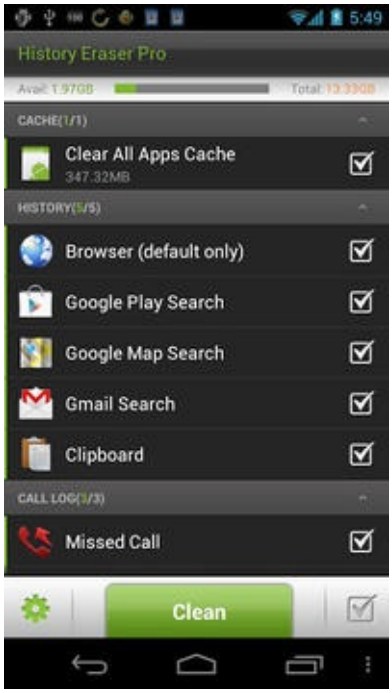
Cache bereinigen

So ein Cache ist schon was feines. Er sorgt nicht nur für schnellere Zugriffe, sondern spart dabei auch gleich noch CPU, Traffic und Akku. Was also ist daran so schmutzig, dass es bereinigt werden muss?

Dafür ganz kurz die allgemeine Erklärung, wie ein Cache funktioniert. Es handelt sich dabei um eine Art „temporären Zwischenspeicher“ – beispielsweise für Dateien, die aus dem Internet geladen werden (wenn lokal vorhanden und aktuell, spart das ein kosten- und zeitintensiveres Nachladen). Oder um die Ergebnisse einer aufwendigeren Berechnung (spart dann Zeit, CPU und Akku). Soweit ist das alles prima und überaus nützlich.

Problematisch wird es erst, wenn kein Verfallsdatum festgelegt wurde, und keine automatische Bereinigungsaktion stattfindet – dann sammelt sich im Cache nämlich überwiegend „Hausmüll“, den niemand mehr braucht. Der wiederum bremst den Zugriff auf sinnvolle Cache-Daten aus, und belegt wertvollen Speicherplatz. Was leider keine Seltenheit ist. Also sprach Zarathustra: Ab und an einmal Hausputz betreiben kann hilfreich sein. Nicht zu oft natürlich, sonst ist der Cache ja insgesamt nutzlos. Wie oft eine Bereinigung sinnvoll ist, lässt sich so pauschal schwer sagen – doch spätestens wenn der interne Speicher knapp wird, ist dies sicher eine der ersten zu ergreifenden Maßnahmen. Negative Seiteneffekte stehen nicht zu befürchten: Findet eine App das gewünschte Element nicht im Cache, wird es einfach neu heruntergeladen bzw. generiert. Was auch der Grund ist, warum eine Bereinigung hin und wieder eine zickige App wieder in die Bahn bringt: Kaputte Elemente im Cache können so etwas verursacht haben. Sind die weg, können sie nicht mehr länger für Probleme sorgen.

Wo geht es nun zum Besenschrank? Mit Bordmitteln heißt es da: Unter *Einstellungen › Apps* alle Apps einzeln anwählen und schauen, ob und wie viel Cache sie belegen, sowie ggf. den passenden Button zum Leeren des entsprechenden Cache betätigen. Keine Übersicht, die vielleicht gar noch nach belegter Datenmenge sortiert wäre.



Bis vor noch gar nicht all zu langer Zeit war das auch das Ende der Fahnenstange für den „Normal-Anwender“. Dinge wie „Alle Caches mit einem Klick löschen“ blieben „Nerd Root“ vorbehalten. Aber das hat sich glücklicherweise geändert.

So bietet beispielsweise History Eraser mehr als nur die Möglichkeit, alle Caches mit einem Klick zu putzen. Die werbefreie Version gibt es für zwei Euro.

Ähnliches bietet CCleaner, dessen Name einigen vielleicht aus der Windows-Welt bekannt ist. Schon die gratis-Version kommt ohne Werbung daher (verspricht zumindest der Entwickler) – und eine Kaufversion ist nicht zu sehen. Mit dabei ist bei dieser Lösung auch ein App-Manager, mit dem man sich weiteren überflüssigen Ballast von Leibe (oder besser vom Gerät) schaffen kann.

Ebenfalls gratis (oder besser: werbefinanziert) gibt es mit 1Tap Cleaner eine weitere App, die sich sowohl um den Cache als auch um die anderen „historischen Fälle“ kümmert, und dabei auch einen Scheduler mit an Bord

hat. Nebenbei sorgt sie sich auch um die sogenannten „App Defaults“, und setzt sie bei Bedarf zurück – hilfreich, wenn man beispielsweise einen alternativen Launcher zum Standard gemacht hat und nicht weiß, wie man diesen Standard nun wieder los wird, um den „alten“ Launcher nutzen zu können (klar geht das auch in dessen Einstellungen unter *Anwendungen verwalten* – aber warum lange suchen?). Neben diversen Sortiermöglichkeiten kann man mit dieser App die Liste der Anwendungen sogar nach Namensbestandteilen filtern. Außerdem lässt sich mit der App das Dateisystem aufräumen (die Liste ist nach den „größten Speicherfressern“ unter den Verzeichnissen sortiert), wobei auch „Hinterlassenschaften“ deinstallierter Apps als solche aufgeführt werden – ein Bereich, für den sonst SD Maid als besonders erfolgreich bekannt ist. Wer die Gratis-Version ausgiebig getestet hat, und den Entwickler belohnen (oder einfach nur die Werbung loswerden) möchte: Mit gut drei Euro ist man dabei.

Selten genutzte Apps am automatischen Starten hindern

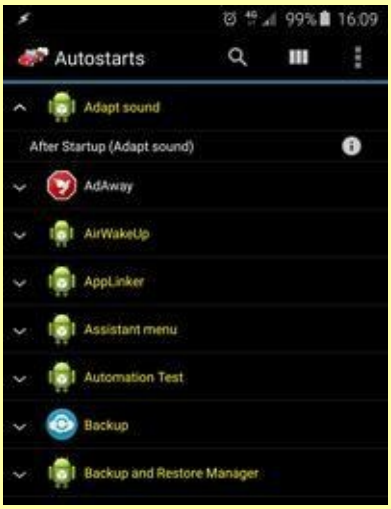
Ein altbekanntes Lied, nicht nur auf unseren geliebten Androiden: So manche App fühlt sich so furchtbar wichtig, dass sie meint, ständig laufen zu müssen. Am besten bereits vor dem Booten. Am schlimmsten sind dabei die ach-so-tollen Apps, mit denen uns Hersteller und Provider „zwangsbeglücken“ – und die „Otto Normalanwender“ nicht einmal beseitigen (sprich: deinstallieren) kann. Kaum ist der Androide gestartet, schon laufen auch Flickr, FM-Radio, Google Maps, Peep … ohne dass wir sie darum gebeten hätten. Belegen dabei Speicher, und verbraten CPU und Akku gleichermaßen.

Da möchte man gern einmal kräftig dazwischenschlagen. Wenn man diese Bösewichte nicht einfach deinstallieren kann (etwa weil es sich um System-Apps handelt und man keine root-Rechte hat, oder weil man die betreffende App doch hin und wieder einmal benötigt) – kann man diesen Wahnsinn nicht einfach abstellen? Bei mancher App findet sich ein betreffender Punkt in den Einstellungen. Ist dies jedoch nicht der Fall, sieht es ohne root recht mau aus.

Ein einfacher Task-Manager ist hier definitiv der falsche Ansatz. Allerdings gibt es einige Apps, mit denen sich das automatische Starten unterbinden lassen soll. Ohne root-Rechte sieht das aber eher so aus, dass die betroffenen Apps nach dem Auto-Start automatisch gekillt werden – und das geht leider all zu häufig in die Hose, da so manche App dann einfach wieder neu startet – was in einem Teufelskreis mit Akkufresser und CPU-Heizer enden kann. Es sei denn, man hat sich für Autorun Manager entschieden: Diese

App kann derartige Kreisläufe erkennen, und sieht sodann von einer weiteren Behandlung ab. Als Anwender sollte man dann ebenfalls davon Abstand nehmen, eine Behandlung erzwingen zu wollen.

Weitaus bessere Chancen hat man, sofern man mit root-Rechten ausgestattet ist – und zwar beispielsweise mit gerade genannter App. Diese ist im Erweiterten („Advanced“) Modus in der Lage, automatische App-Starts von vornherein zu unterbinden. Um automatisch gestartet zu werden, muss eine App nämlich bei ihrer Installation für die gewünschten Ereignisse einen sogenannten BroadcastReceiver registrieren – nur mit vollen Systemrechten (oder per Deinstallation der App) kann man diese Registrierung wieder auflösen (sofern die App dies nicht selbst ermöglicht). Genau das geschieht an dieser Stelle – und so hat man volle Kontrolle darüber, wann eine App automatisch starten darf. Nicht nur das „nach dem Booten starten“ lässt sich hier abschalten – sondern beispielsweise auch Dinge wie „bei eingehendem Anruf“, oder „nach hergestellter WLAN-Verbindung“, „USB angeschlossen“, und mehr.



Natürlich lassen sich einzelne „Receiver“ unabhängig voneinander deaktivieren. Und diese Aktion später ggf. wieder rückgängig machen, sollte dies notwendig werden. Zu beachten ist jedoch: Vor Deinstallation des *Autostart Manager* sollte man die ursprünglichen Einstellungen wieder herstellen – andernfalls ist dies, wie beschrieben, nur durch Neuinstallation der betroffenen Apps möglich, was sich bei System-Apps als zumindest schwierig darstellen dürfte.

Autorun Manager ist aus meiner Sicht in diesem Bereich die beste Wahl, zudem es auch noch gratis verfügbar ist. Als Alternative käme eventuell noch Autostarts in Frage, was nach dem gleichen Prinzip arbeitet (Deaktivieren von Receivern) – jedoch gut einen Euro kostet. Auch bei dieser App gilt es, vor der Deinstallation die ursprünglichen Einstellungen wieder herzustellen.

Jetzt sollte man aber nicht einfach wild alles abschalten, was einem vor die Flinte (oder besser: Unter die Finger) kommt – von System-Apps am besten ganz die Finger lassen (solange man nicht weiß, was man da tut). Verhält sich eine App sich anschließend irgendwie seltsam, kann man natürlich alles wieder rückgängig machen (oder die betroffene App einfach neu installieren). Aber warum gleich von vornherein für Probleme sorgen? Wenn man sich auf Dinge beschränkt, die relativ klar und einleuchtend sind, lassen sich selbige eher vermeiden. Besonders Ausschau halten sollte man nach folgenden Intents:

- *After Startup* (Nach dem Booten), aka *BOOT_COMPLETED*: Hier soll etwas nach dem Booten (also nach jedem Systemstart) geschehen
- *Connectivity Changed* (Netzwerkstatus geändert): Darauf lauscht u. a. der

LocationService von *Google Maps*, um die neuen Zellen zur Übertragung an Google im *LocationCache* abzulegen

Natürlich sollte man sich hier jeweils fragen, ob das nicht eventuell doch sinnvoll sein kann. In der Regel hat man jedoch bereits einen „Dauerbrenner auf dem Kieker“, der da ständig läuft und nuckelt – womit sich die Frage meist erübrigt.

Es gibt noch jede Menge weiterer Receiver, die sich hinterfragen lassen. Doch sollte dabei bedacht werden: Besser die Finger von Dingen lassen, die man nicht versteht. Schaltet man hier zu viel ab, funktioniert eine App u. U. nicht mehr richtig – und das ist sicher nicht Ziel der Übung.

Wer sich einfach nur einen Überblick verschaffen möchte, welche App sich für welches Ereignis im System registriert hat, kann dazu ebenfalls das gerade genannte *Autostarts* verwenden: Auch ohne root-Rechte lässt sich diese App installieren, arbeitet dann aber nur im reinen Lese-Modus.

Und bitte nicht wundern, wenn die nicht-root Apps scheinbar weniger automatisch startende Apps finden, als *Autostarts* im Lesemodus anzeigt: Das liegt daran, dass erstere nicht alle „Startrampen“ prüfen, sondern sich auf die gängigsten beschränken. Die volle Power bleibt also wieder einmal „Nerd Root“ vorbehalten…

Das Märchen vom Task-Killer

Viele „alte Hasen“ schwören darauf: Task-Killer seien das ultimative Mittel, um das System flüssig und den Strombedarf niedrig zu halten. Von genügend anderen Anwendern, die definitiv tiefgreifendes Wissen haben, hört man genau das Gegenteil – teilweise sogar bis zu dem Extrem, dass man „Task-Killer gar nicht mehr benötigt, da kümmert sich Android doch selbst drum“.

Da haben wir sie also nun: Zwei Statements, die verschiedener nicht sein könnten – beide von Leuten mit viel Erfahrung. Wem soll nun Glauben geschenkt werden? Da liegt schon fast auf der Hand, dass die Antwort irgendwo in der Mitte liegen muss, also beide Aussagen einen gewissen Wahrheitsgehalt haben.

Es ist durchaus richtig, dass Android sich selbst um den Speicher kümmert – und bei Bedarf nicht mehr benötigte Apps aus selbigem herauswirft. Dafür gibt es den OOM-Killer: Wird der Speicher knapp (ist das System also Out-Of-Memory), tritt er in Aktion. Aber nur dann. Läuft hingegen eine App Amok, und verwandelt die CPU in eine Warmhalteplatte für den Kaffeebecher, fühlt sich der OOM-Killer davon überhaupt nicht angesprochen – solange besagte Amok-App nicht zeitgleich sämtlichen Speicher auffrisst. Dies wäre also ein Fall für den „klassischen Task-Killer“.

Die Frage ist also weniger, *ob*, als vielmehr *wann* man einen Task-Killer einsetzen sollte:

- Falsch: „ich will den Speicher freiräumen“. Dafür ist der „OOM Killer“ zuständig (siehe RAM bereinigen)
- Falsch: „ich will Akku sparen“. Das geht mit einem Task-Killer i. d. R. eher nach hinten los.
- Richtig: „eine App hat sich aufgehängt, und blockiert [irgendwas]“. Hier ist der Task-Killer angesagt – weil bis der „OOM-Killer“ hier zuschlagen würde … Und ein Reboot ist nicht gerade die wünschenswerte Alternative.
- Richtig: „eine App läuft Amok“ (Panik-Mode: Man erwischt gerade eine App dabei, wie sie alle persönlichen Daten inkl. Nackt-Fotos auf eine berüchtigte Website hochlädt …). Oh ja: Abschießen! Oder gleich abschalten. Weil: Bis der OOM-Killer … genau, da ist es dann eh zu spät.
- Notfalls richtig: Kurz nach Betätigung des Start-Knopfes einer App (wann sonst?) fällt einem auf: Das würde jetzt zu lange laufen, weil [xxx]. Richtigerweise hätte man dann besser gar nicht erst den Knopf gedrückt. Lässt sich die App nicht anders vom „Weitermachen, bis der Arzt kommt“ abbringen, und belegt dabei wertvolle Ressourcen (CPU, Netztraffic), ist das „Abschießen“ mittels Task-Manager auch hier die einzige Alternative zum Reboot.

Die generelle Aussage, dass Task-Manager ein tolles Mittel zum Akku-Sparen seien, fängt jedoch definitiv mit „Es war einmal …“ an. Für Android-Versionen, die vor dem Punkt nur eine „1“ stehen haben, traf das vielleicht tatsächlich zu. Aktuell ist es jedoch wesentlich häufiger der Fall, dass eine frisch gekillte App binnen weniger Sekunden schon wieder aktiv ist – was bei „automatischen Task-Killern“ zu einem ewigen Kreislauf aus „Start-Kill-Start“ führen würde, und somit alles andere als ressourcensparend ist.

Längere Akkulaufzeiten erreichen

Kürzlich absolvierte ein Bekannter seinen zweiwöchigen Auslandsurlaub. Da er dort eine lokale SIM-Karte einsetzen wollte, um Roamingkosten zu vermeiden, benötigte er ein Reserve-Handy – um auch unter seiner heimischen Rufnummer erreichbar zu sein. Zu diesem Zweck wurde ein „Billig-Telefon“ der Marke Nokia hergenommen und aufgeladen. Das Ladegerät hätte er ruhig zu Hause liegen lassen können: Obwohl nur im Flugzeug abgeschaltet, und ansonsten permanent in Bereitschaft, hielt es mit der einen Akkuladung die vollen zwei Wochen durch, einschließlich einiger kürzerer Telefonate.

Zwei Wochen mit einer Akku-Ladung! So manch einer wäre schon froh, wenn er es auf zwei Tage bringen würde! Sind unsere Akkus schlechter geworden, oder woran liegt es?

Wohl eher daran, dass moderne Smartphones mittlerweile vollwertige Computer im Hosentaschen-Format sind. Was musste denn das oben genannte Nokia „leisten“? Kurz gesagt: „GSM Standby“, und ein paar GSM Gesprächsminuten. Mehr nicht. Unsere Smartphones sind hingegen permanent online, surfen am liebsten mit 3G oder gar LTE, und spielen parallel dazu Youtube-Videos ab. Wer nun einen kurzen Blick in die passende Übersicht im Anhang wirft, sieht den Unterschied deutlich: Würden wir unsere Androiden auch auf selbigen Standby reduzieren, und sämtliche „überflüssigen“ Dienste deaktivieren, sollten wir eigentlich ebenfalls die Wochenfrist überschreiten können.

In „voller Ausbaustufe“ ist das sicherlich wenig sinnvoll – man hat sich so ein Smartphone ja unter anderem wegen all dieser „Extras“ gekauft. Aber gibt es vielleicht Kompromisse? Schließlich fährt ja auch keiner seinen Porsche im fünften Gang mit 220 km/h in die Garage, nur weil er ihn für „schnelles Fahren“ gekauft hat. Ebenso wenig telefoniert man non-stop, und schaut auch nicht 24 Stunden pro Tag Youtube-Videos über das mobile Datennetz. Da fehlt dem Gerät wohl ein wenig Intelligenz: Es muss doch *fühlen*, wann wir etwas brauchen und wann nicht … Aber dann hieße es wohl „*die* Smartphone“, und nicht „*das*“. Also müssen wir dem kleinen Androiden ein wenig männliches Macho-Gehabe (meiner ist größer, schneller, weiter) ab- und weibliche Intuition angewöhnen. Möglichkeiten dazu sollte es doch geben. Und nein: Lippenstift, Lidschatten und Maskara helfen dabei nicht.

Ob da gerade etwas überdimensional Strom zieht, während der Androide beispielsweise eigentlich “ungenutzt” in der Tasche steckt, verrät u. a. die App DrainGuard. Sie teilt dem Anwender zwar nicht mit, wer der Schuldige ist – doch zumindest warnt sie bereits, wenn der Akku-Sauger am Werk ist, und nicht erst, wenn der Akku bereits leer gesaugt worden ist. Die notwendigen Werkzeuge zum

Aufspüren des Übeltäters finden sich im Kapitel System-Info.

Deaktivierung ungenutzter Dienste

Unschwer in der im Anhang befindlichen Übersicht erkennbar: Die Daten-Dienste gehören (nach der Video-Wiedergabe) zu den größten Verbrauchern. An dieser Stelle ließe sich also am besten sparen – sofern man auf etwas verzichten kann. Was das vielleicht sein könnte, ist wiederum von Anwender zu Anwender unterschiedlich: Als erstes gilt es daher, das eigene Nutzungsverhalten zu analysieren.

Wer über kein einziges Bluetooth-Gerät verfügt, kann Bluetooth komplett deaktivieren. Im Standby macht das zwar (nach GPS Standby) am wenigsten aus, aber auch Kleinvieh macht Mist. Das passende Häkchen findet sich in den System-Einstellungen unter *Drahtlos & Netzwerke*.

Auf der gleichen Seite findet sich mit *Mobilfunknetze* ein Untermenü, das ein Häkchen für *Nur 2G Netzwerke* anbietet – und somit 3G komplett abschaltet. Das bietet ein weit größeres Einsparpotential, sofern man kaum mobile Daten verwendet; für die „kleinen Dinge“ wie Mail und die Synchronisation der meisten Apps (insbesondere Kalender und Adressbuch) ist 2G definitiv ausreichend. Weniger interessant ist diese Möglichkeit hingegen für diejenigen, die viel mit dem Webbrowser oder der Youtube-App unterwegs sind, ohne dabei auf WLAN-Netze zurückgreifen zu können – da ein Umschalten über dieses Unter-Unter-Menü auf Dauer doch ein wenig umständlich ist. Und spätestens seit Android 2.3 (Gingerbread) ist ein direktes Umschalten per Widget ohne root leider ohnehin nicht mehr möglich.

Ähnliches gilt für GPS, welches aber glücklicherweise im Standby so gut wie gar keinen Strom verbraucht. Sobald allerdings eine App zur Standortbestimmung darauf zugreift (erkennbar am GPS-Symbol in der Statusleiste – siehe Beispiel-Symbole), macht sich auch das deutlich bemerkbar. Wird GPS vom Anwender selbst nur selten genutzt (sagen wir, maximal an einem Tag im Monat, oder nur im Urlaub), schaltet man dieses ebenfalls besser komplett aus – spätestens, wenn man eine dieser werbefinanzierten Apps beim Zugriff erwischt. Das passende Häkchen findet sich im Menü *Standortzugriff*.

Bleibt noch unser bester Freund: Das WLAN. Wer das gar nicht braucht … der gehört mit Sicherheit einer Minderheit an. Glücklicherweise kann man diesen Dienst nach Belieben mit diversen Schnellumschaltern schnellumschalten; bei aktuellen Android-Versionen ist der passende „Knopf“ überdies bereits in der Notification-Area integriert.

Ist zwar nicht direkt ein Dienst, passt aber mit in diese Reihe: Auch die Kamera ist ein besonderer Schluckspecht. Die Kamera-App sollte man daher besser nicht

nach Benutzung im Hintergrund weiterlaufen lassen, sondern beenden. Das gilt auch für diverse Barcode-Scanner, die ja auch die Kamera benutzen: Sollte sich ein solcher nicht richtig beenden wollen (was man meist am kontinuierlich starken Akku-Verbrauch erkennen kann), ist wohl ausnahmsweise einmal der Einsatz eines Task-Killers gerechtfertigt.

Display-Einstellungen

Diese bieten sich ebenfalls für deutliche potentielle Einsparungen an: Ein voll aufgedrehtes Display verbraucht wesentlich mehr Strom als ein 2G-Telefonat, während es auf niedrigster Stufe nur gut halb soviel benötigt wie besagtes Gespräch. Hier empfiehlt es sich daher, einen guten Kompromiss zu finden – zum Einen die Helligkeit, und zum Anderen das Timeout (nach welchem Zeitraum der Benutzer-Inaktivität das Display abgeschaltet werden soll) betreffend.

Beides lässt sich mit Bordmitteln im Menü *Display* konfigurieren. Bequemer geht auch dieses über diverse Schnellumschalter – besonders, wenn bei direkter Sonneneinstrahlung ohnehin kaum noch etwas auf dem Bildschirm zu erkennen ist. Die *Automatische Helligkeit* ist leider meist ein wenig zu hell (und damit stromhungrig); die „beste Einstellung“ ist jedoch sehr subjektiv, und muss daher individuell ermittelt werden.

Hintergrunddaten und Synchronisierung

Dieses Thema wurde ja bereits im Kapitel Datensynchronisation im Hintergrund angesprochen: So manche App möchte gern ständig Daten austauschen. Nicht immer ist dies aber nötig: Die wenigsten aktualisieren etwa ihre Kontaktliste mehrmals stündlich, sei es am Androiden oder am heimischen Bürorechner. Was nicht oder nicht ständig aktualisiert werden muss, kann man daher einfach im Menü Konten & Synchronisierung abschalten. Ein manueller Abgleich aus der betreffenden App bzw. genanntem Menü heraus ist in der Regel immer noch möglich.

Nicht jede App trägt sich jedoch an dieser Stelle ein. Bei den meisten lässt sich aber zumindest das Intervall konfigurieren (andernfalls ist es vielleicht ratsam, sich nach einer Alternative umzusehen, bei der dies der Fall ist). Für RSS-Reader reicht in den überwiegenden Fällen ein Intervall von einer Stunde oder höher (zum Einsparen des Datenvolumens kann man das darüber hinaus auch noch auf bestehende WLAN-Verbindungen einschränken).

In Sachen Mail empfiehlt sich, sofern möglich, die Verwendung von IMAP Push gegenüber einem ständigen „Pull“: Dabei fragt nicht der Client ständig nach „Gibt es neue Mail?“ (das wäre „Pull“ oder neudeutsch „Pollen“), sondern der Server sagt im Falle des Falles von sich aus Bescheid. Die Information kommt daher im Regelfall sogar früher. Auf den Energieverbrauch bezogen gilt diese Empfehlung

allerdings nur bedingt, da auch bei IMAP Push die Verbindung ab und an einen „Refresh“ benötigt, und darüber hinaus die Verbindung ständig aufrecht erhalten muss. Grob gesagt: Wer bei Eingang einer Mail sofort reagieren muss, ist damit gut dran. Genügt jedoch eine halbstündliche Aktualisierung, ist das altgewohnte „Pollen“ sparsamer.

Viele Apps berücksichtigen auch eine System-Variable, die kundtut, ob eine Datensynchronisation im Hintergrund zur Zeit erwünscht ist (die Android-API ist hier leider schwammig: Statt die Berücksichtigung verbindlich zu machen („must“), heißt es dort nur „sollte berücksichtigt werden“ („should“). Markanterweise gehören zu den Missachtern an erster Stelle Apps wie GMail – Google geht also wieder einmal mit „gutem Beispiel“ voran.

Für Apps, die diese System-Variable berücksichtigen, greifen dann wieder die kleinen Schnellumschalter, wie das abgebildete Power Toggles: In der Abbildung nicht enthalten wäre es der nach „Recycling“ aussehende Button, über den mit einmal Antippen die Synchronisation unterbunden (und natürlich zu einem beliebigen späteren Zeitpunkt wieder erlaubt) werden kann.

Weitere interessante Überlegungen zum Thema Hintergrunddaten finden sich bei SE-World – obwohl der Artikel bereits ein wenig älter ist, bleiben die Informationen interessant (und sind m. E. noch nicht veraltet oder überholt).

Verschiedenes

Dann wären da noch verschiedene Dinge, die so selbstverständlich sind, dass man gar nicht daran denkt. Etwa, dass man energiehungrige Apps nicht unbedingt im „Hintergrund“ weiterlaufen lässt – sondern sie beendet. Klar, eigentlich sollten sie im Hintergrund ja keine Ressourcen belegen; aber eigentlich gibt es das Wort „eigentlich“ auch nicht. Denken wir beispielsweise an eine Navigations-App: Diese aktualisiert i. d. R. auch im Hintergrund permanent den Standort – bei Verwendung von GPS nicht gerade eine Akku-schonende Tätigkeit. Auch so mancher Barcode-Scanner hält im Hintergrund die Kamera auf Trab. Wieder andere Apps pflegen im Hintergrund fleißigen Netzwerkverkehr, und so manche „Gratis-App“ lädt auch alle paar Sekunden ein neues Werbebanner, selbst wenn es niemand zu Gesicht bekommt. Ergo: Apps, die man nicht mit voller Absicht im Hintergrund behalten möchte (etwa, weil man sie gleich wieder zu verwenden gedenkt) nicht über die „Home-Taste“ (die mit dem Häuschen), sondern – sofern verfügbar – direkt über die „Beenden“-Funktion (häufig über die Menü-Taste erreichbar), oder zumindest über die „Zurück“-Taste verlassen. Bei hartnäckigen Verweigerern, die sich als Stromfresser herausstellen, ist dann ausnahmsweise einmal der Einsatz eines Task-Killers gerechtfertigt.

Bei Apps, die permanent im Hintergrund laufen wollen (und sich partout nicht beenden lassen), obwohl der Anwender das nicht benötigt, sollte man sich nach

Alternativen umsehen. Nicht ersetzbare Apps sind selten, häufig gibt es mehrere Apps, die das gleiche können. Guter Startpunkt für eine Suche ist die zur App gehörige Seite bei AppBrain: Dort sind in der Regel mögliche Alternativen direkt am Ende der jeweiligen App-Seite genannt. Auch die Große Sammlung der App-Übersichten nach Einsatzzweck kann sich als hilfreich erweisen. Da sollte sich doch etwas finden lassen! Sehr begrüßenswert wäre es allerdings, wenn sich der zuständige Entwickler auf Anfrage des Problems annimmt – und eine Lösung liefert.

Monitoring zeigt da einige Möglichkeiten auf. Auch ein Blick in das Kapitel Ausführlichere System-Infos fördert hilfreiche zu Tage.

Akku-Pflege

Nix mit „Waschen, Schneiden, Legen“, nein – Akkus mögen kein Wasser. Aber sie mögen es, wenn man sie gut behandelt. So kann die Beherzigung einiger kleiner Tipps nicht nur zu einer besseren Laufzeit im aktuellen Ladezyklus, sondern auch zu einer höheren Lebensdauer des kleinen Kraftwerks beitragen. Nichts kompliziertes, auch spezielles Zubehör braucht es nicht. Vielmehr geht es um das „Gewusst-Was“ und „Gewusst-Wie“. Da ist für jeden etwas dabei.

Zusätzliche Informationen (auch zum Umgang) lassen sich ebenfalls der Wikipedia entnehmen: Sowohl für LiIo-Akkus als auch für LiPo-Akkus. Eine kurzgefasste Aufstellung der wichtigsten Fakten findet sich auch bei Sinnvoll-Online.DE.

Temperatur

Aktuelle Androiden verwenden sogenannte Lithium-Ionen-Akkus (LiIo; einige setzen auch auf die Weiterentwicklung Lithium-Polymer (LiPo), wie Motorolas *Droid Razr* oder auch *CLIQ*; ebenso eine ganze Reihe Tablets: *Acer Iconia*, *Asus Transformer*, *Samsung Galaxy Tab*, u. a. m. – für diese Geräte gilt das folgende im Groben aber ebenso; platt gesagt, sind LiPo-Akkus kompakter, dafür aber auch empfindlicher was den Temperatur-Bereich betrifft). Diese mögen es nicht gern so warm: Also nicht in die Sonne legen (sie werden ohnehin nicht braun), auch die Hosentasche ist nicht der ideale Aufbewahrungsort. Bevor jemand fragt: Nein, der Kühlschrank auch nicht. Lieblings-Temperatur von LiIo-Akkus: Um die 20°C. Unterhalb von 10°C lässt die Leistungsfähigkeit nach, der „Arbeitsbereich“ wird meist mit 0..40°C angegeben.

Laden / Entladen

Eine vollständige Entladung bis zur Selbstabschaltung des Gerätes sollte zwar hin und wieder zur Kalibrierung stattfinden – aber nicht zu häufig: Sowohl LiIo als auch LiPo Akkus mögen eine vollständige Entladung nicht so gern. Idealerweise schließt man sie also wieder an das Ladekabel an, bevor der Ladestand die 50% Marke unterschreitet. Höhlen-Taucher wissen, wovon ich Rede, wenn ich hier jetzt den Begriff „Reserve“ in den Raum stelle: Natürlich kann man auch die „unteren 50%“ nutzen – gesünder (in diesem Fall für den Akku) ist es jedoch, darauf nicht angewiesen zu sein. Einen Memory-Effekt kennen weder LiIo noch LiPo, diesbezüglich ist daher nichts zu befürchten.

Den Akku kalibrieren

Der Ausdruck ist ziemlich irreführend. Genau genommen wird nicht etwa der Akku, sondern dessen Controller kalibriert. Letzterer lernt dabei nämlich, wie es um die tatsächliche Akku-Kapazität bestellt ist, und kann seine Statistiken entsprechend anpassen. Damit wird vermieden, dass das Gerät „plötzlich und unerwartet“ bei einem angeblichen Akku-Stand von über 30% „das Zeitliche segnet“. Oder aber die Anzeige schnell von 100% auf 1% fällt, um dann noch ein paar Stunden dort stehen zu bleiben – bevor der Akku wirklich leer ist.

Dies geschieht durch einen sogenannten „vollständigen Ladezyklus“. Bildlich gesprochen, wird der Akku dabei einmal von 100% auf 0% und wieder auf 100% gebracht – wobei während des gesamten Zyklus das Ladegerät exakt ein Mal mit dem Gerät verbunden wird (oder auch umgekehrt; auf jeden Fall eine volle Ladung ohne Unterbrechung).

Optimalerweise werden zu Beginn dieses Zyklus noch die Akku-Statistiken gelöscht, was in der Regel jedoch root-Rechte voraussetzt. Doch dieser Schritt scheint eher einem Mythos gezollt, wie Android-Entwicklerin Diane Hackborn laut XDA-Developers erklärt: In der `batterystats.bin` Datei werden laut ihrer Aussage keine Daten der Batterie, sondern ausschließlich die (auch unter *Einstellungen › Akku-Verbrauch* einsehbaren) Verbrauchsdaten der Apps gespeichert. Somit erübrigt sich eigentlich auch der Einsatz der folgenden Apps, mit deren Vorstellung ich dem Mythos dennoch meine Referenz erweisen möchte:

Hilfreich bei der Kalibrierung sind Tools wie das abgebildete Battery Calibration. Diese App führt den Anwender Schritt für Schritt durch den Prozess, und übernimmt nebenbei auch die Löschung der gespeicherten Akku-Statistiken aus dem Android-System.

Das Vorgehen ist dabei total einfach: Man startet die App, und steckt das Ladekabel an. Anschließend stellt man sicher, dass beide Buttons jeweils ein „grünes Lämpchen“ enthalten – womit die App angewiesen wird, bei Erreichen einer vollständigen Aufladung (100%) den Anwender mit Piepsen zu benachrichtigen, dass es weitergehen kann. Sobald dieser Zeitpunkt erreicht ist, betätigt man den großen Button in der Mitte (das löscht die `batterystats.bin` aus dem System), und startet den Androiden neu (womit die genannte Datei neu angelegt wird). Ladekabel raus, und einmal komplett leerlaufen lassen (mindestens bis 20%, besser weniger); anschließend Ladekabel wieder anschließen, und in einem Rutsch wieder vollständig aufladen. Fertig.

Wichtig ist, wie beschrieben, ein vollständiger Zyklus – da könnte man das also auch umgekehrt machen. So tut das etwa Battery Calibrator, und startet am „Nullpunkt des Seins“. 20% Ladung oder weniger, gerade genug, um die `batterystats.bin` Datei noch vor dem Durchstarten löschen zu können, werden als Anfang des Prozesses gewertet. Das Vorgehen ist ansonsten identisch: Die Datei wird gelöscht, das System neu gestartet, und in einem Zug einmal auf 100% und zurück gebracht.

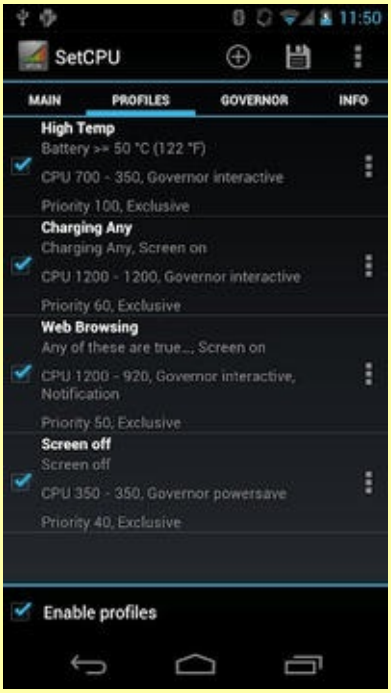
Wo man beginnt, bleibt dem Anwender überlassen – das Ergebnis sollte in beiden Fällen identisch sein: Eine korrekte Ladestands-Anzeige. Nur für das Löschen der Statistik-Datei benötigt es i. d. R. root, die genannten Apps vereinfachen lediglich den Prozess.

Bleibt die Frage: Wie oft sollte man dies tun, und wie sollte man vorgehen? Eine gute Zusammenfassung verlässlicher Quellen findet sich in einem Artikel bei StackExchange. Das Wichtigste kurz zusammengefasst: Keinesfalls öfter als einmal im Monat (alle drei Monate sollte völlig ausreichend sein) sollte man seinen Akku so weit als möglich entladen. Die 5% Marke sollte dabei auch definitiv nicht unterschritten werden. Anschließend das Gerät ohne Unterbrechung vollständig aufladen. Und immer daran denken: Vollständige Entladungen sind für einen Lithium-Ionen-Akku schädlich und verkürzen seine Lebensdauer, weshalb man eine „Kalibrierung“ so selten wie möglich durchführen sollte.

Wenn wir uns die Praxis anschauen: Oft genug kommt es vor, dass der Akku des Android-Gerätes nahezu vollständig entladen wird, ohne dass dies beabsichtigt war (Warum ist am Ende des Akkus noch so viel Tag übrig?). Hier sollte man anschließend sogleich eine „vollständige Aufladung ohne Unterbrechung“ durchführen, womit sich die Kalibrierung nebenbei erledigt hat.

Alternative Apps? Aber klar doch, wie immer in einer speziellen Übersicht.

CPU untertakten



Untertakten? Wollen nicht alle in die andere Richtung? Klar, das kann im Winter ganz praktisch sein: Zusätzlich 3G an, heftig Downloaden, HD-Videos schauen, und das Ladekabel dabei angeschlossen – diese Kombination ergibt einen wunderbaren Fingerwärmer. Der aber schnell erkaltet, sobald das Ladekabel abgezogen wird – weil der Akku das dann nicht lange mit macht.

Und jetzt bin ich dran mit der Frage: Wollten wir nicht in die andere Richtung, also den Akku zu möglichst langer Laufzeit animieren? Ein geringerer Prozessor-Takt kann dazu beitragen. Nicht umsonst hat man an 5-Kern-CPUs für Smartphones getüftelt: Vier Kerne mit ordentlich Power, und der fünfte mit geringerer Taktrate sowie weniger Power für die Hintergrundaktivitäten bei abgeschaltetem Bildschirm. Das bekommen wir softwareseitig sicher nicht so ganz hin – können uns dem aber zumindest annähern. Root vorausgesetzt.

Beispielsweise unter Zuhilfenahme des abgebildeten SetCPU und entsprechenden Profilen: Wann immer die volle Leistung nicht benötigt wird, kann die CPU gedrosselt werden. Etwa generell bei abgeschaltetem Display. Oder während unserer Nachtruhe. Empfehlenswert ebenfalls, sofern ein bestimmter Ladestand des Akkus unterschritten wird (damit man es noch zur nächsten Ladestation schafft, ohne dass das Gerät abschaltet).

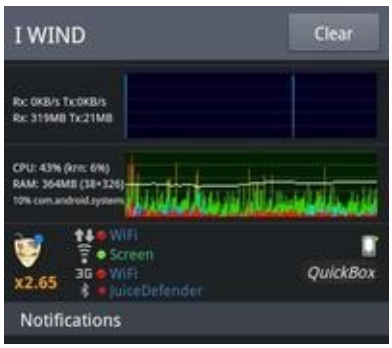
Eine Steuerung der CPU ist allerdings auch in diverse andere „Automaten-Apps“ integriert. Natürlich in Tasker – aber auch in einige der kleinen Helferlein, die im Folgenden kurz vorgestellt werden.

Kleine Helferlein

Menehune? Heinzelmännchen? Eine gewisse Ähnlichkeit lässt sich nicht leugnen: Die Apps, um die es hier geht, bekommt man während ihrer Arbeit nicht zu Gesicht. Sie arbeiten flink, und können wahre Meisterwerke vollbringen (wie beispielsweise, ein Smartphone mit einer einzigen Akkuladung über mehrere Tage am Laufen zu halten). Der Unterschied: Unsere Helferlein sind auch heute noch am Werkeln. Und ihre Erfolge lassen sich überprüfen und belegen. Außerdem haben sie bei *IzzyOnDroid* eine eigene Übersicht.

Apps für den „Handbetrieb“ wurden ja bereits bei den Schnellumschaltern

vorgestellt. Zeitschaltuhren, orts- und eventbasierte Schalter etc. gab es im Kapitel Automatisierung. Bleibt denn da überhaupt noch etwas übrig? Aber klar doch: Die App-Automaten, die sich speziell dem energiesparenden Thema gewidmet haben: Save Battery!



Der „SaftVerteidiger“ ist die erste App, die man in Foren zu diesem Thema genannt bekommt. Und sie ist sicher auch die umfangreichste und flexibelste App – zumindest in der Ultimate-Version. JuiceDefender (Bild: In der Notification-Area) kommt nämlich in drei Ausbaustufen daher: Einer kostenlosten, einer gehobenen (Plus), und einer Deluxe (Ultimate) Variante. Und natürlich unterscheiden sich die drei Versionen in ihrem Funktionsumfang, wie diese Übersicht auf der Homepage zeigt(e).

Alle drei Versionen bieten Widgets sowie Notifications, und darüber hinaus mindestens zwei Profile an (normal / aggressiv; extreme / customized kommen ab der Plus-Version dazu). Alle Versionen erlauben die Kontrolle der mobilen Daten (ab der Plus auch WLAN, und mit Ultimate zusätzlich Network Switching, AutoSync und Bluetooth). Die Hintergrund-Daten-Synchronisierung lässt sich mit allen drei Versionen steuern (spezielle nächtliche Schedules gibt es dann ab der Plus, und mit Ultimate zusätzlich Schedules für Peak-Hours und Wochenende). Dinge wie Display-Timeout und -Helligkeit sowie GPS-Kontrolle gibt es nur in der Ultimate, mit Ultimate und root lässt sich sogar die CPU steuern. Eine Sonderbehandlung einzelner Apps ist ab der Plus Version verfügbar.

Das waren jetzt ein Stapel Details. Abschließend noch einige Worte zur allgemeinen Funktionalität: Je nach Bedarf regelt *JuiceDefender* die entsprechenden Verbraucher. So lässt sich beispielsweise die Netzwerkverbindung „Stottern“ (d. h. in Intervallen aktivieren und deaktivieren; fürs mobile Netz geschieht dies durch Umbenennen des APN), wobei (ab der Plus-Version) ausgewählte Anwendungen auch permanente (oder gar keine) Aktivierung für ihre Übertragungen gewährt bekommen können. Wird ein Minimum an Ladezustand unterschritten, kann die App den Netzwerkverkehr auch komplett unterbinden. Ab der Plus-Version ist es überdies möglich, WLAN auch ortsbezogen zu aktivieren.

Bereits die Gratis-Version verspricht eine signifikante Laufzeitverlängerung, mit der Plus-Version sind für ca. zwei Euro die meisten Ansprüche abgedeckt. Für diejenigen, die ein Maximum herausholen möchten, gibt es schließlich die Ultimate-Version für ca. fünf Euro. In allen Fällen wird jedoch die Gratis-Version als „Unterbau“ benötigt: Plus und Ultimate sind lediglich AddOns.

Allerdings wird *JuiceDefender* offensichtlich nicht mehr gepflegt (die letzten Aktualisierungen stammen vom Januar 2012). Als Alternative zu wird häufig Green Power genannt – beide Apps sind auch durchaus vergleichbar. Die meisten Dinge werden von beiden Apps gleichermaßen unterstützt, wenn sich ihre Verfügbarkeit auch unterschiedlich auf die Gratis- und Bezahlversion(en) verteilt. So ermöglicht *Green Power* bereits in der kostenlosen Version, mobile und WLAN Daten basierend auf Schedule, Display Status, Ladegerät, Signal Level, und weiteren Kriterien zu steuern (Signal-Level heißt hier: Wird die Verbindung zu schwach, etwa in der U-Bahn, wird sie abgeschaltet – und in Intervallen geprüft, ob sich eine Aktivierung wieder lohnt). Dafür sind Widget und Nachtmodus erst in der Bezahlversion (für knapp vier Euro) verfügbar, ebenso wie Bluetooth-Schaltung. Hinzu kommt in der Premium-Version auch Unterstützung für Tasker – womit sich die fehlende ortsabhängige WLAN-Schaltung und CPU-Steuerung dann umsetzen ließe. Notfalls auch ohne *Green Power*.

Definitiv einen Blick wert ist auch Greenify, mit welchem sich akkuhungrige Hintergrund-Monster ruhigstellen lassen. Diese werden bei abgeschaltetem Display in den „Kälteschlaf“ versetzt – in welchem betroffene Tierchen bekannterweise zur „Überwinterung“ ihren Energiehaushalt auf ein Minimum reduzieren. Analog geschieht das Gleiche auch mit von *Greenify* in „Hibernation“ versetzte Apps – wobei sich (zumindest mit dem Donation-Package) auch Ausnahmen definieren lassen, sodass eine App etwa bei eingehender „Cloud-Message“ geweckt wird. Im Prinzip also ähnlich zum Deaktivieren von Apps, oder dem Einfrieren mit *Titanium Backup* – allerdings mit dem markanten Unterschied, dass Einfrieren und Auftauen mit *Greenify* automatisch und transparent erfolgt, ohne dass der User das jedes Mal manuell anstoßen müsste.

Selbstverständlich schickt *Greenify* nicht alles in die Truhe, sobald das Licht ausgeht. Welche Apps es dieser Behandlung unterziehen soll, legt der Anwender fest – wird dabei jedoch mit einer Auflistung unterstützt: Welche Apps laufen gerade im Hintergrund? Welche werden sich bei anstehenden Ereignissen an den Ressourcen vergreifen? Dies sollte beim „Fine-Tuning“ helfen. Die großen Fresser hat man meist ohnehin schon auf andere Weise aufgespürt.

Es ließen sich noch eine ganze Reihe weiterer Apps aufzählen, die einzelne Aspekte wie die automatische WLAN-Umschaltung u. a. m. abdecken. In der

eingangs genannten Übersicht sind sie aufgeführt – hier würden sie einfach den Umfang des Buches sprengen.

Platz im internen Speicher schaffen

Aufräumen

Irgendwie naheliegend: Wenn man etwas löscht, wird Platz frei. Besonders geeignete Kandidaten dafür sind Apps, die man ohnehin nicht mehr benötigt – was ja bereits in diesem Kapitel beschrieben wurde (genau wie für root-User das Säubern des Dalvik-Caches). Die Bereinigung von Cache, Chroniken & Co war ebenfalls bereits genannt.

Doch so manche App hinterlässt auch nach ihrer Deinstallation noch die eine oder andere Leiche im Keller. Um diese aufzuspüren und zu beseitigen, bietet sich ein Blick auf SD Maid an. Damit lassen sich gezielt Rückstände aufspüren und entfernen.

So ganz nebenbei kümmert sich *SD Maid* mittlerweile auch um die anderen gerade genannten Dinge: Installierte Apps lassen sich (zusammen mit dem von ihnen belegten Speicherplatz) auflisten, um unnötige Exemplare zu entsorgen. „Duplikate“ zeigt doppelte Dateien auf. Mit dem „Searcher“ lassen sich Dateien finden, deren Name auf ein bestimmtes Muster passt. Um Cache, Log-Dateien, und dergleichen mehr kümmert sich wiederum der „Systemreiniger“; auch ein Dateimanager ist mit an Bord und, root vorausgesetzt, lassen sich sogar „aufgeblasene Datenbanken“ bereinigen und komprimieren.

SD Maid scheint also für den Frühjahrsputz bestens geeignet. Ihr volles Potential kann diese App aber wieder erst mit root ausspielen, da sie nur dann auch auf das gesamte System Zugriff erhält.

Auslagern

Genügt das noch immer nicht, stellt man sich oft die Frage: Kann man Apps eigentlich auf der SD-Karte installieren? Oder dorthin auslagern? Ab Android 2.2 (Froyo) ist dies zumindest teilweise bereits von Haus aus möglich. Die Funktionalität nennt sich naheliegenderweise „App2SD“ – muss aber von der jeweiligen App explizit unterstützt werden (andernfalls lässt sich ein Verschieben zwar u. U. erzwingen, aber das kann zu unerwünschten Nebeneffekten führen). Ist dies der Fall, kann man sie aus dem Anwendungsmanager (*Einstellungen › Apps*) mit dem Button *Auf SD-Karte verschieben* auf die selbige befördern. Zumindest teilweise, denn ein gewisser „Grundstock“ verbleibt dabei im internen

Speicher. Unter anderem auch der zur App gehörende Dalvik- bzw. ART-Cache.

Zwar steht anschließend wieder mehr interner Speicher zur Verfügung – die Sache hat jedoch einen kleinen Haken. Sobald man den Androiden per USB-Kabel an den heimischen Computer anschließt, und die Karte an selbigen per UMS „freigibt“, steht sie lokal ja nicht mehr zur Verfügung: Vom Androiden aus ist ein Zugriff erst dann wieder möglich, wenn die USB-Verbindung getrennt wurde. Die mit App2SD auf die Karte verschobenen Apps können bei am Computer eingebundener SD-Karte also nicht ausgeführt werden. Dies führt insbesondere zu Problemen, wenn Widgets involviert sind – weshalb man Apps mit Widgets besser nicht auf der Karte installiert. Für die Freigabe per MTP gilt diese Einschränkung zwar nicht, jedoch gibt es mit Widgets dennoch ein Problem: Da die SD-Karte erst nach dem `BOOT_COMPLETED` Broadcast eingebunden wird, steht sie bei einem Gerätestart für den Homescreen u. U. zu spät zur Verfügung.

Lösungen zu diesen Problemen sind im Modding-Bereich unter Das Dateisystem linken beschrieben, und setzen root voraus.

Need For Speed

Der Wunsch nach „mehr Speed“ steht in der Regel dem Wunsch nach längeren Akku-Laufzeiten entgegen: Was mehr Leistung bringt, benötigt meist auch mehr Energie. Ausnahmen hiervon sind ein von unnötigen Altlasten befreiter Cache (siehe Cache bereinigen), die Deinstallation nicht mehr benötigter Apps einschließlich der Beseitigung etwaiger Rückstände (siehe Aufräumen, sowie genügend große Intervalle bei der Synchronisierung der Hintergrunddaten (siehe Hintergrunddaten und Synchronisierung) – diese Dinge eignen sich für beide Zwecke. Weiteren Punkten wollen wir uns im Folgenden widmen.

RAM bereinigen

Wer bei diesem Thema an Task-Killer denkt, liegt völlig daneben. Aber das erwähnte ich ja bereits bei den generellen Maßnahmen. Task-Killer sind dafür da, Amok-laufende Apps zu beenden, deren man anderweitig nicht habhaft werden kann. Um die Speicher-Bereinigung kümmert sich Android eigentlich selbst recht gut. Zum Verständnis an dieser Stelle ein kurzer Ausflug hinter die entsprechenden Kulissen:

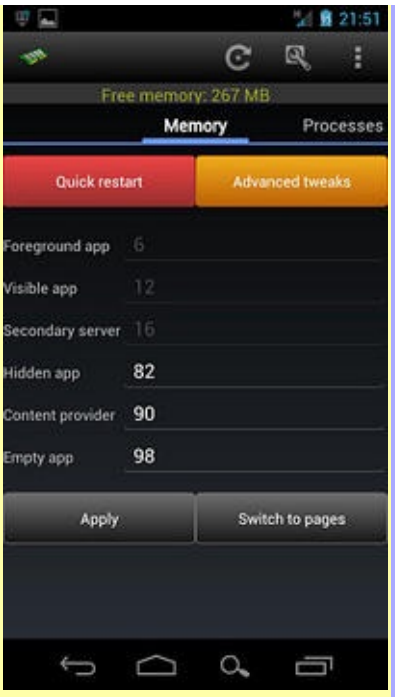
Der Grundgedanke ist, dass das Laden einer App aus dem internen Speicher bzw. gar von der SD-Karte naturgemäß (zusätzliche) Energie benötigt. Ist selbige hingegen bereits geladen, spart man sich den Ladevorgang. Also versucht Android, einmal geladene Apps im Speicher zu halten – auch, wenn der Anwender diese bereits wieder verlassen hat. Da nicht unbegrenzt RAM zur Verfügung steht, kommt dafür ein ausgeklügeltes System zum Einsatz.

Verantwortlich für das Aufräumen ist der sogenannte OOM-Killer (OOM steht für **O**ut **O**f **M**emory). Steht nicht mehr genügend freier Arbeitsspeicher zur Verfügung, ist der OOM-Killer dafür verantwortlich, solchen zu schaffen – indem er „irgend etwas“ aus dem RAM schmeißt. Für die Auswahl dieses „Verlierers“ werden verschiedene Kriterien verknüpft. Um diesen Vorgang verständlich zu halten, stelle ich das einmal vereinfacht dar:

Zunächst werden im RAM befindliche Prozesse in „Klassen“ eingeteilt. Hierbei erhalten ständig benötigte System-Prozesse eine sehr hohe Priorität. Auf der nächsten Stufe stehen „sichtbare“ Apps (also die, welche der Anwender gerade vor sich sieht, weil er damit arbeitet). Einige „Klassen“ weiter folgen am Ende dann die „leeren“ Apps – das sind die, die vom Benutzer „Beendet“ wurden, und die keine Tätigkeiten mehr ausführen. Naheliegenderweise wird beim Aufräumen mit der niedrigsten Priorität angefangen. Für die „engere Auswahl“ kommen dann noch weitere Kriterien ins Spiel, die wir an dieser Stelle nicht näher betrachten wollen.

Woher weiß der OOM-Killer nun, wann er tätig werden soll? Dazu sind für jede

„Klasse“ Schwellwerte im System konfiguriert. Ein Bild sagt mehr als tausend Worte, und so erklären sich diese Schwellwerte am besten anhand des Screenshots, der einen Bildschirm der App AutoKiller Memory Optimizer: Je niedriger die Priorität, desto mehr freies RAM muss verfügbar sein, damit die App geladen bleibt.



Unschwer zu erraten: Diese App ist meine Empfehlung für diesen Zweck. Als Alternative käme RAM Manager in Betracht, der sich gleich mit um das nächte Kapitel kümmert. Beide Apps benötigen root-Rechte – da die genannten Schwellwerte im System verankert sind; *AutoKiller* kann einige Einstellungen jedoch auch ohne root vornehmen.

Natürlich gilt es nach Installation der jeweiligen App, die richtigen Einstellungen zu finden. Diese sind zum Einen stark vom Gerät (und dem dort insgesamt vorhandenen RAM) abhängig – ist weniger RAM verfügbar, fallen auch die Grenzwerte geringer aus. Zum Anderen ist da auch noch ein subjektiver Faktor, der nicht verleugnet werden kann. Zum Herantasten bieten beide genannten Apps gute „Presets“ (Voreinstellungen) an. Es empfiehlt sich, zunächst mit „Moderate“ (so heißt das entsprechende Preset bei *AutoKiller* – bei *RAM Manager* wäre die Entsprechung „Balance“) zu beginnen, und sich von dort aus an die best geeigneten Werte heranzutasten.

Sind die richtigen Werte gefunden, und das System läuft damit flüssig und stabil? Dann weist man im letzten Schritt die ausgewählte App an, diese Werte nach jedem Neustart automatisch zu laden. Aber erst, wenn man sich dessen wirklich sicher ist! Bei zu aggressiven Einstellungen kann das andernfalls zu einem Dauer-Boot-Kreislauf führen, weil beispielsweise das System nach jedem Start gleich wieder „abgeschossen“ wird. Um dies im Ernstfall zu vermeiden, lässt sich zumindest *AutoKiller Memory Optimizer* so konfigurieren, dass die Einstellungen um 2 Minuten verzögert geladen werden – damit hat man im Ernstfall noch die Möglichkeit, einzugreifen. Nebenbei bringt diese App noch etliche weitere System-Tweaks mit, die sich optional aktivieren lassen. Details dazu finden sich auf der Projektsite.

CPU übertakten

Um das Untertakten der CPU haben wir uns bereits in einem früheren Abschnitt gekümmert – jetzt soll es also in die andere Richtung gehen. Aber auch das geht natürlich nur mit root-Rechten.

Die verfügbaren Mittelchen (auch Apps genannt) sind natürlich die gleichen

wie im genannten Abschnitt: SetCPU war ja bereits genannt, CPU-Tuner ist eine gängige Alternative (weitere Alternativen finden sich in dieser Übersicht).

Da es beispielsweise wenig Sinn macht, den Androiden im Leerlauf mit „vollem Speed“ laufen zu lassen (wenn man schläft, hat man daran in der Regel wenig Bedarf), bieten beide Apps auch „Timer“ an – sowie die Möglichkeit, unterschiedliche Taktraten für aktiviertes und deaktiviertes Display zu verwenden. Alternativ lassen sich, wie bereits beschrieben, die CPU-Einstellungen auch sehr granular mit Tasker vornehmen: So kann man es beispielsweise einrichten, dass beim Start bestimmter ausgewählter Apps die Taktrate der CPU entsprechend angepasst – und bei Beendigung/Verlassen derselben der Standard wieder hergestellt wird. Dies garantiert ein gesundes Mittelmaß zwischen Performance und Akku-Laufzeit – ohne dabei mittelmäßig zu sein.

Vorsicht ist natürlich angesichts der „Obergrenze“ (der höchstmöglichen Taktrate) geboten: Ab einem bestimmten Schwellwert wird das System instabil. Dann folgt der Kaffeewärmer, der Teekocher, und schließlich der Ziegelstein (Brick) – womit das Gerät dann nach Finnland kann. Dort wurde anscheinend die Sportart des Handy-Weitwurfs erfunden. Zumindest gibt es dort seit dem Jahr 2000 die entsprechende Weltmeisterschaft. Der Rekord lag seit 2005 bei gut 90 Metern, bis er 2012 mit 101,46 Metern überboten wurde.

Modding

In den Kapiteln des fünften Teils dieses Buches geht es um tiefe Eingriffe in das System unseres Androiden. Bevor man sich an dieses Thema wagt, sollte man sich schon recht gut auskennen – zu leicht kann man andernfalls Schaden anrichten, und der Androide wird zum Briefbeschwerer! Niemand soll sagen, ich hätte nicht gewarnt

Wie zu erwarten, wird bei all diesen Themen root benötigt. Da das in diesem Zusammenhang als Selbstverständlichkeit vorausgesetzt werden darf, werden die Abschnitte in diesem Teil des Buches nicht allesamt gesonders hervorgehoben.

Custom Recoveries

Einer der ersten Schritte (oftmals nach, manchmal aber auch vor dem Rooten des Gerätes) sollte die Installation eines Custom Recovery sein – und zwar auf jeden Fall *bevor* man sich an andere schwerwiegende System-Modifikationen heran wagt. Der Name lässt vielleicht bereits vermuten, warum dem so ist: Geht beim Modden etwas schief, ist der Recovery-Modus oft die letzte Rettung. Allerdings nur selten die Variante, die der Hersteller vorinstalliert hat – der fehlt es i. d. R. schlichtweg an grundlegenden Funktionalitäten. Nur wenige Punkte stehen hier zur Auswahl – insbesondere Dinge wie Nandroid Backup oder die Installation von Custom ROMs fehlen hier gänzlich. Ein Werksreset sowie das Einspielen von vom Hersteller/Netzanbieter offiziell angebotenen Updates ist jedoch meist möglich.

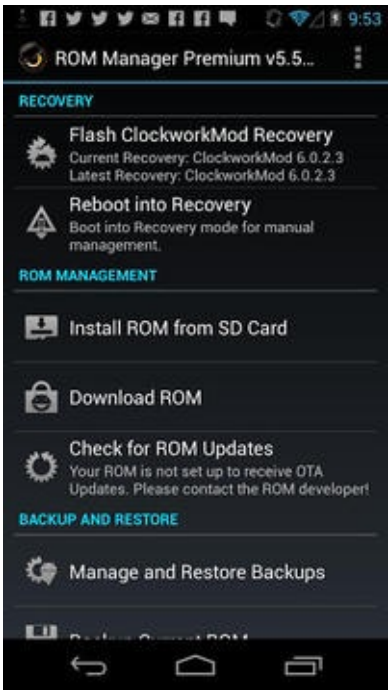
Im Funktionsumfang gleichen sich die gebräuchlichten Kandidaten stark, wie bereits an einem Vergleich der Screenshots ersichtlich ist. Zusätzlich zu den Möglichkeiten der vorinstallierten „Stock Recoveries“ bieten sie Dinge wie den Wipe des Dalvik bzw. ART Caches sowie einzelner Partitionen, das Formatieren letzterer, ein vollständiges Backup (auch Nandroid Backup genannt), sowie die Möglichkeit eines voll privilegierten Zugangs via ADB an. Darüber hinaus sind sie meist auch die Voraussetzung für die Installation eines Custom ROM: Stock Recoveries verweigern nämlich üblicherweise das Flashen von Archiven, die nicht vom Hersteller des Gerätes signiert wurden.

Custom Recoveries kommen im Wesentlichen in zwei Flavors: Touch basiert, und „non-touch“. Während die touch-basierten Recoveries – wie das Android-System selbst – über den Touchscreen bedient werden, steuert man die „klassische Variante“ über Hardware-Tasten: Die Lautstärke-Taste bewegt den Auswahl-Balken des Menüs nach oben bzw. unten, und mit der Power-Taste (auf einigen HTC-Geräten auch mit dem Trackball) bestätigt man die Auswahl. Diese Art von Bedienung erweist sich als großer Vorteil, wenn beispielsweise das Display beschädigt ist und nicht mehr auf Berührungen reagiert. Die Touchscreen-Steuerung ist dafür weit bequemer. Der bekannteste Kandidat für die „klassische Steuerung“ ist ClockworkMod – und für die Touchscreen-Steuerung TWRP. Beide sollen im Folgenden kurz vorgestellt werden.

ClockworkMod

Mit einem Custom Recovery – welches viele gleich mit dem bekanntesten Kandidaten, dem Recovery, gleichsetzen – wird das Android-Leben also wesentlich interessanter. CWM ist das verbreitetste Custom-Recovery, und für die größte Auswahl an Geräten auch verfügbar. Daher soll dieses als erstes näher betrachtet werden. Und zwar in seiner klassischen Variante – denn mittlerweile ist auch CWM als Touch-Recovery erhältlich.

ClockworkMod Recovery wurde von Koushik Dutta (in der Community besser bekannt als *Koush*) entwickelt. Eine komplette Tour durch das ganze Paket einschließlich Anleitungen zur Installation etc. finden sich in einem Complete Guide (Englisch) bei AddictiveTips. Obwohl dieser Guide laut der Webseite auf das Jahr 2011 datiert wird, scheint er derweil bereits mindestens einmal aktualisiert worden zu sein (unschwer an den Screenshots erkennbar). Aktueller und noch dazu in deutscher Sprache finden sich passende Artikel im DroidWiki sowie bei PocketPC.CH.



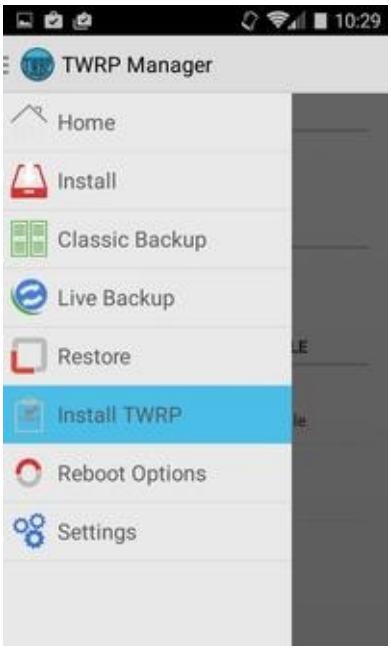
Besonders praktisch bei *ClockworkMod Recovery* ist, dass der Entwickler auch eine Companion-App namens ROM Manager bereitstellt, mit der sich viele Aufgaben auch bequem aus dem laufenden System anstoßen lassen. Wie der Screenshot zeigt, lässt sich mit dieser App auch bequem das Recovery-Image installieren. Ebenso findet man hiermit eine Auswahl von Custom-ROMs, die mit dem Gerät kompatibel sind, und kann diese direkt herunterladen und installieren. Ist bereits ein von *ROM Manager* unterstütztes Custom ROM installiert, findet man mit der App auch heraus, ob etwa Updates verfügbar sind – die sich natürlich dann ebenfalls herunterladen und installieren lassen.

Wer des Englischen mächtig genug ist, dem sei auch ein Video bei Youtube nahegelegt: ROM-Manager (Indepth Look) gibt eine Video-Tour durch die Möglichkeiten, die *ROM Manager* bietet – und erklärt auch die einzelnen Features. Will man hingegen CWM selbst zunächst einmal in „bewegten Bildern“ sehen fördert, oh Wunder, eine Suche nach „clockworkmod deutsch“ auf YouTube einige Videos zutage.

Wie kann man nun herausfinden, ob *ClockworkMod Recovery* bereits auf dem Gerät installiert ist? Hat man das Gerät gerade neu erworben, erübrigt sich diese Frage sicherlich: Ich habe noch nie gehört, dass ein Hersteller oder Netzbetreiber ein Custom Recovery vorinstalliert hat. Aber sei es drum: Wer ohnehin gern auf den *ROM Manager* zurückgreifen möchte, werfe noch einmal einen Blick auf den Screenshot. Ist nämlich *ClockworkMod Recovery* bereits auf dem Gerät installiert, wird die installierte Version hier auch gleich im ersten Punkt mit ausgewiesen. Alternativ kann man natürlich auch in den Recovery-Modus booten und schauen, ob die erweiterten Funktionalitäten angeboten werden – und die erste Zeile (wie im obigen Screenshot zu sehen) mit „ClockworkMod“ beginnt.

TWRP

In letzter Zeit ist auch TWRP verstärkt anzutreffen, und zieht mit *ClockworkMod* gleichauf. Diese Custom Recovery ist von Anfang an auf Touchscreen-Bedienung ausgelegt. Ein YouTube Video gibt einen guten Überblick über die Bedienung sowie die Möglichkeiten, die dieses Recovery bietet. Dazu gehören neben den Features, die sich TWRP mit CWM teilt, natürlich gewisse Vorzüge der Touch-Bedienung: So kann man etwa Backup-Dateien einfacher benennen, Dateilisten sortieren, und einiges mehr. Die Oberfläche sieht optisch auch ansprechender aus – obwohl das für ein Recovery nicht unbedingt das wichtigste Feature ist. Interessanter ist da sicher der integrierte Dateimanager – oder die Tatsache, dass TWRP ab Version 2.8 auch MTP direkt unterstützt. So lässt sich auch im Recovery-Modus bequem vom PC aus auf das Gerät zugreifen, um etwa bei Boot-Problemen schnell noch die Urlaubsbilder zu kopieren, bevor man sich an die Reparatur des Gerätes macht.



Ähnlich wie bei CWM, gibt es mit dem TWRP Manager auch für *TWRP* eine „Companion-App“ – allerdings ohne die Suche und Installation kompatibler ROMs. Mit dieser lassen sich ebenfalls Backups aus dem laufenden System heraus verwalten, oder nach Updates für das Recovery selbst suchen sowie selbige installieren. Einen guten Überblick über diese App verschafft eine Review bei YouTube – leider wiederum nur in englischer Sprache.

Nandroid Backups

Was ein Nandroid-Backup ist, haben die meisten Leser, die bis hierher durchgehalten haben, sicher bereits bei der Begriffserklärung „Nandroid“ nachgeschlagen. Doch wie gerade zuvor beim Thema „Custom Recovery“, sollen auch hier nähere Details folgen.

Zuerst einmal nehmen wir das Wort auseinander – denn wer hinter dem Begriff „Nandroid“ ein mit einem „N“ versehenes „Android“ vermutet, liegt leicht daneben. Meine Schuld, zugegeben: Ich hätte es deutlicher schreiben können. Dann sähe das Wort nämlich so aus: NANDroid. Und die ersten vier Buchstaben beziehen sich auf das, was gesichert werden soll: Der Inhalt des NAND-Flash, den meisten als „interner Speicher“, „Gerätespeicher“, oder auch „Telefon-Speicher“ bekannt. Wie man selbiges aus dem Recovery Menü heraus erstellt, wurde bereits im

entsprechenden Abschnitt zum Thema „Backup“ behandelt. Von dort ist auch bereits bekannt, dass dabei die kompletten Dateisysteme gesichert werden – ein solches Backup ist also definitiv ein „Komplett-Backup“. Ebenfalls bereits an betreffender Stelle geklärt wurde die Tatsache, dass sich ein Nandroid-Backup normalerweise nur wieder vollständig herstellen lässt (zumindest aus dem Recovery-Menü heraus); wer einzelne Daten benötigt, kann dazu jedoch auf spezielle Apps zurückgreifen.

Verwendet man zur Erstellung von Nandroid Backups gerade behandeltes *ClockworkMod Recovery*, landen die gesicherten Daten im Verzeichnis `clockworkmod/backup/` auf der SD-Karte – wo sie natürlich jede Menge Platz belegen: Schließlich handelt es sich bei jedem Backup je nach Modus entweder um die vollständigen Images oder die vollständigen Tar-Archive der internen Dateisysteme. Daher empfiehlt es sich, hier hin und wieder aufräumen: Während man auch aus Gründen der Datensicherheit eine Kopie dieser Backups auf einem anderen Gerät (z. B. dem heimischen Computer oder NAS) ablegen sollte, muss man auf der SD-Karte schließlich nicht alle Backup-Generationen aufheben. Löschen kann man ältere Backups auf verschiedene Weise:

- Mit einem Datei-Manager direkt auf dem Androiden
- Ebenfalls direkt auf dem Android-Gerät mit zuvor genanntem *ROM Manager* (unter „Manage Backups“) bzw. *TWRP Manager*
- Vom Computer aus, indem die SD-Karte dort eingebunden wird – oder man mit einem Remote-Manager über WLAN darauf zugreift

In welchem Format die erstellten Backups angelegt werden, hängt vom verwendeten Nandroid-Backup-Tool ab (es handelt sich in der Tat um ein separates Tool, welches lediglich in diverse Custom-Recoveries integriert wurde). So ist etwa *ClockworkMod* mit Version 5 dazu übergegangen, Tar-Archive zu erstellen. Welches Format man nun genau vor sich hat, lässt sich i. d. R. jedoch leicht am Dateinamen erkennen:

- `system.ext3.tar`: Eine Sicherung der Partition `/system`, deren Dateisystem `EXT3` ist, in einem Tar-Archiv
- `data.img`: Image-Datei der Partition `/data`, wobei das Dateisystem nicht angegeben ist (unter Dateisysteme wird beschrieben, welche Dateisysteme zum Einsatz kommen)

Und wie kommt man nun am Computer an die Inhalte eines solchen Backups heran – etwa zur Inspektion einzelner Dateien, oder Rettung einzelner Datensätze aus einer der enthaltenen Datenbanken? Bei `.tar` Dateien ist das klar: Auspacken. Aber wie schaut das etwa bei einem YAFFS2-Image aus? Für Linux-Anwender gibt es da das kleine Tool unyaffs, welches das komplette Image entpackt (und zwar in das Verzeichnis, in dem sich auch die Image-Datei befindet).

Auf genannter Website steht sowohl eine statisch kompilierte Binärdatei zur Verfügung – als auch die Möglichkeit, sich den Quellcode zum Selbst-Kompilieren herunterzuladen. Ich habe mir einmal erstere gegriffen: Sie lief problemlos auf meinen 32-Bit Ubuntu 8.04 – und läuft noch immer tadellos auf meinem 64-Bit Ubuntu 12.04 (mit installiertem `ia32-libs` Paket):

- `unyaffs` als root nach `/usr/local/bin` kopieren, damit es im `$PATH` liegt – und somit von überall aus verfügbar ist: `sudo cp unyaffs /usr/local/bin`
- Die zu entpackende Image-Datei in ein leeres Verzeichnis legen (da `unyaffs` scheinbar alles immer dort auspackt, wo diese Datei liegt – und nicht in das Verzeichnis, aus dem es aufgerufen wird): `mkdir mydata && cp data.img mydata/`
- In das Verzeichnis welchseln, und das Westpaket auspacken: `cd mydata && unyaffs data.img`
- Optional: Die Image-Datei wieder löschen (sofern man noch eine Kopie an anderer Stelle liegen hat): `rm -f data.img`
- Auf Forschungsreise gehen

Verwendet die in einer Image-Datei gesicherte Partition ein anderes Dateisystem (etwa ExtFS), lässt es sich auf dem PC auch mit dem an anderer Stelle in diesem Buch beschriebenen Programm *TestDisk* analysieren.

Zur Erforschung der zahlreichen Datenbank-Dateien, die überwiegend im SQLite-Format vorliegen, gibt es zahlreiche Front-Ends: An der Kommandozeile arbeitet etwa das Standard-Tool namens `sqlite3`, als grafisches Frontend für Linux, Mac und Windows gibt es auch SQLiteMan. Und interessante Datenbanken gibt es zur Genüge:

- SMS/MMS: `data/com.android.providers.telephony/databases/mmssms.db`
- MultiMedia Metadaten: `data/com.android.providers.media/databases/*.db`
- Kalenderdaten: `data/com.android.providers.calendar/databases/calendar.db`

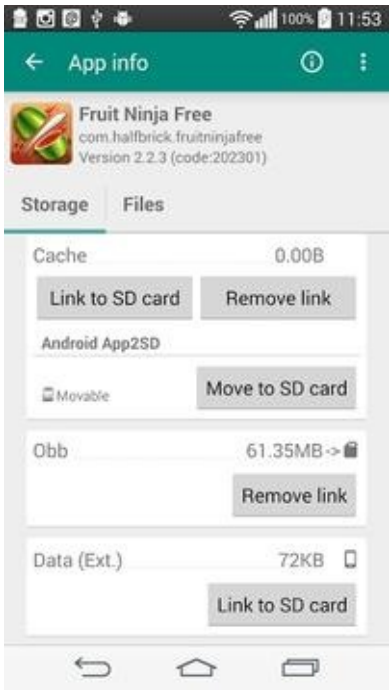
Und zahlreiche weitere. Viele Apps speichern ihre Daten auf diese Weise, und sie finden sich zumeist unter `data/<Paketname der App>/databases` wieder (der Paketname lässt sich der Google-Play-URL der App entnehmen – er ist dort mit dem Parameter `id=` angegeben). Fröhliches forschen!

Das Dateisystem linken

Lokale Dateisysteme

Bitte was? Können wir etwa das Dateisystem derart täuschen, dass sich unsere zwei Terabyte große Video-Sammlung auf der 32 GB SD-Karte unterbringen lässt? – Leider nicht. Ich muss mich für die nicht ganz eindeutige Überschrift entschuldigen: In den folgenden Abschnitten geht es vielmehr darum, das Android-System zu „täuschen“. Damit wir beispielsweise Daten auf der SD-Karte unterbringen können, obwohl Android selbige zwanghaft im internen Speicher haben will.

Der Hintergrund ist hier meist die Entlastung des oftmals knappen internen Speichers – zu schnell füllt sich dieser bis zum Rand, insbesondere wenn man ein paar umfangreichere Spiele installiert. App2SD hilft dabei nur begrenzt: So es denn überhaupt verfügbar ist, lagert es nur Teile der Apps aus (die das auch noch explizit unterstützten müssen), nicht jedoch deren Daten, Cache, etc.



Einige Custom-ROMs bieten daher App2SD+ an. Dies benötigt allerdings eine eigenständige (zusätzliche) Partition, die mit dem Dateisystem Ext3 versehen sein muss. Wird nun eine USB-Verbindung zum PC hergestellt, verbleibt diese Ext3-Partition beim Androiden – freigegeben wird lediglich die „herkömmliche“ FAT Partition. Somit stehen die ausgelagerten Apps weiterhin lokal zur Verfügung.

Ein weiterer Vorteil von App2SD+ ist, dass auch der Dalvik-Cache mit auf die Karte wandert: Es wird also deutlich mehr Platz freigeschaufelt. Als eingehendere Lektüre dazu empfiehlt sich dieser Artikel in der Brutzelstube.

Wer noch auf einer Android-Version vor Froyo festsitzt, oder kein App2SD+ unterstützendes ROM verwendet, findet in Link2SD eine Alternative. Auch diese Lösung benötigt eine separate Partition (es muss jedoch nicht zwingend Ext3 sein – FAT wird von *Link2SD* ebenfalls unterstützt), auf welche man sodann die Apps verschieben kann. Im Unterschied zu App2SD werden die Apps hier allerdings vollständig auf die Karte befördert, und im internen Speicher sodann durch einen passenden symbolischen Link ersetzt. Damit ist die Kompatibilität sichergestellt: Dem System wird so vorgegaukelt, dass sich die ausgelagerte App nach wie vor im internen Speicher befinden würde.

Wie bei App2SD+ gilt in diesem Falle: Über die USB-Verbindung wird die „App-Partition“ einfach nicht freigegeben, sodass Apps und ihre Widgets weiterhin lokal verfügbar bleiben. Und der Dalvik-Cache lässt sich mit *Link2SD* ebenfalls

auslagern, so man dies wünscht. Ebenso die Daten und weiteres „Zubehör“, wie der Screenshot zeigt.

Einen ähnlichen Ansatz verfolgen Directory Bind und, etwas aktueller und besser bewertet, FolderMount. Hier werden jedoch nicht die Apps selbst ausgelagert – es wird verzeichnisweise „verknüpft“. Dafür legt man Verzeichnis-Paare fest: Ein Verzeichnis auf der (externen) SD-Karte wird einem auf der internen SD-Karte (oder dem internen Speicher) zugeordnet. Die App *FolderMount* kümmert sich dann auf Wunsch sogar darum, initial die Daten zum neuen Speicherort zu verschieben. Wie bereits bei *Link2SD* wird dem Android-System dabei das Vorhandensein der entsprechenden Daten am Ursprungsort mittels symbolischer Links vorgetäuscht – was selbiges auch i. d. R. anstandslos akzeptiert. Ein Video-Tutorial bei YouTube veranschaulicht dies (in englischer Sprache): FolderMount macht den Prozess wirklich einfach. Der integrierte App-Analyzer zeigt einem sogar, wo die wirklich „großen Brocken“ zu finden sind.

Noch einen Schritt weiter geht External 2 Internal SD, und vertauscht gleich einmal die gesamte interne mit der externen SD-Karte. Hier ist allerdings besondere Vorsicht geboten, und zwar nicht nur beim Herausnehmen der Karte aus dem Gerät: Die App wurde explizit für das *Samsung Galaxy S3* geschrieben. Zwar scheint sie auch bei anderen Samsung-Geräten zu funktionieren, eine Gewähr gibt es jedoch nicht.

Remote-Laufwerke mounten

Sicher gibt es eine Reihe guter Dateimanager, mit denen man auf’s Netzwerk zugreifen kann. Doch spätestens, wenn es um das Abspielen von Titeln aus einer größeren Video-Sammlung geht, wird das mühsam: Warum erst jede Datei händisch auf den Androiden kopieren? Geht das nicht einfacher?

Die erste Antwort wäre wahrscheinlich „Streaming“ – was jedoch in der Regel eine passende Streaming-Server-Software auf der Gegenseite voraussetzt. Viel praktischer wäre es doch, man hätte das entfernte Laufwerk lokal zur Verfügung!

Exakt das versprechen die Apps aus dieser Übersicht zu leisten. Leider wird die Liste dort immer kürzer: Früher einmal verfügbare Apps wie *CifsManager* (für Samba) und *Mount Manager* (für Samba und NFS) werden nicht mehr gepflegt, und sind aus dem Playstore verschwunden; neue Apps sind leider auch nicht aufgetaucht. Geblieben sind damit derzeit nur zwei Kandidaten: CIFS Proxy Service (das laut Beschreibung von nutzenden Apps explizit unterstützt werden muss) sowie SSHFSAndroid (zur Einbindung von Laufwerken über SSH.

Eine Alternative dazu ist, das Ganze selbst mit einem Skript zu erledigen. Das funktioniert mittels *Busybox* und einem Kernel, der NTFS unterstützt (entweder

direkt, oder über ein entsprechendes Modul). Das verlinkte Skript ist für NFS gedacht – sollte sich jedoch auch für Samba anpassen lassen.

Swap-Space nutzen

Unter Windows gibt es hierfür die „Auslagerungs-Datei“, unter Linux richtet man sich am besten eine Swap-Partition ein oder greift, falls man dies bei der Installation vergessen hat und es schnell gehen muss, ebenfalls zu einer Swap-Datei. Was aber hat es mit diesem „Swappen“ oder „Auslagern“ auf sich?

Wird der Arbeitsspeicher (das RAM) knapp, muss Platz geschaffen werden. Im vorigen Abschnitt habe ich erläutert, in welcher Form sich der OOM-Killer um diese Angelegenheit kümmert. Wurde eine App auf diese Weise aus dem RAM entfernt, müssen bei einem Neustart derselben sämtliche benötigte Speicher-Strukturen neu aufgebaut werden. Beim Swappen hingegen wird ein Abbild des entsprechenden Speicherbereiches auf den Datenträger ausgelagert – ein Neuladen geht auf diese Weise wesentlich schneller und energiesparender vonstatten.

Da böte sich doch eigentlich an, dass dies von Haus aus bereits eingerichtet ist – wo also ist der Haken? Am Datenträger hängt er. Während bei Desktop-Rechnern „normale Festplatten“ zum Einsatz kommen, verwenden unsere Androiden fast ausnahmslos sogenannten Flash-Speicher. Und der ist hinsichtlich der Schreibzyklen ein wenig empfindlich: Zu viele davon verträgt er nicht. Wer also permanent auf dem gleichen Bereich schreibt (und das ist beim Swapping kaum auszuschließen), verringert damit die Lebensdauer des Datenträgers. Das ist für den Einen „schlimm“ – der Andere hingegen „pfeift darauf“, und kauft sich halt öfter einmal eine neue SD-Karte.

Wer Swap nutzen möchte, hat dafür zwei Möglichkeiten: Das Verwenden einer Swap-Partition (beispielsweise mit AParted), oder einer Swap-Datei (etwa mit im vorigen Punkt erwähntem *RAM Manager*).

Im Hinterkopf behalten sollte man dabei auch die begrenzten Schreibzyklen der verwendeten Speichermedien, da Swap doch recht „schreibintensiv“ sein kann. Insgesamt stehen daher folgende Kriterien zur Erwägung:

- Wieviel Swap wird benötigt? Nicht übertreiben, besser „von unten“ herantasten – angefangen mit ca. 10…20% der RAM-Größe.
- Wo soll der „Swap-Space“ erstellt werden? Der interne Speicher ist meist schneller – lässt sich aber nach Verschleiß schlecht austauschen.
- In welcher Form soll der „Swap-Space“ erstellt werden? Eine Swap Partition verspricht (geringfügig) mehr Performance, dafür wird jedoch auch immer an der gleichen Stelle geschrieben (schnellerer Verschleiß). Eine Swap-Datei ist

daher meist zu bevorzugen – und sollte am Besten auch bei jeder Aktivierung (i. d. R. also nach jedem Boot) neu erzeugt werden. Damit kann sich der Controller besser um eine „gleichmäßigere Schreib-Verteilung“ kümmern.
- Wie groß soll die „Swappiness“ sein? Klar größer als 0 (sonst wird gar nicht ausgelagert) – aber definitiv kleiner als 100 (sonst wird aggressiv ausgelagert, was wiederum zu erhöhter Abnutzung des Speichermediums führt). Zu empfehlen sind Werte zwischen 10 und 20 – von da aus kann man sich langsam an ein „Optimum“ herantasten.

Eine passende Übersicht für alternative Apps habe ich natürlich auch für dieses Thema parat.

Das Xposed-Framework

Es muss nicht gleich ein Custom ROM sein. Auch mit dem Xposed-Framework lässt sich so einiges anstellen. Dabei schließt das Eine das Andere nicht aus: Mit einem Custom ROM *und* dem Xposed-Framework … Aber eins nach dem anderen.

Was ist Xposed?

Xposed ist ein Framework für Module, mit dem sich das Android-System sowie darauf installierte Apps anpassen lassen, ohne dass man dafür die zugehörigen APK Dateien selbst anfasst. Stattdessen greift das Framework tief ins System ein, und modifiziert die entsprechenden DEX bzw. ART Repräsentation der jeweiligen App. Um dies zu ermöglichen, muss natürlich Xposed selbst entsprechend tief in das Android-System integriert sein – wofür bei der Installation des Frameworks gesorgt wird, indem einige System-Komponenten (nachdem sie für eine etwaige Wiederherstellung gesichert wurden) ersetzt werden.

Dieser Ansatz bietet verschiedene Vorteile. Einige davon sind:

- Es lassen sich Teile des Systems anpassen, auf die man normalerweise keinen Zugriff hätte.
- Mehrere verschiedene, voneinander unabhängige Modifikationen der selben App können zeitgleich angewendet werden (anders als bei der Installation einer „gepatchten Version“ dieser App, bei der man sich für eine Variante entscheiden muss).
- Änderungen können leicht rückgängig gemacht werden: Da diese generell dynamisch beim Systemstart angewendet werden, muss man lediglich die entsprechenden Module deaktivieren und das System neu starten.

Wie installiert man das Framework?

Das wichtigste zuerst: Voraussetzung ist ein gerootetes Gerät. Am Besten auch ein aktuelles Nandroid-Backup – man kann nie wissen. Ist diese erfüllt, kann es losgehen:

Android 4.x

Unter Android 4.x ist die Installation recht einfach: man lädt sich die APK-Datei des Installers herunter, installiert sie (beispielsweise per `adb install`), und started die App. Daraufhin begrüßt einen ein Bildschirm wie der abgebildete. Hier tippt man den obersten Eintrag an, der mit „Framework“ beschriftet ist.

Hier wird das Framework selbst verwaltet. Wie die Schaltflächen korrekt suggerieren, kann man es hier installieren, aktualisieren, und im Bedarfsfall auch wieder entfernen. Der Installations-Modus kann festgelegt (`/system` direkt beschreiben, eine `.zip` Datei erstellen und automatisch per Recovery installieren, oder auch die erstellte `.zip` Datei für die manuelle Installation per Recovery auf der SD-Karte bereitstellen), und das Gerät neu gestartet werden. Dieser letzte Schritt ist

immer jeweils nach Installation/Aktualisierung des Frameworks oder seiner Module notwendig; wie bereits erwähnt, erfolgen die jeweiligen Anpassungen ja ausschließlich bei Systemstart.

Nach erfolgter Installation muss, wie beschrieben, das Gerät einmal neu gestartet werden. Dabei hat man die Wahl zwischen einem „Software-Neustart“ und einem „Geräte-Neustart“. Ersteres ist schneller (das Gerät wird nicht komplett heruntergefahren) und reicht in den meisten Fällen (zumindest bei Modul-Änderungen) aus – letzteres ist gründlicher (vollständiger Shutdown und Neustart). Der nächste Boot-Vorgang dauert nun ein ganzes Stück länger als gewohnt – schließlich muss sich Xposed nun mit dem System „verschmelzen“. Das kann u. U. mehrere Minuten brauchen, also nicht nervös werden. Das Anschließen des Ladekabels ist sicher kein schlechter Gedanke – insbesondere, wenn der Akku-Stand schon ein wenig niedrig ist.

Android ≥ 5

Beginnend mit Lollipop, wird die Sache ein wenig komplexer. Hier kommt statt Dalvik die neue ART zum Einsatz – was Xposed zusätzlichen Aufwand bei der Integration des Frameworks abverlangt. Wie im offiziellen Thread für Lollipop/Marshmallow beschrieben, muss zunächst die zur Architektur des Gerätes passende `.zip` Datei (`xposed-v*-sdk*-*.zip`) von dort heruntergeladen und per (Custom) Recovery geflasht werden (während man dort ist, lädt man sich sicherheitshalber auch gleich das Päckchen mit herunter, welches die Angelegenheit im Notfall wieder rückgängig machen kann: `xposed-uninstaller*.zip`. Ist das erledigt, fehlt noch das ebenfalls im ersten Post des Threads verlinkte `XposedInstaller_3.0*.apk` – mit dem man verfährt,

wie oben für Android 4.x beschrieben.

Und wie wird man es im Bedarfsfall wieder los?

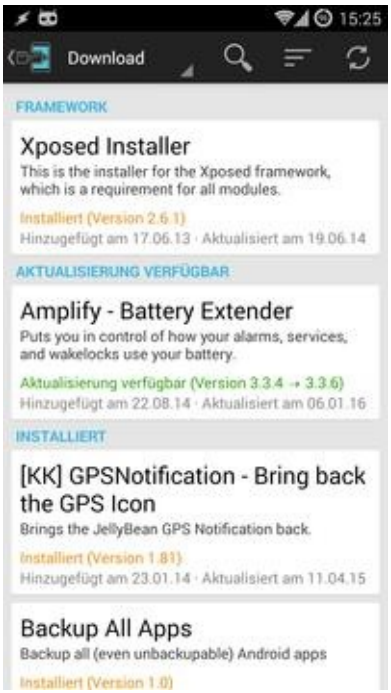
Eine Variante dafür war bereits weiter oben zu erkennen: Solange nichts „kaputt gegangen“ ist, lässt sich dies über den entsprechenden Menüpunkt im Xposed Installer erledigen: In der Framework Sektion gibt es schließlich eine mit „Deinstallieren“ beschriftete Option. Hat man noch Module installiert schadet es sicher nichts, diese zuvor noch separat zu entfernen: Ist das Framework weg, funktionieren sie ohnehin nicht mehr.

Was aber, wenn etwas schief gegangen ist, und das Gerät nicht mehr startet? Für diesen Fall gibt es eine `.zip` Datei, welche man per Recovery flashen kann. Unter Android 4.x wurde sie bei der Installation erzeugt und auf der SD-Karte abgelegt (der Installer hat den entsprechenden Speicherort sodann kundgetan). Ab Android 5 gab es den Uninstaller an der gleichen Stelle zum herunterladen, wo es auch den Installer gab (siehe dort) – auch diesen gilt es dann, per Recovery einzuspielen.

Wie benutzt man Xposed – und Was sind Module?

Unmittelbar nach der Installation des Frameworks wird man – abgesehen davon, dass der Boot-Vorgang länger als gewöhnlich gedauert hat – keine Veränderungen feststellen. Das ist jedoch völlig normal: Das Framework selbst bietet lediglich das Fundament, auf dem die Module aufbauen. Es stellt Schnittstellen zum System bereit, welche sich von ihnen nutzen lassen. Um wirklich Veränderungen zu sehen, benötigt man entsprechende Module. Einige davon findet man im Google Play Store (sowie in einigen anderen App-Märkten); die meisten jedoch sind im Xposed Repository verfügbar. Einige stelle ich im Anschluss kurz vor. Doch zuvor noch ein paar generelle Hintergrund-Informationen, die auf alle Module zutreffen.

Installierte Module werden in der gleichnamigen Sektion des Installers verwaltet, wie im rechten Screenshot zu sehen. Unmittelbar nach der installation des eigentlichen Frameworks bleibt diese Seite vermutlich leer: Hat man noch keine Module installiert, gibt es hier schließlich auch noch nichts zu verwalten. Allerdings hat der Installer den Zugriff auf sein Repository in die App integriert: Wechselt man zur Download Sektion, lassen sich verfügbare Module auflisten, installieren, und auch aktualisieren. Wie der Screenshot links zeigt, wird man über Updates installierter Module sogar informiert (leider nur hier; keine



Benachrichtigung etwa in der Notification Bar, da (zumindest derzeit) keine automatischen Updates implementiert zu sein scheinen). Wichtiger Hinweis: Einige Module zielen auf spezielle Android-Versionen ab, oder funktionieren nur mit bestimmten ROMs. Man sollte also auf jeden Fall die Beschreibung genau lesen, bevor man ein Modul installiert.

Nach der Aktualisierung eines (oder mehrerer) Module(s) muss das Gerät neu gestartet werden (nach meiner Erfahrung genügt dafür in den meisten Fällen ein „Software-Neustart“). Hat man neue Module installiert, müssen diese zuvor in der Module Sektion aktiviert werden. Dies ist eine Sicherheitsvorkehrung, damit bösartige Apps einem nicht einfach Module „unterjubeln“ können: Selbst wenn sie Apps (und somit auch Xposed Module) installieren können, werden diese somit nicht automatisch aktiv.

In der Sektion *Module* öffnet ein langes Drücken auf einen Modul-Eintrag ein Kontext-Menü, aus welchem man auf Updates prüfen, App-Details aufrufen, oder auch das entsprechende Modul wieder deinstallieren kann. Ein kurzes Antippen hingegen führt in die GUI (i. d. R. spezielle Einstellungen) des Modules, so es darüber verfügt. Bietet das Modul keine solche Option, informiert eine kurz eingeblendete Toast-Nachricht über diesen Fakt.

Ein weiterer genereller Hinweis: Für weitere Informationen zu Framework und Modulen, sowie im Fall von Problemen, ist der Xposed Bereich im XDA-Forum eine gute Anlaufstelle. Hier werden Neuigkeiten kundgetan (etwa die Verfügbarkeit neuer Versionen oder die Unterstützung neuerer Android-Versionen), nützliche Informationen gesammelt, Module vorgestellt – und man kann sich über etwaige Probleme austauschen. Vor dem Posten einer Frage sollte man jedoch nachschauen, ob nicht jemand anderes damit schneller war – und sich u. U. sogar schon eine Lösung/Antwort findet.

Einige Module kurz vorgestellt

Abschließend möchte ich noch einige der Module kurz vorstellen, die ich benutze und schätzen gelernt habe:

- **Backup All Apps:** Entwickler können festlegen, dass ihre Apps von ADB Backups ausgeschlossen werden. Dieses Modul „annulliert“ eine etwaige solche Einstellung. Eines der wichtigsten Module für mich.
- **DynamicAlarmIcon:** Es ist schön, wenn ein Icon in der Notification Area auf einen anstehenden Alarm hinweist (das tut es). Viel schöner und praktischer ist es jedoch, wenn dieses Icon auch die Weck-Zeit ausweist (dafür sorgt

dieses Modul).

- **Bring back the GPS Icon:** Ein weiteres kleines Goodie. Ich möchte gern sehen, wenn eine App meinen Standort abfragt. Oder wann generell noch die Satelliten gesucht werden bzw. ein „Fix“ steht.
- **FakeGApps:** Es ist vielleicht bekannt, dass ich „Android without Google“ nutze – mittels microG. Dieses Modul hilft mir, störrische Apps vom Gegenteil zu überzeugen – und so zum Laufen zu bewegen.
- **Amplify:** Hält Apps davon ab, das System unnötig lange wach zu halten – was den Akku länger durchhalten lässt. Unterstützt Tasker – und bietet (speziell mit der Donation-Version) eine flexible Konfiguration (auch wenn die Voreinstellungen ganz OK sind).
- **XPrivacy:** Berechtigungen/Privacy-Einstellungen von Apps verwalten (um persönliche Daten zu schützen). Sehr granular. Hat wahrscheinlich jeder bereits einmal erwähnt gehört.

Einige andere Apps bringen ihre eigenen Module gleich mit, wie beispielsweise Secure Settings. Dann gibt es noch eine ganze Menge weiterer Module, die ich hier nicht alle vorstellen oder auch nur kurz benennen kann: Allein das Xposed Repo listet derzeit fast 800 Module. Da gibt es welche für Multi-Window, Pie Control, und vieles mehr. Wer nach Quellen für „Best-of-Listings“ sucht (welche natürlich immer subjektiv sind), ist herzlich eingeladen, mit den Links am Ende meiner Xposed Framework Materialsammlung zu beginnen.

Custom ROMs

Wem „einfache“ Modifikationen nicht weit genug gehen – oder wer sich „aus Googles Klauen befreien“ möchte, der greift zu einem Custom ROM. Aus dem Namen ließe sich bereits zweierlei schließen: Dass sie angepasst (aus dem Englischen „to customize“ – was nicht direkt „kostümieren“ heißt, auch wenn das oftmals ein Nebeneffekt ist), und/oder auf den Kunden (Englisch: „customer“) zugeschnitten wurden. Beides ist in gewisser Weise richtig.

Zu unterscheiden sind hier im wesentlichen zwei Gruppen: Die einen basieren auf dem AOSP-ROM (und pflegen die gerätespezifischen Treiber-Anpassungen selbst nach) – die anderen nehmen eher das Stock-ROM des Herstellers als Grundlage. Bekanntester Vertreter ersterer Gruppe ist sicher CyanogenMod (das gleich noch separat behandelt wird). Für letztere Gruppe gibt es dann fast so viele ROMs wie Geräte.

Was zeichnet nun ein Custom-ROM aus? Meist kommt es bereits in gerooteter Form daher. Und oftmals läuft es stabiler und performanter als jegliches für das gleiche Gerät verfügbare Stock-ROM – da es besser optimiert und an die Hardware angepasst wurde. Doch auch die Bloatware spielt hier eine Rolle: Die meisten *AOSP* basierten ROMs lassen diese nämlich weg, einschließlich der Hersteller-spezifischen grafischen Aufsätze. Dass dann weniger ungenutzte Apps und Dienste ständig im Hintergrund mitlaufen, kommt natürlich der Performance zugute. Stattdessen findet dafür die eine oder andere wirklich nützliche App ihren Weg in das ROM – wie etwa bereits zuvor genannter ROM-Manager.

Bei den Stock-ROM basierten Kandidaten bleibt hingegen die Bloatware in der Regel drauf, die Hersteller-spezifische Oberfläche ohnehin. Worin dann hier die Vorteile liegen? In Optimierungen und Anpassungen, sowie hin und wieder auch aktuelleren Android-Versionen. So kann es durchaus passieren, dass ein Hersteller für ein Gerät keine neueren Android-Versionen mehr bereitstellt – für ein anderes, relativ baugleiches jedoch schon. Hier genügen dann oftmals geringfügige Anpassungen, um auch das „alte Eisen“ wieder mit „jüngerem Gemüse“ zu bestücken.

Und wie steht es um die Update-Geschwindigkeit? *AOSP*-basierte Custom-ROMs bieten fast ausnahmslos neuere Android-Versionen weit früher an als die Geräte-Hersteller oder gar Netzanbieter (sofern die Geräte überhaupt noch „offizielle Updates“ erhalten). Bei den anderen gibt es dann oftmals noch Updates für Geräte, welche die Hersteller schon längst abgeschrieben haben (für die es also gar keine „offiziellen“ Updates mehr gibt); auf diese Weise läuft beispielsweise mein von Motorola nur bis Android 2.2 unterstütztes *Milestone 2* derzeit mit Android 4.4.4, und wird mein offiziell bis Android 4.1 supportetes *LG P880* in Kürze von 4.4.4 auf 5.1 aktualisiert.

Klingt gar nicht so verkehrt – aber wo bekommt man so ein Custom-ROM nun

her? Die meisten finden sich sicher bei den XDA Developers. Wobei „finden“ dabei das eigentliche Problem darstellt: Das richtige ROM für ein bestimmtes Gerät aufzuspüren, ist gerade bei einem derart großen und aktiven Forum nicht sonderlich einfach. Wie gut, dass es auch dafür spezielle Anlaufstellen gibt:

- PDADB: Lange Liste (weit über 800 Seiten) für Geräte aller Couleur – allerdings einschließlich Blackberry, Symbian, und so weiter. Keine Filter-Möglichkeit, daher recht unhandlich
- TheUnlockr: Gruppiert nach Herstellern, und sodann nach Gerät. Scheint sowohl recht aktuell als auch recht vollständig zu sein – m. E. die beste Online-Anlaufstelle!
- Man nutzt einfach bereits genannten *ROM Manager*, der die ihm bekannten mit dem Gerät kompatiblen ROMs anzeigt, auf Wunsch herunterlädt, und auch gleich installiert – sicher die bequemste Variante
- Man schaut bei Android.StackExchange.Com nach weiteren alternativen Quellen

100% vollständig ist sicher keine der genannten (oder auch ungenannten) Übersichten – das wäre auch nahezu unmöglich. Doch eine gute Auswahl stellen sie alle dar. Apropos Auswahl: Die Anzahl an verfügbaren Custom ROMs ist schier überwältigend. Da jedoch aus harwaretechnischen Gründen nicht einfach jedes ROM auf jedes Gerät passt, ist die Verfügbarkeit für den eigenen Androiden eine ganz andere Sache. Für manche Geräte hat man die Qual der Wahl – für andere steht vielleicht kein einziges Custom ROM bereit. Einige der meist verbreiteten (und für eine größere Anzahl an Geräten verfügbaren) Kandidaten sollen im Folgenden kurz vorgestellt werden.

CyanogenMod

CyanogenMod ist mit Sicherheit das bekannteste Custom-ROM – und der ständig erwähnte ROM Manager findet natürlich automatisch die aktuellste verfügbare Version auf dem eigenen Gerät (sofern es unterstützt wird). Offiziell unterstützt werden mittlerweile über 700 verschiedene Smartphones und Tablets verschiedener Hersteller. Gehört das eigene Gerät nicht dazu, findet sich oftmals eine passende „inoffizielle Portierung“ – die nicht selten irgendwann ihren Weg in die „offizielle Liste“ nimmt. Über zehn Millionen Installationen (Stand: 12/2013, und der Zähler tickt in rasender Geschwindigkeit weiter) sprechen für sich.

Wie bereits zuvor erwähnt, basiert *CyanogenMod* direkt auf dem *AOSP*-Code: Hersteller-spezifische Benutzeroberflächen findet man hier also nicht. Ebenso wenig die „tolle“ vorinstallierte Bloatware – obwohl man selbige hier Dank Kollegen root (der ja gleich frei Haus mitgeliefert wird) recht einfach wieder

loswerden würde.

Stattdessen legt das Team um Steve Kondik eher Wert auf sinnvolle Anpassungen und Ergänzungen, die jeder im Alltag benötigt – oder die zumindest keinen durch zusätzlichen Ressourcen-Verbrauch belästigen, der sie nicht benötigt. So einige der Features, welche eine neue Android-Version mit großen Worten als „Neu!“ bezeichnete, gab es bei *CyanogenMod* schon lange vorher – beispielsweise fanden sich die mit Android 4.2 eingeführten „Quick Settings“ dort schon mindestens zwei Versionen zuvor.

Und bevor jemand fragt: Ja, ich weiß, dass CyanogenMod mittlerweile ein neues Maskottchen hat. Mit gefällt aber nun einmal der Andy auf dem Skateboard …

AOKP

Wem *CyanogenMod* noch nicht „customized“ genug ist: Da wäre noch ein Ableger zu nennen, der seinerseits wieder auf *CyanogenMod* aufbaut. AOKP (Android Open Kang Project) nimmt noch weitere Anpassungen vor, und ermöglicht dem Anwender auch zusätzliche Einstellungen. Allerdings ist hier die Anzahl unterstützter Geräte mit gut 90 geringer als bei *CyanogenMod*.

Wie erwähnt, finden sich viele von *CyanogenMod* bekannte Features in *AOKP* wieder: Etwa die Verfügbarkeit diverser „Themes“, die sich z. B. aus dem *Google Play Store* installieren lassen und die Anpassung des „Look and Feel“ ermöglichen, oder die Verwaltung und Kontrolle von App-Permissions. Analog zu *CyanogenMod* gibt es auch hier stabile „Milestones“ (mit einer Version pro Monat) sowie weniger stabile „Nightlies“ (etwa alle drei Tage) zum Download; über deren Verfügbarkeit kann man sich per *AOKPush* (via *Google Cloud Messaging*) informieren lassen.

Typische AOKP-Ergänzungen sind etwa der *Navigation Ring*, mit dem man über den Home-Key bis zu fünf Apps direkt starten kann – oder auch diverse „Ribbons“, über die sich per Wisch-Aktion eine Reihe von Shortcuts erreichen lassen; egal, ob man sich gerade auf dem Homescreen oder in einer beliebigen App befindet.

Eine detailliertere Übersicht sowie einige Screenshots finden sich auf der AOKP Homepage.

Paranoid Android

Auch Paranoid Android kommt mit einer „Theme Engine“, die das Aussehen manipulieren lässt. Außerdem bringt es „Pie Control“ mit: Im Vollbild-Modus (dem so genannten „Immersive Mode“) stehen dadurch die Funktionen der Soft-Buttons zentral zur Verfügung, ohne dass diese selbst sichtbar sind – sodass mehr Bildschirm-Fläche für die eigentlichen Apps verbleibt. Dazu definiert man einen Punkt oder Bereich des Bildschirms, bei dessen „Drücken“ sich eine „halbe Torte“ öffnet – in der die Button-Funktionen (und mehr) sodann verfügbar sind.

Mit „Halo“ hat *Paranoid* übrigens die „Heads-Up Notifications“ vorweggenommen, welche mit Lollipop schließlich Einzug in AOSP gefunden haben. Diese Art von Integration ist kein Einzelfall: So geschieht es häufiger, dass besonders beliebte Funktionen aus Custom ROMs in „Standard Android“ Einzug halten.

Ein weiteres Feature, mit dem *Paranoid* bereits 2012 die Android-Welt ein wenig revolutionierte: Die Einstellungen dieses ROMs ermöglichen es dem Anwender selbst zu wählen, ob er sein Gerät für ein „Phone“, „Phablet“ oder gar „Tablet“ hält – und passt die Oberfläche entsprechend an. Da gleiches auch für die Bildschirmauflösung möglich ist, lässt sich so auf dem Vier-Zoll-Bildschirm eines Smartphones ein Sieben-Zoll-Tablet „emulieren“ – was dann aber meist gute Augen und „spitze Finger“ voraussetzt.

Offiziell unterstützt werden nur etwa fünfzehn Geräte (Nexus, Oppo, OnePlus). Hinzu kommen jedoch noch eine Zahl von „Legacy Devices“ sowie inoffiziellen Portierungen.

OmniROM

Hinter dieser Custom ROM stehen Namen wie Chainfire (bekannt für seine SuperUser App), XpLoDWilD (von ihm stammt u. a. für die Kamera-App *Focal*), und Dees_Troy (TWRP). Das Projekt entstand als Reaktion auf die Kommerzialisierung bei CyanogenMod (die allerdings letztendlich zum unabhängigen *Cyanogen OS* führte). Für mehr als 80 Geräte ist OmniROM mittlerweile verfügbar; eine Liste der offiziell unterstützten Geräte findet sich im Wiki des Projekts.

Erwartungsgemäß wird hier die Installation über TWRP empfohlen (eine englischsprachige Installations-Anleitung findet sich ebenfalls im genannten Wiki; sie beschreibt im Groben das Vorgehen, wie es auch für andere Custom ROMs zutrifft). Anders als für Custom ROMs üblich, kommt OmniROM nicht von Haus

aus mit root-Zugang daher – allerdings lässt er sich recht einfach nachrüsten, indem man das dafür zuständige update.zip installiert (flasht). Für eine spätere System-Aktualisierung ist ein Updater ins System integriert, den man auch für das automatische Herunterladen und Installieren verfügbarer Updates konfigurieren kann.

Für weitere Details sei auch ein Blick in den Testbericht bei PCWelt (10/2015) empfohlen. Die OmniROM Homepage findet sich hier, und informiert auch über die aktuelle Entwicklung sowie anstehende Neuigkeiten.

Weitere Custom ROMs

Mit der kleinen Auswahl habe ich allenfalls an der Oberfläche gekratzt, und die bekanntesten Kandidaten genannt. Nicht wirklich unbekannt sind auch die ROMs von Dirty Unicorns, die für etwa 30 verschiedene Geräte (hauptsächlich Samsung, Nexus, HTC) verfügbar sind – und von denen PCWelt schreibt, „es gibt kein vergleichbares ROM, welches so viele und so gut strukturierte Funktionen bietet“. Oder das für etwa 70 Geräte verfügbare, besonders schlanke SlimROMs (was jedoch leider bei Android 4.4 stehen geblieben zu sein scheint).

Eine vollständige Liste aller verfügbaren Custom ROMs gibt es nirgends (eine Auswahl wird allerdings bei PCWelt präsentiert). Wer auf der Suche nach einem passenden ROM für sein Gerät ist, findet dafür jedoch eine Reihe möglicher Anlaufstellen:

- Die meisten Android-Foren wie Android-Hilfe.DE oder AndroidPIT bieten gerätespezifische Bereiche, denen sich wiederum ein Bereich für „root & Mods“ unterordnet.
- Eine besondere Fundgrube, allerdings in englischer Sprache, ist das Forum der XDA Developers. Nicht wenige ROMs haben hier ihren Ursprung.
- Stack Exchange bietet unter der Frage Where can I find stock or custom ROMs for my Android device? eine gute Quellen-Sammlung.

Für unentschiedene: MultiROM

Während manche ihre wahre Not haben, überhaupt eine Custom ROM für ihr Gerät zu aufzutreiben, finden sich andere mit einem gegenteiligen Problem konfrontiert: Nachdem sie den Berg verfügbarer Kandidaten durchgesiebt haben, bleiben mehrere in der „engeren Auswahl“ hängen – und man kann sich nicht entscheiden! Für diese „Unglücklichen“ wurde MultiROM entwickelt: Eine Art „Boot-Manager“, mit dem man mehrere ROMs auf einem Gerät verwalten kann. Vorausgesetzt natürlich, das Gerät wird auch unterstützt.

Was sich u. a. herausfinden lässt, indem man die entsprechende Liste in der Beschreibung der App MultiROM Manager analysiert.

Mit der gerade erwähnten App lässt sich *MultiROM* übrigens auch bequem direkt vom Androiden aus verwalten: Sie kann sowohl MultiROM selbst, dessen Recovery sowie passende Kernel installieren und aktualisieren. Darüber hinaus versorgt es auf Wunsch das Gerät auch mit einem auf Ubuntu Touch basierenden ROM. Mit MultiROM kann man sogar von einem über OTG angeschlossenen externen Laufwerk booten. Um dies zu ermöglichen, kommt im Hintergrund ein modifiziertes TWRP Recovery zum Einsatz, mit welchem sich ROMs auch als so genannte „secondary ROM“ installieren lassen. Eine Anleitung findet sich u. a. bei Android-Hilfe.DE in deutscher Sprache, sowie als englischsprachiges Video bei YouTube.

Android ohne Google

Nanu? Ist das nicht ein Widerspruch in sich? Ist Android nicht das mobile Betriebssystem von Google? Genau so gut könnte man ja nach „iOS ohne Apple“ fragen!

Ja und nein. Richtig ist, dass Google das 2003 von Andy Rubin gegründete Unternehmen „Android“ im Jahr 2005 erworben hat. Richtig ist auch, dass Mitarbeiter von Google wohl den meisten Code zum Projekt beitragen. Allerdings ist Android im Kern Open Source, und wird vom Android Open Source Project (kurz: AOSP) betreut. Wird von „Android ohne Google“ gesprochen, meint man landläufig: Ein Android ohne die proprietären Ergänzungen wie Google Apps und Bloatware.

Die Gründe, warum jemand sein Gerät „von Google befreien“ will, sind vielfältig. Um einige davon zu nennen:

- Zu viele vorinstallierte Apps, die man nicht braucht und auch nicht wirklich los wird.
- Für so manche vorinstallierte App verwendet man eine für die eigenen Bedürfnisse besser geeignete Alternative. Warum also ungenutzte Dinge installiert halten?
- Privatsphäre: Nicht jeder möchte seine persönlichen Daten (Kontakte, Kalender, etc.) auf „fremder Leute Computer“ (aka „Cloud“) speichern – was die Google-Apps standardmäßig tun, sofern man es ihnen nicht explizit untersagt.
- Privatsphäre: Man hat Bedenken, dass die proprietären Komponenten „nach Hause telefonieren“.

Mit einer einfachen Deinstallation ist es leider nicht getan: Zu tief sind manche Komponenten im System verankert. Eine gute Voraussetzung ist daher ein Custom ROM. Die kommen schon aus lizenzrechtlichen Gründen i. d. R. ohne Google Apps sowie (zumindest im Fall der im vorigen Kapitel benannten Kandidaten) ohne zusätzliche Bloatware daher.

Fährt man so „ganz ohne“, bleibt allerdings auch manche Bequemlichkeit auf der Strecke: Etwa die netzwerkbasierte Ortsbestimmung, oder die automatische Synchronisation der Kontakte und Kalender. Hinzu kommt, dass verschiedene Apps sich entweder gar nicht mehr installieren lassen („Ihr Gerät unterstützt keine Google Services“) – oder schlicht in einigen Funktionalitäten „versagen“ (Benachrichtigungen via Google Cloud Services, Anzeige von Kartenmaterial via Google Maps). Die folgenden Kapitel sollen daher aufzeigen, wie man diesen Komfort – zumindest größtenteils – wieder herstellt.

microG

Die meisten Funktionalitäten lassen sich mit Hilfe von microG wieder herstellen. In diesem ehemals unter dem Namen NOGAPPS bekannten Projekt haben sich mehrere Entwickler zusammengetan – mit dem Ziel, eine leichtgewichtige Open-Source-Alternative zum proprietären Google-Framework zu schaffen. Während jedoch die Installation der Komponenten mit NOGAPPS noch recht aufwändig war, geht sie mit microG recht leicht von der Hand. Und obwohl noch im Alpha-Stadium, läuft alles erstaunlich stabil: Ich setze diese Lösung bereits seit Monaten auf mehreren Geräten ein. Der Kern besteht im Wesentlichen aus drei Komponenten:

- GmsCore entspricht in etwa den *Google Services*
- GsfProxy ergänzt den Core um *Google Cloud Messaging*
- FakeStore täuscht installierten Apps das Vorhandensein des *Google Play Store* vor, während Blankstore sogar einen Großteil der Funktionalität beinhaltet (Apps suchen, installieren, aktualisieren). Zusätzlich benötigt man ggf. (abhängig vom verwendeten ROM) noch das Xposed Modul FakeGApps.

Die Beschreibung des Implementierungsstandes von *GmsCore* auf der Projektseite bei XDA (bzw. im Wiki des Projektes, Stand 1/2016) gibt auch gleich einen guten Einblick, welche Funktionalitäten in den Google Services stecken. Grob übersetzt:

- Ads/Analytics API: kaum angefasst, wird nie vollständig implementiert
- Auth API: nahezu vollständig implementiert
- Cast API: Arbeit begonnen, jedoch noch nicht nutzbar; Apps, die es benutzen wollen, könnten abstürzen
- Drive API: Noch nichts implementiert, Abstürze möglich
- Fitness API: Noch nichts implementiert
- Games API: Noch nichts implementiert
- Cloud Messaging API: voll einsatzbereit
- Location Provider API: voll einsatzbereit, kümmert sich aber nicht um Vorgaben der AppOps
- Geofencing API: noch nicht unterstützt, Apps könnten abstürzen
- Maps API: teilweise implementiert, die meisten Apps stürzen *nicht* ab, hin und wieder kleine Pannen
- Plus API: nur minimale Implemtierung gegen Crashes, grundlegende Account-Informationen
- Wearable API: Arbeit begonnen, jedoch noch nicht nutzbar; Apps, die es benutzen wollen, könnten abstürzen

Installation

Für diese sind zunächst einige **Voraussetzungen** zu beachten. Man benötigt:

- ein 4.4+ ROM, das *völlig frei von GApps* ist – wie bereits erwähnt, trifft dies i. d. R. für alle im vorigen Kapitel genannten Custom ROMs zu (zumindest unmittelbar nach deren Installation).
- das ROM muss „signature faking“ unterstützen – oder man verwendet alternativ FakeGApps (das war in meinem Fall mit CyanogenMod nötig, um *Google Cloud Messaging* funktionsfähig zu bekommen).
- war zuvor *UnifiedNlp* installiert, muss dies entfernt werden (ist jetzt in *GmsCore* integriert)

Sind diese Voraussetzungen erfüllt, kann es auch schon losgehen:

1. GmsCore installieren: gibt es in keinem Store, aber die `.apk` Datei lässt sich von Github oder aus dem XDA-Thread herunterladen – und einfach über den Package-Manager oder per `adb install` aufs Gerät bringen.
2. Will man *Google Cloud Messaging* benutzen, ist das Gleiche auch für GsfProxy nötig – ebenfalls auf „normalem Wege“.
3. Wer BlankStore noch nicht auf dem Gerät hat, kann dies jetzt nachholen (sofern Zugriff auf den *Google Play Store* gewünscht ist – andernfalls greift man zum FakeStore). Die Vorgehensweise ist auf der Github-Seite der FakeGApps erklärt. In meinem Fall musste (unter Android 4.4) nur die `.apk` Datei nach `/system/priv-app` kopiert, ihre Berechtigungen angepasst, und das Gerät neu gestartet werden. Für den *BlankStore* empfiehlt der Autor jedoch die Installation als „User-App“ (also beispielsweise per `adb install`); andernfalls werden vor der Installation von Apps deren Permissions nicht angezeigt.
4. Jetzt finden sich auch die *microG Settings* im App-Drawer. Hier markiert man beide Checkboxen (dies ist die einzige unterstützte Konfiguration – alles andere ist „experimentell“ und „auf eigenes Risiko“, wie MaR-V-iN beschreibt: *you are free to disable them if you like playing with fire)*.
5. Optional lassen sich jetzt noch die *UnifiedNlp* Backends und Einstellungen konfigurieren.
6. Für die Verwendung der „Location Backends“ darauf achten, dass in *Systemeinstellungen › Standort* nicht „nur Gerät“ ausgewählt ist, sonst wird nur GPS verwendet!
7. Reboot (oder „alles Unerwartete ist möglich“).

Backends

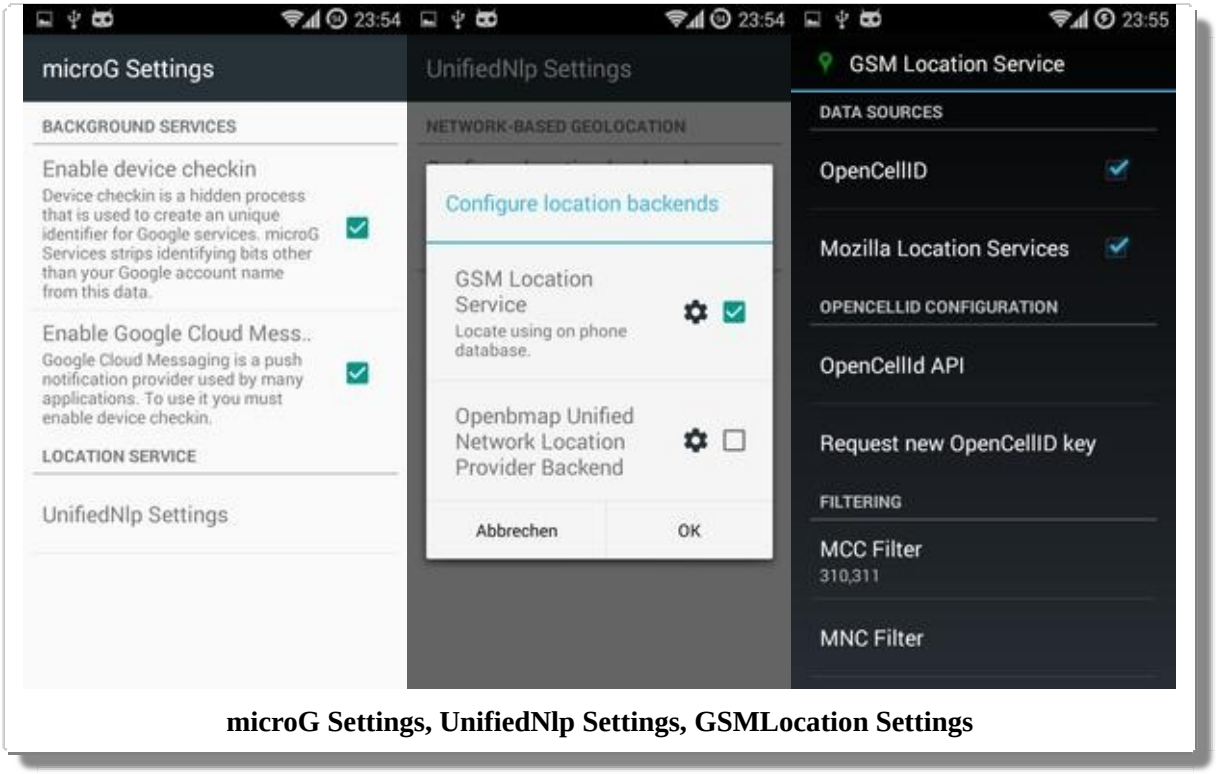
Jetzt ist zwar die Installation der Kernkomponenten vollständig – aber der „Ortungsdienst“ weiß noch nicht, woher er seine Standort-Daten beziehen soll. Für diesen Zweck stehen mehrere Location Backends bei F-Droid zur Verfügung. Diese greifen entweder auf externe Datenbanken im Internet zu, oder arbeiten auch vollständig offline. Wer wie ich die letztere Variante bevorzugt, konzentriert

sich auf die folgenden Kandidaten:

- LocalGsmNlpBackend ermittelt den Standort über Mobilfunkzellen, und nutzt dazu eine lokale Datenbank von OpenCellID *auf dem Gerät selbst.* Diese kann man im Backend selbst erstellen, was recht bequem geht – oder auch auf dem Linux-PC mit lacells-creator, was schneller geht.
- NominatimNlpBackend ist das Gegenstück dazu, um beispielsweise in der Galerie-App Eurer Wahl die in den Fotos gespeicherten GPS-Koordinaten in Ortsbezeichnungen umzuwandeln. Benötigt wird dazu jedoch eine Netzverbindung.
- OpenBmapNlpBackend wäre eine Alternative zum *LocalGsmNlpBackend*, welche statt der Daten von *OpenCellID* auf OpenBmap zurückgreift.

Anwendung

Einfach laufen lassen. Ein Eingreifen ist eigentlich nur nötig, wenn man die Konfiguration anpassen möchte:



microG Settings, UnifiedNlp Settings, GSMLocation Settings

Sollten sich einzelne Apps noch immer nicht installieren lassen (Fehlermeldung „App wurde nicht installiert“, und im LogCat „com.google.android.maps not available“), hilft ggf. die zusätzliche Installation der alten MapsAPI.

Kalender und Kontakte synchronisieren

Es ist schon praktisch, diese Daten überall aktuell verfügbar zu haben. Mit der Trennung von den entsprechenden Google-Diensten haben wir uns aber auch dieser Möglichkeit beraubt. Es gibt jedoch relativ unkomplizierte Möglichkeiten,

dies auf andere Weise zu realisieren:

Mit eigenen Ressourcen wären da etwa WebDAV Server wie Baïkal, Radicale oder ownCloud zu nennen. Für letzteres habe ich das Vorgehen ausführlich in einem Artikel bei IzzyOnDroid beschrieben. Wer keine entsprechenden „eigenen Ressourcen“ hat, kann stattdessen auch einen speziellen Anbieter in Anspruch nehmen. Damit liegen die Daten zwar nicht mehr ausschließlich in den eigenen Händen – aber man kann den „Speicherort“ zumindest wählen. Und da WebDAV einen anerkannten und verbreiteten Standard darstellt, ist man damit nicht nur auf *ownCloud* festgelegt. Dafür lassen sich mit *ownCloud* allerdings gleich mehrere Dinge unter einen Hut bringen und synchronisieren:

- **Kalender:** Mit DAVdroid und CalDAV-Sync stehen hier gleich zwei sehr gut bewertete Android-Pendants zur Verfügung.
- **Tasks:** Werden häufig mit den Kalendern synchronisiert. Einige Aufgaben-Apps erledigen das aber auch selbst über CalDAV.
- **Kontakte:** Dito mit DAVdroid und CardDAV-Sync.
- **Dateien:** Hierfür bietet sich u. a. FolderSync an, das bereits im Kapitel *Synchronisation* unter Dateien und Verzeichnisse in der Cloud beschrieben wurde.

Ich bevorzuge *DAVDroid*, da es sich sowohl um Kontakte als auch Kalender kümmert. Funktioniert in meinem Fall bereits seit etwa einem Jahr problemlos. Eine nähere Beschreibung findet sich weiter vorn im Buch.

Alternativen zu Google Apps

Mit zuvor ausgeführtem ist zwar die Kernfunktionalität wieder hergestellt – doch diverse Apps fehlen nach wie vor: Was nimmt man nun her für Navigation, Mail, Musik & Co? Und vor allem: *Wo* nimmt man es her?

Um zunächst die Quellen zu klären: Meine erste Anlaufstelle ist F-Droid. Dort stehen derzeit knapp 2.000 Open-Source-Apps bereit, die aus dem vom F-Droid Team geprüften Quellcode erstellt wurden. Meine Xposed Module stammen natürlich aus dem Xposed Repository. Benötige ich etwas, das sich nicht in einer dieser beiden Quellen findet, kann ich noch immer mit dem Blankstore auf die Gratis-Apps des *Google Play Store* zugreifen. Oder vom PC aus mit Raccoon, das ebenso den Zugriff auf von mir (z. B. über die *Google Play Store* Website) gekaufte Apps ermöglicht. Wollen letztere – etwa wegen fehlender Verifizierung der „Google Lizenz“ – nicht laufen, helfen oftmals die Entwickler selbst aus: So gibt es etwa Locus Map Pro in Kürze in einer alternativen Version, bei der die Lizenzprüfung auf andere Weise umgesetzt ist.

Bliebe die Frage nach den „App Alternativen“. Die finden sich reichlich, beispielsweise in meinen Übersichten. Desweiteren habe ich vor einer Weile

einen Artikel speziell zu diesem Thema geschrieben, der einige Kandidaten kurz vorstellt: App-Alternativen.

ANHANG

Fragen und Antworten

Bei Woody Allen hieße dieses Kapitel sicher: *Was Sie schon immer zu Android wissen wollten – sich aber nie zu fragen trauten.* Nur keine Hemmungen! Immer her mit den Fragen an Radio Eriwan! Das Prinzip solcher Fragen erklärt der verlinkte Wikipedia-Artikel treffend mit einem kurzen Beispiel:

Anfrage an Radio Eriwan:

Stimmt es, dass Iwan Iwanowitsch in der Lotterie ein rotes Auto gewonnen hat?

Antwort:

Im Prinzip ja. Aber…

- es war nicht Iwan Iwanowitsch, sondern Pjotr Petrowitsch.
- und es war kein Auto, sondern ein Fahrrad.
- und er hat es nicht gewonnen, sondern es ist ihm gestohlen worden.

Alles andere stimmt.

Jeder wusste, was gemeint war – aber keiner konnte die (versteckte) Kritik am System wirklich „dingfest“ machen. So war der politische Witz im Osten. Und auch wenn sich bei manchen vollmundigen Versprechen der Industriegrößen nicht selten ähnliche Fragen aufdrängen (gesagt wird A – aber jeder weiß, dass eigentlich B dahinter steckt), geht es im Folgenden hoffentlich nicht ganz so wild zu …

Apps & Co.

Was passiert bei Deinstallation einer App mit ihren Daten?

Ganz einfach: Sie werden gelöscht. Alles, was das System der zu deinstallierenden App explizit zuordnen kann, wird mit entfernt. Dazu gehört auch das Verzeichnis, in dem die App ihre Konfigurationsdateien und ggf. Daten (im internen Speicher) abgelegt hat. Auch etliche andere Dateien ließen sich recht einfach zuordnen: Da jede App unter Android einen eigenen „User“ darstellt, lässt sich schließlich ermitteln, welche Dateien ihr gehören.

Eine Ausnahme bildet dabei die SD-Karte: Hier erlaubt das Dateisystem keine Benutzerzuordnung. Daher lassen sich Dateien auf der SD-Karte nicht so leicht zuordnen, und bleiben demzufolge meist liegen.

Will Herr oder Frau Poweruser doch einmal eine App deinstallieren, und dabei

die Daten behalten, lässt sich dies mit Hilfe der ADB von der Kommandozeile aus erreichen:

```
pm uninstall -k com.app.wegdamit
```

deinstalliert die App mit dem Paketnamen `com.app.wegdamit` – behält (`-k` = „keep“, behalten) aber die Daten auf dem Gerät.

Natürlich hätte man sie auch mit *Titanium Backup* sichern, und anschließend wieder herstellen können – *Titanium Backup* fragt ja auch brav: Die App, die Daten, oder beides?

Ist das Löschen von Cache und Daten das Gleiche wie Entfernen und Neu-Installation einer App?

Hin und wieder empfiehlt ein Entwickler bei speziellen Problemen mit seiner App diese zu deinstallieren und anschließend neu zu installieren. So manch einer fragt sich: Wäre da das Löschen von Cache und Daten nicht ausreichend? Ist das nicht dasselbe, vom Resultat her?

Ist es nicht. In beiden Fällen werden zwar App-Cache und Daten gelöscht. Bei einer Deinstallation wird aber auch der Dalvik-Cache der App entfernt, und bei einer Neuinstallation wieder frisch aufgebaut, etwaige BroadcastReceiver werden neu registriert, und mehr. Manchmal kann genau an dieser Stelle „der Wurm“ liegen.

Wo speichert Android die Metadaten zu meinen Fotos?

Äh: Was bitte sind Metadaten? Vielleicht sollte das zuerst geklärt werden: Das sind etwa die Stichworte, die man den Fotos zuweisen kann. Und einiges anderes mehr: Welche Covers von Musikalben sich auf dem Gerät finden (und wo), welche Musikalben von welchen Künstlern, Audio-Genres, Playlisten, Videos … Irgendwo muss der Medien-Scanner das ja alles lassen!

Und wo wäre das? Da hierfür der sogenannte *Media Content Provider* zuständig ist, landen die Daten in dessen Datenbanken – die sich unter `/data/data/com.android.providers.media/databases` finden. Und zwar fein getrennt in mindestens zwei Datenbanken: `internal.db` für den internen Speicher, und `external[-<Geräte-ID>].db` für die jeweilige Speicherkarte. Herankommen tut an diese Dateien natürlich wieder einmal nur Freund root – und bei Forensikern erfreuen sie sich besonderer Beliebtheit.

Wer genaueres zu den Strukturen der in den genannten Datenbanken enthaltenen Tabellen wissen möchte, findet beispielsweise in der Entwickler-Doku weitere Angaben – in englischer Sprache, versteht sich.

Wie kann ich eine App im Speicher halten?

Ich benutze App xyz ständig. Doch fast immer, wenn ich zu ihr wechseln möchte, wird sie neu gestartet. Kann ich Android irgendwie dazu bringen, sie im Speicher zu halten?

Diese Frage haben sich die meisten schon einmal gestellt: Warum schließt Android laufend unsere Lieblings-App? Das Prinzip habe ich ja bereits beim Thema RAM Bereinigen erklärt: Wird der Speicher knapp, schlägt der OOM-Killer zu. Mit welcher Wahrscheinlichkeit ihm nun unsere Lieblings-App zum Opfer fällt, hängt unter anderem davon ab, zu welcher „OOM-Klasse“ sie gehört. Ändern kann man diese wieder einmal nur mit root – und deshalb male ich gleich wieder meinen Kasten um das Folgende:

CyanogenMod bietet zumindest für den Standard-Launcher eine entsprechende Option – aber auch nur für diesen. Dafür kann die App einfach in eine „wichtigere Klasse“ eingeteilt werden, indem ihr `oom_adj`-Wert gesenkt wird. Wie man dies für eine beliebige App erledigen kann, erklärt ein Post bei XDA: Man erstellt eine Datei namens `etc/init.d/99applock`, und füllt sie mit folgendem Inhalt:

```
#!/system/bin/sh

sleep 60

PPID=$(pidof com.your.app)
echo “-17” < /proc/$PPID/oom_adj
renice -18 $PPID
```

`com.your.app` ist dabei mit dem Paketnamen der gewünschten App zu versehen – und `etc/init.d/99applock` mit den passenden Rechten: `chmod 777 etc/init.d/99applock`.

Unterstützen alle Android-Apps NFC?

Das brauchen sie gar nicht – darum sollte sich das System selbst kümmern. Lediglich passende „Intents“ müssen sie bereitstellen, über die sie von außen angesprochen und mit entsprechenden Daten gefüttert werden können (vergleichbar mit APIs). Um das Auslesen und Interpretieren der NFC-Tags kümmert sich der zentrale NFC-Service, der dann auch die passende Aktion auslösen sollte.

Vergleichen lässt sich das am besten mit den Barcode-Readern, die ja ebenfalls die passende Aktion wählen: So muss etwa die Kontaktverwaltung auch keine Barcode-Unterstützung haben. Erkennt der Barcode-Reader eine Adresse, ruft er die Kontaktverwaltung auf, und übergibt ihm diese. Oder die

Telefonnummer an die Telefon-App, bzw. die URL an den Web-Browser.

Passende NFC-Apps wären etwa on{X}, oder Trigger (ehemals *NFC Launcher*). Und natürlich muss auch das Gerät selbst NFC unterstützen.

Ich habe meinen einzigen Launcher gelöscht!

Die Tücken von root und Custom ROMs: Einem „normalen Anwender“ dürfte das nicht passieren, da der „Standard Launcher“ im read-only Bereich des Systems installiert ist. Ohne Launcher sieht es unter Android natürlich mau aus: Eingehende Anrufe lassen sich zwar noch annehmen, aber man kann keine Apps mehr starten. Wenn das Kind nun in den Brunnen gefallen ist, gibt es dennoch eine ganz triviale Lösung:

Einfach am PC mit dem Web-Browser den Google Play Store besuchen, sich mit dem auf dem Gerät verwendeten Google-Konto anmelden, sich einen alternativen Launcher aussuchen, und die Installation von dort aus anstoßen. Kurz darauf ist wieder ein Launcher verfügbar, und der Androide somit voll einsatzbereit.

Kein Playstore verfügbar? Auch kein Beinbruch. Dann lädt man sich die APK-Datei aus einer anderen Quelle, und installiert sie mittels ADB: `adb install launcher.apk`.

Backup & Co.

Hilfe! Ich habe versehentlich etliche (einen ganzen Thread) SMS/MMS gelöscht!

Das passiert leicht: Ein unachtsamer Tappser – und statt einer einzelnen SMS verschwindet gleich der ganze Thread im Daten-Nirvana. Sofern kein aktuelles Backup der SMS-Daten existiert, sieht es da schlecht aus (und würde ein solches existieren, würde man sicher nicht nach einer Antwort suchen müssen). Gibt es da noch Chancen?

Mit Android < 4.0: Nur für root, der über das Custom-Recovery sofort ein Nandroid-Backup anlegt. Mit Android ab Version 4.0 kann man den Schritten unter Komplett-Backups ohne root folgen – und hoffen, dass dies die nötigen Dateien enthält.

Diese finden sich nach dem Entpacken der Image-Datei der Datenpartition im Verzeichnis `data/data/com.android.providers.telephony/databases/` (ja, root könnte sich dieses auch mit einem Dateimanager oder per `adb pull` besorgen). Mit einem Programm wie etwa Sqliteman lässt sich nun etwa die im genannten

Verzeichnis befindliche Datei `mmssms.db` nach den gelöschten Nachrichten durchsuchen. Auch die Datei `mmssms.db-wal` (eine SQLite Journal-Datei – genauer gesagt: Das Write-Ahead-Log) enthält häufig weitere diesbezügliche Daten. Vor dem Bearbeiten mit irgendwelchen Tools legt man sich davon am besten Kopien an – damit man bei einem Fehlschlag nicht wieder ganz vorn anfangen muss.

Ach ja: Genauso funktioniert es natürlich auch mit jeder anderen Datenbank, die beliebige andere App in ihrem Datenverzeichnis abgelegt hat.

Wem das alles zu kompliziert erscheint, kann alternativ auch zu einer App greifen: GT SMS Recovery kann, root vorausgesetzt, ebenfalls versehentlich gelöschte Kurznachrichten wieder herstellen.

Die SMS sind noch da – dafür habe ich aber eine Reihe Dateien gelöscht!

Auch kein Weltuntergang. Sicher waren diese auf der SD-Karte – nicht-root Anwender also bitte einfach nach dem Kasten weiterlesen.

Die Dateien waren im internen Speicher? Aha, mit dem root-Explorer gespielt. Dann sollte schnellstens eine Sicherung der Partition durchgeführt werden. Anbieten tun sich hier Nandroid, aber auch `dd`. Da letzteres sich für eines der unter dem Kasten aufgeführten Recovery-Tools besonders gut eignet, dazu ein wenig mehr.

Zuerst muss herausgefunden werden, welche Partition denn benötigt wird. Ist dies nicht bekannt, kann dazu (per ADB oder mit einer Terminal-App) der Befehl `mount` ausgeführt werden. Ohne Parameter aufgerufen, zeigt dieser nämlich alle eingebundenen Dateisysteme an. Ich kürze diese Ausgabe hier einmal ein wenig ab:

```
$ mount
<snip>
/dev/block/mmcblk1p26 on /data type ext3
(rw,nosuid,nodev,noatime,nodiratime,errors=continue,data=ordered)
/dev/block/vold/179:1 /mnt/sdcard vfat <snip>
```

Befanden sich die gelöschten Daten also beispielsweise unterhalb von `/data/data`, muss sich hier das Gerät `/dev/block/mtdblock6` gemerkt werden. Des Weiteren ist auch zu sehen, dass sich die SD-Karte unter `/mnt/sdcard` finden lässt – und hoffentlich genügend Platz aufweist:

```
$ df -h /mnt/sdcard /data/data
Filesystem               Size     Used Available Use% Mounted on
/dev/block/vold/179:1   14.9G     1.7G     13.2G  11% /mnt/sdcard
/dev/block/mmcblk1p26    6.5G   382.1M      6.1G   6% /data
```

Über 13 GB frei auf der Karte, knapp 7 GB maximal werden benötigt: Passt. Es kann also zur Tat geschritten werden. Bei Aufruf des Befehls `dd` genau auf

die Parameter achten! Nicht umsonst wird gemunkelt, `dd` stünde für „disk destroyer“: Verwechselt man hier die Eingabe und Ausgabe, ist letztere danach futsch. Also:

```
dd if=/dev/block/mmcblk1p26 of=/mnt/sdcard/data.img
```

Kurz erklärt: „Disk Duplicator, lese von `/dev/block/mmcblk1p26` und schreibe alles, was dort ist, nach `/mnt/sdcard/data.dd`.“ `if` steht hier also für Eingabe-Datei (englisch „input file“), `of` für „Ausgabe-Datei“ („output file“). Letzteres ist in unserem Beispiel tatsächlich eine Datei – die Eingabe jedoch ein Gerät. Doch unter Unix/Linux sind auch Geräte Dateien, siehe Wikipedia: Gerätedatei.

Wer das nicht alles per Hand erledigen möchte, kann sich dabei auch von meinem kleinen Tool namens Adebar unter die Arme greifen lassen. Dieses erstellt nämlich u. a. auch ein Skript, mit dem sich von sämtlichen wichtigen Partitionen eine Image-Datei anlegen lässt.

Zum Retten der Daten wird nun die SD-Karte an den PC angeschlossen, und ein zuvor heruntergeladenes Datenrettungs-Programm darauf (oder auf die im Kasten erstellten Image-Dateien) losgelassen. Davon gibt es eine ganze Reihe, und drei möchte ich hier auch benennen:

- Scalpel gibt es für Linux (bevorzugt), Mac OS X und Windows. Es ist eines der mächtigeren Tools in dieser Klasse – erfordert aber auch ein wenig Eingewöhnung.
- foremost gibt es (nur?) für Linux. Da es aber auch mit `dd` Images umgehen kann, ist dies für den im obigen Kasten beschriebenen Fall die passende Lösung.
- TestDisk unterstützt die größte Anzahl an Betriebssystemen, und stellt auch eine menügeführte Benutzeroberfläche bereit.

Details zur Nutzung finden sich auf den verlinkten Webseiten des jeweiligen Tools – und würden an dieser Stelle den Umfang des Kapitels sprengen.

Kann ich Daten aus einem verschlüsselten ADB-Backup extrahieren?

Unter Komplett-Backups ohne root habe ich ja beschrieben, wie man ab Android 4.0 auch als „Normalo“ vollständige Backups erstellen kann – Apps und auch ihre Daten. Dabei konnte ein Passwort angegeben werden, mit dem das Backup verschlüsselt wird – gegen unbefugte Zugriffe. Die Zugriffe des Eigentümers sind sicher nicht unbefugt (zumindest, wenn es um die Daten geht) – aber wie kommt er da nun heran, ohne sie auf dem Androiden wieder herzustellen (und damit aktuellere Daten zu überschreiben)?

Der bereits bei besagtem Backup genannte Ryan Conrad hat natürlich auch

dafür eine Antwort parat. Er hat sich eingehend mit dem Thema beschäftigt, da er schließlich dieses Backup/Restore in seinem Programm implementierte. Die einfachste Antwort lautet also: Sein Programm Droid Explorer in Version 0.8.8.7 oder höher zu installieren, und die gewünschte Datei damit entpacken:

Droid Explorer: Restore

Dieser Tipp gilt zumindest derzeit nur für diejenigen, die Windows auf ihrem Computer einsetzen (eine Linux-Version ist zwar in Arbeit, lässt aber mangels Unterstützung durch entsprechende Mono-Entwickler noch immer auf sich warten). Für alle anderen ist ein mögliches Vorgehen auf der verlinkten Seite erklärt. Um wenigstens einen kleinen Anhaltspunkt zu nennen: Die `.ab` Datei ist ein mit AES verschlüsseltes Archiv, in dem sich eine TAR-Datei befindet. Und es gibt ein Tool namens Android Backup Extractor (kurz: ABE), mit welchem man diesen Tarball aus der `.ab` Datei herausholen kann:

```
java -jar abe.jar unpack <backup.ab> <backup.tar> <password>
```

Da es sich bei *ABE* also offensichtlich um ein Java-Programm handelt, sollte es auf den meisten Betriebssystemen laufen. Auch mein eigenes Tool Adebar bietet dafür einen Einzeiler, der zumindest unter Linux funktioniert – zu finden unter `tools/ab2tar`; benötigt wird hierfür OpenSSL und ZLib Unterstützung.

Panik! Nandroid findet kein sd-ext!

Da erstellt man also pflichtbewusst sein Nandroid Backup – und die Bildschirmausgabe endet mit

```
Backing up system…
```

```
Backing up data…
Backing up .android_secure
Backing up cache
No sd-ext found. Skipping backup of sd-ext.
Generating md5 sum…

Backup complete!
```

Was heißt hier: `No sd-ext found`? Habe ich jetzt ein Problem? Ist da etwas „futsch“?

Mit Neffen und mitnichten. Das bedeutet lediglich, dass die SD-Karte nicht über eine EXT3 bzw. EXT4 Partition verfügt, die auf diesen Namen hört. Eine solche wurde/wird von verschiedenen Custom ROMs für ein erweitertes *Apps2SD* verwendet. Gibt es diese Partition also nicht, kommt diese Art von *App2SD* auf dem Gerät wahrscheinlich auch nicht zum Einsatz – die Meldung ist also kein Grund zur Sorge. Andernfalls würde sie nachdrücklicher zum Ausdruck gebracht, und nicht nur „beiläufig ausgegeben“ werden.

Welche Daten sichert eigentlich Google Backup?

Bei der Ersteinrichtung wird man (zumindest ab Gingerbread) ja gefragt, ob man seine Daten bei Google sichern möchte. Was aber wird dabei eigentlich gesichert, wenn man die Frage bejaht?

Ganz offensichtlich sind damit nicht die Kontakte oder Kalenderdaten gemeint – denn die werden auch in die Google-Cloud synchronisiert, wenn man diese Frage verneint. Im Honeycomb User Guide (englisches PDF) heißt es dazu sinngemäß:

- Android Settings, wie WLAN Netzwerke und Passworte, Benutzerwörterbuch, usw.
- Die Einstellungen vieler Google-Apps, wie etwa Browser-Lesezeichen
- Die aus dem Playstore heruntergeladenen Apps

Zumindest der letzte Punkt macht wenig Sinn, da diese Informationen ja ohnehin im Playstore hinterlegt sind. Weiter heißt es jedoch, dass auch einige Apps von Drittanbietern diese Backup-Schnittstelle nutzen, um die Daten ihrer Apps zu sichern. Einige, nicht alle.

Wenn man sich nun mit einem neuen bzw. auf Werkseinstellungen zurückgesetzten Gerät erstmalig mit seinem Google-Konto wieder anmeldet, sollen diese Daten wieder hergestellt werden.

Soweit die Theorie. In der Praxis schlägt leider die Wiederherstellung oftmals fehl, wie zahlreiche Berichte zeigen – man sollte sich also keinesfalls auf diese Funktionalität verlassen. Zumal der Umfang der gesicherten Daten ohnehin recht eingeschränkt ist, und Übertragung sowie Speicherung teilweise im Klartext erfolgt – wie hier im Beispiel der WLAN-Passwörter. Denkt man dabei noch an

Google's *Streetview* Cars, welche auch fleißig alle erreichbaren WLAN-Netze gescannt haben, kann einem überdies mulmig werden: Google weiß also, wo der Router steht – und wir liefern dazu die passenden Zugangsdaten, im Klartext …

Rund um die SD-Karte

Warum kann ich App xyz nicht auf die SD-Karte verschieben?

Das kann verschiedene Ursachen haben:

- Die App unterstützt kein *App2SD* (hat der Entwickler so festgelegt). In diesem Fall wird die Option zum Verschieben erst gar nicht angeboten.
- Die App war dort zuvor bereits installiert (und man hat beispielsweise einen Werksreset gemacht).

Während im ersteren Falle der Entwickler Abhilfe schaffen kann (es sei denn, die App beinhaltet Widgets oder Hintergrund-Dienste – dann darf sie nicht auf die SD-Karte), gilt es in letzterem Falle, selbst Hand anzulegen. Hier sind wahrscheinlich ein paar „Dateileichen" auf der Karte liegen geblieben – und die müssen weg. Am einfachsten geht das natürlich, wenn man die SD-Karte mit einem Kartenleser an einen Computer anschließt, da Android (ohne root) das entsprechende Verzeichnis „versteckt" und keinen Zugriff zulässt. Das Vorgehen ist dann etwa folgendermaßen:

1. Den Paketnamen der entsprechenden App ermitteln. Am Einfachsten geht dies, indem man mit dem Web-Browser die App-Seite im Playstore aufsucht, denn dann steht der Paketname in der URL: Unmittelbar hinter dem `id=` (und endet, sobald ein `&` oder `#` auftaucht). Heißt die URL beispielsweise `https://play.google.com/store/apps/details?id=com.rovio.angrybirds&feature=search_result`, so lautet der Paketname `com.rovio.angrybirds`
2. Auf der SD-Karte nach der zugehörigen Datei suchen. Dies befindet sich unter `.android_secure`, und hieße in unserem Beispiel höchstwahrscheinlich `com.rovio.angrybirds-1.asec`.
3. Diese Datei löschen.
4. App auf SD verschieben – jetzt sollte es wieder klappen.

Ausführliche Gründe, die eine App für die Installation auf der SD-Karte disqualifizieren, nennt die Android Entwickler-Seite. In der Regel hängt dies damit zusammen, dass die SD-Karte z. B. bei Bereitstellung an einem Computer auf dem Android-Gerät u. U. nicht zur Verfügung steht:

- **Services:** Eine App, die Hintergrunddienste bereitstellt. Diese Dienste

stünden dann ggf. nicht zur Verfügung, und würden auch nicht automatisch neu gestartet.
- **Alarm Services:** Registrierte Alarme würden bei Nichtverfügbarkeit der SD-Karte abgebrochen.
- **Live Wallpapers:** Könnten ohne die Karte nicht weiterlaufen, und würden durch das „Standard Wallpaper“ ersetzt.
- **App Widgets:** Könnten nicht mehr aktualisiert werden, und „verschwinden“ vom Homescreen. Meist sind sie erst nach einem Reboot wieder verfügbar.
- **Account Managers:** Können bei nicht verfügbarer SD-Karte natürlich auch nicht arbeiten, sind sie auf selbiger installiert. Die damit verbundenen Konten sind daher nicht sichtbar, wenn die Karte „nicht sichtbar“ ist.
- **Sync Adapters:** Funktionieren nicht, wenn die Karte nicht verfügbar ist.
- **Device Administrators:** Wären dann ebenfalls außer Funktion, was die Geräte-Funktionalität negativ beeinflussen kann.
- **Broadcast Receivers, die auf *boot_completed* lauschen:** Da dieser Event (`boot_completed`) **vor** dem Einbinden der Karte gesendet wird, bekommen die auf der Karte installierten Apps davon nichts mit (hier geht es um “nach dem Booten Starten”).
- **Copy Protection:** Wird Googles Kopierschutz genutzt, kann die betreffende App nicht auf der Karte installiert werden. Googles „Lizenz-Service“ ist davon jedoch nicht betroffen.

Warum kann ich die App xyz nicht updaten?

Sie ist auf der SD-Karte installiert? Ein ähnliches Problem wie das vorige. Aber mit einer anderen Lösung – vorausgesetzt, es ist genügend interner Speicher frei:

1. App in den internen Speicher verschieben
2. Updaten (sollte jetzt funktionieren)
3. App bei Bedarf wieder auf die Karte verschieben

Klingt blöd – klappt aber so.

Muss ich beim Kauf einer SD-Karte auf die „Klasse“ achten?

Das sollte man schon – besonders wenn man auf Performance und Akku-Laufzeit Wert legt. Wer heute eine SD-Karte kauft, sollte nicht aus Geiz-ist-Geil Gründen (Preis) zu einer „Class 2“ Karte greifen – daran hat man wirklich keine Freude. Sie ist recht langsam (spätestens bei Video-Aufnahmen wird man das deutlich spüren), und im Vergleich mit „höheren Klassen“ auch ein wenig Akku-hungrig.

Eine „Class 4“ Karte dagegen ließe sich bereits als „Budget-Variante“ einstufen,

die für viele Einsatzbereiche ausreichend ist – bei Aufnahme von HD-Video könnte es aber auch hier eng werden. Generell empfiehlt es sich daher, noch den einen Euro darauf zu legen, und zumindest zu einer „Class 6“ Karte zu greifen. Deren Spezifikationen sollten auch einer HD-Aufnahme Genüge tun.

Schaden tut eine höhere Klasse natürlich nicht – mehr Speed und schonenderer Umgang mit den Ressourcen haben schließlich noch niemanden gestört. Allenfalls der Preis: Je höher die Klasse, desto höherer auch dieser. Doch da liegt die Toleranzgrenze bei jedem anders.

Nebenbei: Ein Grund, warum höhere Klassen insbesondere bei häufigem Zugriff auf die Karte weniger Akku-Verbrauch bedeuten, ist einfach und einleuchtend: Für das „Bewegen“ der gleichen Datenmenge wird weit weniger Zeit benötigt – die „Ressource“ also wesentlich kürzer beansprucht. An einem Beispiel verdeutlicht: Das Schreiben einer 10 MB Datei (etwa ein Foto mit 12 Megapixel Auflösung) benötigt bei einer „Class 2“ Karte mindestens fünf Sekunden – bei „Class 10“ nur noch eine Sekunde. In diesem Beispiel spart die „bessere Karte“ also 80% der Zugriffszeit, in der CPU, Controller und Karte mit dem Übertragen von Daten (nicht) beschäftigt sind.

Tuning

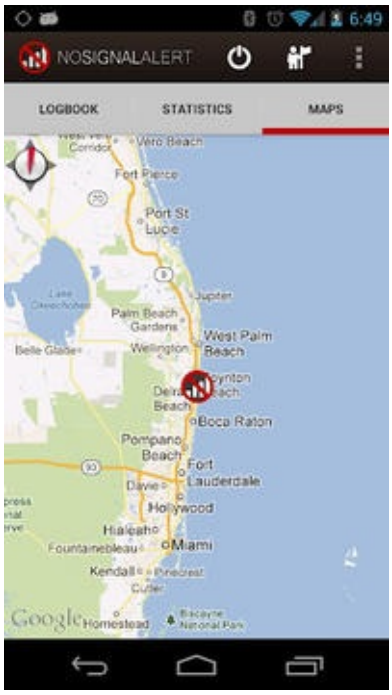
Was ist eigentlich „Mobilfunk-Standby“, und warum braucht es so viel Akku?

Hierbei handelt es sich nicht etwa um die „Hintergrunddaten im Standby“, wie ein Post bei den XDA-Developers suggeriert. Vielmehr hat dieser (auf Englisch, „Cell Standby“ bezeichnete) Zustand direkt etwas mit dem Mobilfunk-Signal zu tun. Oder mit der Abwesenheit desselbigen.

Was passiert, wenn das Signal zu schwach wird? Das Telefon sucht ein neues, stärkeres Signal. Im Regelfall ist ja auch mehr als nur ein Funkmast in der Nähe. Also wird der Antenne mehr und mehr Leistung zugeführt – in der Hoffnung, eine starke und stabile Verbindung gewährleisten zu können. Schließlich will ein Telefon telefonieren, und dazu benötigt es eine Verbindung. Da das System leider nicht intelligent genug zu sein scheint, nach einer angemessenen Zeit dabei eine Pause einzulegen, verbraucht es eben ggf. recht lange Zeit recht viel Strom.

Berichten zufolge (etwa in diesem AndroidForums Post) fällt dieser zusätzliche Verbrauch bei 4G deutlich höher aus als bei 3G. Und wirft man einen Blick auf die

Übersicht der Akkufresser-Daten am Ende dieses Buches, wird klar: 3G frisst sicher noch immer mehr als 2G. Hat man alles gleichzeitig aktiviert, erfolgt die Suche dann auch in allen Bändern (also 2G, 3G und 4G). Daher kann es durchaus sinnvoll sein, sich auf einen Standard festzulegen, wenn man in einem Gebiet mit schlechter Abdeckung unterwegs ist.



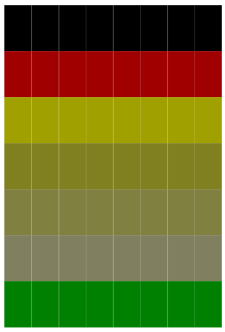
Gänzlich vermeiden kann man dieses Problem natürlich, indem man seinen Androiden ausschließlich im Flugzeug-Modus betreibt – was selbstverständlich keine Lösung ist. Einige interessante Apps gibt es jedoch, die in diesem Zusammenhang genannt werden sollten:

- No Signal Alert protokolliert im Hintergrund, wo man sich in „toten Zonen“ bewegt hat – und kann diese auf einer Karte anzeigen (rechtes Bild). Natürlich lässt sich auch die Log-Datei einsehen. Somit weiß man zumindest, wo dieses Unheil droht.
- Gleiches weiß auch OpenSignalMaps zu berichten. Zusätzlicher Nutzen dieser App: Sie zeigt auch an, welche Sendemasten sich in der Nähe befinden. Außerdem bietet sie einen „Kompass“, der die Richtung zur nächstgelegenen starken Zelle anzeigt. Versteht man also seinen Gesprächspartner immer schlechter, kann man sich Dank dieser App in die richtige Richtung bewegen, um den Empfang zu verbessern. Auch ein Widget ist hier an Bord. Beide Apps warnen auf Wunsch auch, wenn man eine „tote Zone“ betritt.
- Weitere kleine Helferlein wurden ja bereits benannt – speziell für das Mobilfunk-Signal findet man sie auch in einer Übersicht.

Ob man davon betroffen ist, lässt sich leicht an den Akku-Statistiken herausfinden. Diese sind in den Systemeinstellungen enthalten, gelegentlich unter *Telefoninfo*. Taucht auf der ersten Seite *Mobilfunk-Standby* gleich bei den größten Verbrauchern auf, ist die Wahrscheinlichkeit hoch. Dann einfach die kleine Grafik oberhalb der Werte antippen, was zum unteren Screenshot in der Montage führt.

Hier ist der mit *Telefonsignal* beschriftete Balken genauer zu betrachten: Ein kräftiges Grün steht für guten Empfang – was der Verbrauchs-Graph darüber mit einer relativ flachen Kurve als „recht sparsam“ ausweist. Geht der Farbton hingegen ins Gelbliche (mäßiger bis schlechter Empfang), oder wird gar rot (kein Signal), bestätigt auch der Graph darüber einen höheren Stromverbrauch: Die Kurve fällt

stärker. Dass dies nicht etwa daran liegt, dass der Anwender gerade viel mit dem Gerät gemacht hat, lässt sich auch erkennen: Der Bildschirm war zu diesen Zeitpunkten (gelblicher bzw. roter Balken) meist ausgeschaltet (keine blaue Markierung bei *Bildschirm an*).



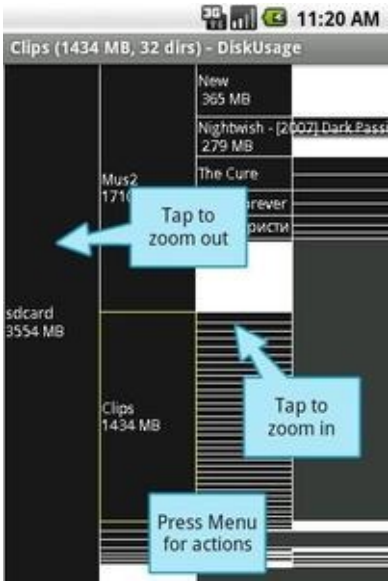
: Radio aus
: Kein Signal
: Signalstärke 0
: Signalstärke 1
: Signalstärke 2
: Signalstärke 3
: Signalstärke 4

Farbdeutung (Quelle: XDA)

Ein Youtube-Video zu Cell Standby kann in diesem Zusammenhang ebenfalls recht aufschlussreich sein.

Wer von dieser Problematik betroffen ist, findet in einer passenden Übersicht einige Helferlein, die in entsprechenden Situationen automatisch tätig werden.

Wer frisst meinen Speicherplatz?



Egal, wie groß der Datenspeicher ist – es ist nie genügend davon vorhanden, er ist immer voll. Aber wo ist er abgeblieben? Und wie lässt sich das herausfinden?

- Das Menü *Einstellungen › Apps* zeigt zum Einen eine „Gesamtzahl“ für belegten und freien Speicher (auf volle Megabytes gerundet). Zum Anderen lassen sich hier die Anwendungen auch nach Größe sortieren, sodass sich besonders speicherhungrige Kandidaten aufspüren lassen.
- Auch wenn eine App mit *App2SD* auf die SD-Karte ausgelagert wurde, bleiben noch immer Reste im internen Speicher zurück (ein vollständiges Auslagern ist also nicht möglich).
- Temporäre Dateien und Cache belegen Speicherplatz. Letzterer lässt sich problemlos aufräumen (siehe Cache Bereinigen) – ersteres ist ein wenig problematischer (siehe unten). Zu diesem gehört übrigens auch das Verzeichnis, in dem der Playstore heruntergeladene Dateien vor der Installation zwischenspeichert. So kann es vorkommen, dass dabei die Fehlermeldung über „ungenügenden Speicher“ erscheint – obwohl am Zielort noch genügend zur Verfügung steht: Es ist dann lediglich der „temporäre Speicher“ voll.
- *Tombstones* (so nennen sich bei Android die unter Unix/Linux als Coredump bekannten Speicherauszüge, die bei Absturz eines Prozesses angelegt werden können) und System-Log-Dateien belegen ebenfalls Speicherplatz

Kleine Helferlein gibt es natürlich auch hier wieder:

- Sandisk Memory Zone zeigt die Kapazitäten (frei/belegt) der verfügbaren Datenspeicher (auch in der Cloud) an, und gibt ebenfalls Einblick in Details
- DiskUsage erlaubt das Durchwandern der Verzeichnisse nach Dateigröße, womit sich die großen Verbraucher schnell aufspüren lassen sollten. Nicht vom Screenshot irritieren lassen: Die App wird noch immer gepflegt.
- Weitere Helferlein finden sich in meiner Übersicht zur Storage-Analyse.

Anwender mit root-Rechten haben natürlich wieder einmal weitere Möglichkeiten. Sie können direkt in der Terminal-App (oder via `adb shell`) auf den internen Speicher zugreifen. Um beispielsweise die fünf „größten Brocken“ auf der Daten-Partition aufzuspüren, würde sich etwa folgender Befehl eignen: `su -c "du /data" | sort -rn | head -n 5`.

Auf gleiche Weise lassen sich natürlich auch die übrigen Dateisysteme erkunden. Welche es da gibt, verrät u. a. ein `df -h` – welches nebenbei auch noch die aktuellen Größenordnungen (freier und belegter Speicher) in gut lesbarem Format anzeigt.

Verschiedenes

Warum werden neue Medien nicht angezeigt?

Ich habe gerade eine ganze Reihe (Bilder / Videos / MP3-Dateien) per WLAN auf mein Android-Gerät kopiert. Warum zeigt (die Galerie / die Video-App / die Musik-App) nichts davon an?

Damit die genannten Apps nicht bei jedem Aufruf erst alle Verzeichnisse nach Medien durchsuchen müssen, hat man sich bei Android etwas Schlaues einfallen lassen: Den *Medien-Scanner*. Dieser wird bei bestimmten Ereignissen automatisch vom System aufgerufen, und durchsucht sodann den gesamten Speicher. Was er dabei findet, wird in eine Datenbank eingetragen. Somit können Apps wie die Galerie oder Video-/Musik-Apps einfach auf diese Datenbank zurückgreifen. Das geht wesentlich schneller – und spart obendrein auch noch Akku.

Der Haken ist: Bei welchen Ereignissen tritt der *Medien-Scanner* denn in Aktion? Die Antwort: Nach dem Systemstart – und nach dem Einbinden der SD-Karte (bzw. dem Trennen der MTP Verbindung). Punkt. Nicht nach dem Kopieren über das WLAN.

Abhilfen mit Bordmitteln wären hier:

- Das Gerät neu starten. Kein toller Vorschlag – klappt aber.
- Die SD-Karte kurz aus- und wieder Einhängen. Geht über die

Systemeinstellungen: *Speicher › Speicherkarte entnehmen* – ist aber auch umständlich.
- Abwarten, bis der *Medien-Scanner* aus einem anderen Grund anspringt – dauert viel zu lange.

Eine einfache Abhilfe findet sich natürlich wieder einmal im Playstore, in Form von kleinen Apps wie SD Scanner (Alternativen in dieser Übersicht). Diese kann man als Shortcut auf dem Homescreen ablegen, und so mit einem einfachen Antippen starten. Die App macht dann nichts anderes, als dem *Medien-Scanner* Bescheid zu geben: Aufwachen, es gibt etwas zu tun! Sodass dieser sofort in Aktion tritt.

Begriffserklärungen

ADB

Die **A**ndroid **D**ebug **B**ridge ist Bestandteil des Android-SDK. Anders als der Name es nahelegt, ist ADB für mehr als nur das Debuggen gut. So lässt sich hiermit ein Android-Gerät steuern und kontrollieren, Dateien können übertragen, installiert oder auch gelöscht werden, und mehr. Eine genauere Beschreibung findet sich auf der Entwicklerseite.

AES

AES ist der Nachfolger des Verschlüsselungs-Standards DES – und bringt als solcher natürlich eine vergleichbar höhere Sicherheit mit.

AOSP

Hinter dieser Abkürzung verbirgt sich das **A**ndroid **O**pen **S**ource **P**roject – also die Kern-Entwicklung des Android-Systems, auf der alle weiteren Android-Entwicklungen (inklusive der Hersteller-spezifischen „Dreingaben“ wie TouchWiz, MotoBlur, Sense, & Co. – aber auch die meisten CustomROMs) aufbauen.

API

Kurzform (oft gesprochen, wie man es schreibt – aber auch als Abkürzung buchstabiert) steht für **A**pplication **P**rogrammers **I**nterface. Gemeint ist hier eine definierte Schnittstelle, über die Informationen bezogen werden können. Die Android-API bietet auf diese Weise z. B. Informationen über Akkustand u. a. m. So muss nicht jeder Programmierer das Rad neu erfinden.

APK-Datei

Die Abkürzung steht für **A**ndroid **P**ac**k**age, und da drin befindet sich in der Regel eine App zur Installation unter Android. Da diese Apps ja in Java geschrieben sind, verwundert es sicher nicht, dass das APK Format eine „Abwandlung“ des JAR (**J**ava **AR**chive) ist, und sich somit mittels WinZip & Co. ein Blick in selbige werfen lässt.

Datei-Manager unter Android erkennen diese Packages natürlich, und bieten an, die enthaltene App zu installieren.

APN

Kürzel für **A**ccess **P**oint **N**ame (zu Deutsch: Name des Zugangspunktes). Gemeint ist in der Praxis mitnichten nur der Name, sondern vielmehr der komplette Datensatz der Einstellungen, die den Zugang zum mobilen Datennetz ermöglichen.

App

Kurzform für *Application*. Wird auch im Deutschen („Neudeutsch“: Applikation) verwendet, da „Anw“ einfach blöd klingt. Denn nichts anderes

bedeutet das englische Wort *Application*: Anwendung.

Im Zusammenhang mit Smartphones aller „Couleur“ (also Früchte wie auch KGMs, kleine grüne Männchen, und ebenso „Fenster-Telefone“) hat sich die Kurzform „App“ eingebürgert – „Application“ (oder im Deutschen „Anwendung“) wird hier eher selten verwendet.

App2SD

Das hat nix mit dem abendlichen „Jezz abba App ins Bett“ zu tun – sondern vielmehr mit der Frage: „Wie kann ich mehr Apps installieren, als in den internen Speicher passen?“. Dazu gibt es mehrere Ansätze, die unter dem Begriff „App2SD“ zusammengefasst sind:

App2SD: Mit Froyo eingeführt. *App2SD* verschiebt Teile der App auf die (einzige) Partition der SD-Karte, wobei die App dies unterstützen muss. Mit Widgets klappt dies in der Regel nicht – hier kommt es zu Abstürzen, da die SD-Karten-Partition bei Anschluss an den PC via USB auf dem Androiden nicht mehr zur Verfügung steht, wenn der USB-Massenspeicher Modus verwendet wird. Dies war ursprünglich der Standard; ab Android 4.0 wurde u. a. aus diesem Grund auf MTP umgestellt.

App2SD+: Gibt es mit einigen Custom-ROMs. Hier wird das Widget-Problem dadurch umgangen, dass eine eigene Partition auf der SD-Karte verwendet wird. Android gibt bei USB-Anschluss lediglich die erste Partition frei, die zweite mit den Apps bleibt somit unangetastet. Wie bereits geschrieben: Benötigt root und Custom-ROM, wobei auch nicht jedes Custom-ROM App2SD+ anbietet (CyanogenMod zum Beispiel nicht).

Link2SD: Im Prinzip wie App2SD+ – nur bedarf es keines Custom-ROMs: Die App wird dabei zunächst auf die zusätzliche Partition verschoben, und sodann ein sogenannter „symbolischer Link“ dorthin im internen Speicher angelegt. Somit wird die App gefunden, als wäre sie im internen Speicher – obwohl sie ganz woanders steckt … Benötigt root und eine zweite Partition auf der SD-Karte.

ART

Die **A**ndroid **R**un**T**ime löst mit Android 5 Dalvik ab. Experimentell lässt sich sich bereits ab Android 4.4 aktivieren, damit Entwickler ihre Apps im Vorfeld auf Kompatibilitäts-Probleme überprüfen konnten. Für den täglichen Einsatz sei davon jedoch bei Android 4.4 noch abgeraten.

Doch was ist mit *ART* denn nun anders? Es ist wesentlich schneller (bis um das Doppelte). Statt wie im Falle von *Dalvik* „Bytecode“ zu erzeugen, wird hier gleich auf nativen Code gesetzt. Das benötigt meist etwas mehr Speicherplatz (durchschnittlich etwa 25% zusätzlich), sowie ein wenig länger bei der Installation – arbeitet dafür aber performanter und ressourcenschonender.

Wer es genauer wissen möchte, wirft am Besten einen Blick in den Artikel ART statt Dalvik bei AndroidNext.

Baseband

Auch „Radio-ROM“ bzw. „Radio-Image“ wird es gern genannt. Das ist quasi die eigentliche Geräte-Firmware („firm“ kommt zwischen „hard“ und „soft“, ist also der „Vermittler“). Hat weniger direkt mit Android, als vielmehr mit der Hardware zu tun – und initialisiert letztere, so dass sie von ersterem genutzt werden kann. Also so etwas ähnliches wie das BIOS beim PC. Und genau wie dieses, befindet es sich i. d. R. auf einem separaten Chip.

Das Teil bootet also die Hardware, und übergibt dann an den eigentlichen Bootloader, der sich dann um Android kümmert.

Bloatware

Vom Hersteller oder Provider vorinstallierte Software, die nicht jedermann benötigt (beispielsweise die Aktien-App). Der Begriff leitet sich vom englischen Wort „to bloat“, „aufblähen“, ab – da diese Apps das ROM unnötig „aufblähen“. Aufgrund der Tatsache, dass sie im ROM integriert sind, kann ein Anwender sie ohne root-Zugriff nicht entfernen.

Bootloader

Sozusagen der „zweite Teil“ nach dem Baseband (daher auch „SPL“ oder „Secondary Program Loader“ genannt). Bleiben wir weiter bei den Hinke-Vergleichen, sind wir hier etwa im „Boot-Manager“ (Lilo, Grub) gelandet (davor wäre noch der „MBR“ oder „Master Boot Record“ auf dem PC – das wäre hier der „IPL“, der „Initial Program Loader“ – der ist bei Androiden in Hardware gegossen, und daher nicht veränderbar).

Aber das wäre jetzt nur sehr grob und ungenau, denn hier steckt mehr drin: Der Android-Bootloader, sowie weitere Boot-Optionen wie das Recovery-Menü, Fastboot, u. a. m.

Brick

In der Regel das Lebensende eines Androiden – der dann nur noch als Briefbeschwerer o. ä. herhalten kann. Wörtlich heißt das zwar „Ziegelstein“, aber das würde im Deutschen u. U. zu meilenweisen Verwechslungen mit gewissen Androiden aus dem Hause Motorola führen …

Was sich Google dabei dachte, als es die gleichnamige Permission einführte, sei der Fantasie anheim gestellt.

Wie verwandelt man einen Androiden nun in einen „Brick“? Dazu werde ich hier keine Schritt-für-Schritt-Anweisung geben (da wenig sinnvoll). Nur soviel sei gesagt: In etwa 95% aller Fälle hängt das mit dem Flashen eines zum Gerät inkompatiblen RUU zusammen. Vermeiden lässt sich solches also durch gründliches Lesen der Anleitungen und prüfen des „Zubehörs“ ***vor*** dem

„Brutzeln“.

BroadcastReceiver

Ein *BroadcastReceiver* ist eine Android-Komponente, die es einer App erlaubt, sich für Ereignisse des Systems oder auch anderer Apps zu registrieren. Tritt dieses Ereignis („Event“) auf, werden alle registrierten „Receiver“ (Empfänger) davon benachrichtigt. Am bekanntesten ist in diesem Zusammenhang der Event `BOOT_COMPLETED`; Apps, die sich für diesen registrieren möchten, erkennt man an der Permission „beim Booten starten“.

Detailliertere Informationen finden sich beispielsweise im Artikel BroadcastReceiver bei WikiBooks (auf Deutsch) sowie im Tutorial Android BroadcastReceiver bei Vogella (auf Englisch).

CalDav

CalDav steht für *Calendaring Extensions to WebDAV* und bezeichnet den nach RFC 4791 spezifizierten Zusatz, der einen Zugriff auf Kalenderdaten über WebDav ermöglicht (siehe Wikipedia).

CSV

Diese Abkürzung steht für den englischen Begriff ***C****omma* ***S****eparated* ***V****alues*. Er beschreibt das Format einer reinen Textdatei zur Speicherung einfacher Tabellendaten (siehe auch Wikipedia für Details).

Custom ROM

Ein *Custom ROM* ist eine alternative Firmware, mit der man das vorinstallierte Android-System ersetzen kann – sofern man über root-Rechte auf dem Android-Gerät verfügt. Custom ROMs sind speziell auf die entsprechenden Geräte angepasst, und bieten in der Regel einen Mehrwert. Etwa eine aktuellere Android-Version, als der Hersteller sie anbietet. Oder weniger „Zwangsbeglückungen“ (vorinstallierte Apps, die sich nicht ohne weiteres entfernen lassen – siehe Bloatware). Auch zusätzliche Funktionalitäten sind hier keine Seltenheit.

Dalvik

Dafür muss ich ein klein wenig ausholen: Android besteht, vereinfacht gesagt, aus einem Linux-Kernel, auf dem eine spezielle Java-Variante läuft. Letzteres ist (bis einschließlich Android 4.x – ab 5.x siehe ART) die sogenannte *Dalvik VM* (wobei „VM“ für „Virtual Machine“ steht). Android Apps sind also in Java geschrieben.

Für die Ausführung der App wird der Java Code in einen sogenannten „Byte Code“ übersetzt, der optimal auf die Hardware (und Android-Version) angepasst ist. Damit das nicht bei jeder Ausführung der App geschehen muss, passiert diese „Übersetzung“ unmittelbar nach der Installation der App – und der Byte-Code wird im sogenannten *Dalvik Cache* abgelegt. Da dies

nach der Installation eines „neuen Systems“ für alle Apps geschehen muss, dauert auch der erste Start nach der Neuinstallation ein wenig länger (dafür geht die Ausführung der Apps nachher entsprechend schneller).

Bei der Installation eines neuen ROMs muss aus genannten Gründen (optimale Anpassung ans System) der *Dalvik Cache* neu aufgebaut werden. Dafür gibt es im Recovery-Menü einen extra Menüpunkt – aber das ist im entsprechenden Kapitel auch erklärt.

Debuggen

Wörtlich „entkäfern“. Ein Computer-antiker Begriff, der noch aus einer Zeit stammt, in der „Programmierung“ durch das Ziehen von Drähten, Stecken von Röhren und Löten von Leiterbahnen stattfand. Da war der „Bug“ im „Programm“ nämlich durchaus wörtlich zu nehmen – wenn ein verbrutzelter Käfer für einen Kurzschluss sorgte.

Die Käfer sind mittlerweile zu groß geworden (oder vielmehr die Chips zu klein), trotzdem haben sich beide Begriffe gehalten: „Bug“ für einen Fehler im Programm, und „Debuggen“ für die Suche nach und das Entfernen desselben.

DES

DES steht für ***D****ata* ***E****ncryption* ***S****tandard*. Es handelt sich dabei um einen symmetrischen Verschlüsselungsalgorithmus, der 1976 als offizieller Standard der US-Regierung bestätigt und seither international vielfach eingesetzt wird.

.dex

Dalvik **EX**ecutable: Der beim Begriff Dalvik beschriebene Bytecode.

DHCP

Die Abkürzung steht für das ***D****ynamic* ***H****ost* ***C****onfiguration* ***P****rotocol*, welches in vielen Netzwerken zum Einsatz kommt. In WLANs ist es fast immer in Verwendung. Meldet sich ein Gerät beim DHCP-Server an, erhält es von diesem die für die Netzwerk-Kommunikation notwendigen Konfigurations-Parameter: Eine eigene IP-Adresse, die Adresse des zuständigen Routers, Nameserver, usw.. Detailliertere Informationen können u. a. diesem Wikipedia-Artikel entnommen werden.

Cloud

„Computer fremder Leute“ – dies sollte man sich immer vor Augen halten, bevor man selbige benutzt. Cloud-Dienste werden über das Netzwerk zur Verfügung gestellt; dabei kann es sich um Speicherplatz, Rechenkapazität, aber auch vollständige Software-Lösungen handeln. Detailliert beschreibt dies u. a. Wikipedia.

Es klingt immer so schön abstrakt und fortschrittlich, wenn es heißt: „Wir

speichern unsere Daten in der Cloud.“ Bei der Prüfung von Konzepten auf Sicherheits-Aspekte sollte man ein globales „Suchen-und-Ersetzen“ auf den entsprechenden Dokumenten durchführen: `s/Cloud/Computer fremder Leute/g`. Der Satz „Wir speichern unsere Daten auf Computern fremder Leute“ dürfte dann zumindest bei sensiblen Firmen-Interna für das eine oder andere Stirnrunzeln sorgen – und ggf. das eine oder andere Konzept in die Überarbeitung zurückverweisen.

DNS

Das Kürzel steht für das ***D****omain* ***N****ame* ***S****ystem*. Vereinfacht gesagt, handelt es sich dabei um eine Art „Telefonbuch“ für das Internet, mittels welchem sich Domain- oder Rechner-Namen wie beispielsweise `www.google.com` in „Anschluss-Nummern“ (die hier „IP-Adressen“ heißen) wie in diesem Beispiel `209.85.148.104` übersetzen lassen. Wer sich für weitere Details interessiert, kann diese u. a. bei Wikipedia erfahren.

eMMC

Eine **e**mulierte **M**ulti**M**edia**C**ard: Gibt sich quasi als SD-Karte aus – tatsächlich handelt es sich jedoch um NAND-Flash.

Factory-Reset

Auch *Rücksetzen auf Werkseinstellungen* genannt: Wiederherstellung des Auslieferungs-Zustandes. Stimmt natürlich nicht so ganz, denn die ursprüngliche Firmware wird dabei nach einem Update nicht wieder hergestellt; es werden lediglich alle Nutzerdaten einschließlich vom Anwender installierter Apps etc. gelöscht.

Fastboot

Der Name ist zunächst ein wenig irreführend – handelt es sich hier doch nicht um die Möglichkeit, das Android-Gerät schneller einsatzbereit zu haben. *Fastboot* hat eigentlich mit dem installierten Android-Betriebssystem nicht einmal direkt etwas zu tun.

Zu finden ist ein Fastboot-Eintrag gelegentlich im Boot-Menü. Und gedacht ist es in erster Linie zum schnellen Bearbeiten von Partitionen via USB. Dazu wählt man am Android-Gerät diesen Punkt aus, und kann dann vom PC aus mit der entsprechenden Software passende Befehle absetzen – etwa um die Daten auf einer Partition zu löschen, mit einer Image-Datei zu überschreiben, oder schlicht das Gerät neu zu starten.

Flashen

Den Androiden mit einem neuen ROM versehen – sei es ein „offizielles Firmware-Update“, oder ein Custom-ROM. Der Name rührt daher, dass hier die Daten größtenteils im „internen Speicher“, dem sogenannten „Flash Speicher“, landen.

FTP

Das ***F**ile **T**ransfer **P**rotocol* dient, wie der Name es sagt, der Übertragung von Dateien im Netz. Es gilt für diesen Einsatzbereich als besonders effizient und ressourcenschonend. Details lassen sich wieder in der Wikipedia nachlesen.

GPG

***G**NU **P**rivacy **G**uard* (GPG) ist ein auf OpenPGP basierendes Kryptografiesystem, welches u. a. RSA-Schlüssel empfiehlt und verwendet.

GPS

Das ***G**lobal **P**ositioning **S**ystem* wird genutzt, um per Kreuzpeilung verschiedener Satelliten den aktuellen Standort in bis zu vier Dimensionen zu bestimmen: Länge, Breite, Höhe und Zeit. Nähere Informationen finden sich in diesem Wikipedia-Artikel.

GSM

Abkürzung für **Global System for Mobile Communications**, manchmal auch als **2G** bezeichnet: Der „Mobilfunkstandard der zweiten Generation“ ist das älteste derzeit noch im Einsatz befindliche System. Benötigt wenig Strom für Telefonate – bietet aber nur sehr geringe Datenübertragungsraten.

Hardreset

Wenn der Softreset, also das „weiche Herunterfahren“, nicht mehr funktioniert, muss dem Gerät zum Ausschalten die Stromzuführ entzogen werden. Dies geschieht i. d. R. durch Entfernen der Batterie, oder im Falle einer fest verbauten selbigen durch das „Anpieksen“ des Reset-Löchleins.

Nicht selten wird dieser Begriff als Synonym für den Factory-Reset verwendet, der jedoch etwas ganz anderes ist (siehe dort).

HTTP

Hinter dieser Abkürzung verbirgt sich das **H**yper**T**ext **T**ransport **P**rotocol, welches zur (unverschlüsselten) Übertragung von Webseiten z. B. mit Web-Browsern verwendet wird. Erkennbar meist daran, dass die Adresse mit `http://` beginnt.

HTTPS

Klingt so ähnlich wie der vorige Punkt – und ist es auch: Das **H**yper**T**ext **T**ransport **P**rotocol, **S**ecure, tut das gleiche – nur eben verschlüsselt, also für sicherere Übertragung. Erkennt man dann daran, dass die Adresse im Browser mit `https://` beginnt.

iCal

Gemeint ist hier das auf vKalender basierende Datenformat ICalendar. Dieses dient zum Austausch von Kalenderinhalten, und ist in RFC 5545 standardisiert.

IMEI

Die International Mobile Station Equipment Identity (IMEI) ist eine eindeutige 15-stellige Seriennummer, anhand derer jedes GSM- oder UMTS-Endgerät eindeutig identifiziert werden kann. (Wikipedia). Diese Nummer ist also Geräte-spezifisch, und wird von diversen Werbe-Modulen gern zur Identifizierung herangezogen. Mit ihr lässt sich aber auch ein Gerät beim Netzanbieter sperren, sodass ein Dieb es nicht mehr verwenden kann (zumindest nicht im gesperrten Netz – dies weltweit durchzusetzen, dürfte ein wenig aufwendig sein)

IntraNet

Ein privates, i. d. R. lokal begrenztes Netzwerk (siehe Wikipedia). Abgeleitet vom lateinischen „intra“ („innerhalb“) und dem englischen „net“ (Netzwerk), da es sich meist um ein firmeninternes Netz handelt.

Kernel

Da steckt das Wort „Kern“ drin, genau. Wenn wir hier vom „Kernel“ sprechen, meinen wir den „Betriebssystem-Kern“, den „Linux-Kernel“. Das ist, vereinfacht ausgedrückt, eine Abstraktions-Schicht: Unten speziell an die jeweilige Hardware angepasst, stellt der *Kernel* „oben“ eine einheitliche Schnittstelle (API) für die Software zur Verfügung.

Bei Android läuft auf dem *Linux-Kernel* die Dalvik-VM (eigentlich je eine pro App), und in der *Dalvik-VM* sodann die App.

LTE

Abkürzung für **Long Term Evolution**, oft auch als **4G** bezeichnet: Schnellerer Nachfolger des Mobilfunkstandards **3G** alias UMTS (bis zu 300 Megabit pro Sekunde).

KML

Die ***K**eyhole **M**arkup **L**anguage* ist ein spezieller XML-Dialekt, der zum Austausch geografischer Informationen verwendet wird. Die bekanntesten Vertreter, die dieses Format verwenden, sind *Google Maps* und *Google Earth*.

Eine Spezialform davon ist *KMZ*: Hier steht das „Z“ für „ZIP“, die Datei ist also komprimiert. Auf diese Weise ist es möglich, zusätzliche Elemente wie Icons mit der KML-Datei beisammen zu halten.

MAC-Adresse

Die MAC-Adresse (Media-Access-Control-Adresse) ist die Hardware-Adresse jedes einzelnen Netzwerkadapters, die zur eindeutigen Identifizierung des Geräts in einem Rechnernetz dient. Bei Apple wird sie auch Ethernet-ID, Airport-ID oder Wi-Fi-Adresse genannt, bei Microsoft Physikalische Adresse. (Wikipedia)

Manifest

Jede APK-Datei (also jede Android-App) enthält eine `Manifest` Datei, in der die Metadaten zur App enthalten sind. Bei der Installation wird diese Datei ausgelesen, um die entsprechenden Informationen dem System verfügbar zu machen. Die Datei enthält u. a.:

- eine eindeutige ID für die Anwendung (den sogenannten „Package Namen“)
- eine Auflistung aller Komponenten einer Anwendung (Aktivitäten, Services, etc.)
- eine Auflistung der von der App geforderten Permissions

Eine detaillierte Beschreibung findet sich auf Android.COM.

MTP

Das Media Transfer Protokoll löst ab Android 4.0 den USB Massenspeicher ab. Damit kann bei bestehender USB-Verbindung gleichzeitig vom Androiden und vom Computer auf die SD-Karte (und ggf. weitere Speichermedien) zugegriffen werden – was allerdings auf dem Computer passende Treiber-Software voraussetzt. Auch ist vom Computer ein Direktzugriff auf die Speicherkarte (z. B. zum Formatieren) damit nicht länger möglich.

NAND-Flash

Daraus besteht der interne Speicher unserer Androiden: Flash-Speicher, der in der sogenannten NAND-Technik gefertigt ist. Genauere Details dazu lassen sich u. a. der Wikipedia entnehmen.

Nandroid Backup

Ein vollständiges System-Backup, welches sich z. B. aus dem Recovery-Menü heraus erstellen und auch wieder herstellen lässt. Hier werden nicht einzelne Dateien gesichert, sondern ein Abbild („Image“) des gesamten Systems wird angelegt (je nach Nandroid-Version auch als TAR Archiv). Es ist also ein „Alles-oder-Nichts“: Die Wiederherstellung einzelner Dateien ist hier nicht vorgesehen (auch wenn dies beispielsweise mit Titanium Backup möglich ist).

Insbesondere bevor man ein Custom-ROM einspielt, aber auch generell vor einem System-Update sollte ein Nandroid-Backup angelegt werden. Es ist natürlich auch sonst immer eine gute Idee, ein komplettes Backup zur Hand zu haben.

NAS

N*etwork* ***A****ttached* ***S****torage* – oder, stark vereinfacht ausgedrückt: Festplatte mit Netzwerk-Anschluss „for the Housegebrauch“. Mittlerweile gibt es NAS-Systeme mit einer Kapazität von 2 TB bereits für weniger als 200 Euro.

NFS

Wie der Name es vermuten lässt, erlaubt das ***N**etwork **F**ile **S**ystem* den Dateizugriff über das Netzwerk – ähnlich wie bei einer Windows-Laufwerks-Freigabe. Dieses Protokoll kommt hauptsächlich unter Linux/Unix zum Einsatz. Nähere Informationen können u. a. der Wikipedia entnommen werden.

Notification Area

Die *Notification Area* ist auch als „Statusbereich“ bekannt, und wird durch die Statusleiste am oberen Bildschirmrand repräsentiert. Zieht man diese Leiste nach unten, öffnet sich die „Area“ mit detaillierteren Informationen, welche Apps dort hinterlegen können.

OOM-Killer

Der Name lässt etwas „Böses“ vermuten – doch der Benannte sorgt für Stabilität: Es handelt sich um den System-Prozess, der – sobald der Arbeitsspeicher (RAM) knapp wird (OOM: **O**ut **O**f **M**emory) – wieder für Platz sorgt. Indem er „Altlasten“ entsorgt: Apps, die nur im Hintergrund rumliegen und länger nicht mehr benutzt, oder gar vom Benutzer bereits beendet wurden, beispielsweise. Mit root-Rechten ausgestattet, kann man auch konfigurieren, wie viel Platz er mindestens freihalten soll.

OTA

Da liegt was in der Luft – denn OTA steht für „**O**ver **T**he **A**ir“. Ja was denn? Beim Rundfunk ist es „On The Air“ und heißt Musik. Bei Phil Collins „In The Air Tonight“. Und bei Android ein „(komplettes) Over The Air Update“ – also ein Kotau, sozusagen. Die Frage wäre da nur, wer dabei der Kaiser ist. Die Übersetzung des Begriffes „Kotau“ bedeutet soviel wie „Kopf stoßen“ – bei so manchem *OTA* dürften die Anwender das gemerkt haben …

Also, kurz gefasst: Beim *OTA* werden Software-Updates des Herstellers/Providers über das Funknetz des letzteren verteilt.

OTG

Kurz für „USB On-the-go“. Ermöglicht es entsprechend ausgerüsteten USB-Geräten, ohne einen zentralen Host-Controller miteinander zu kommunizieren. Zum Einsatz kommt dabei nicht etwa eine Peer-to-Peer Technologie – sondern eines der beiden Geräte übernimmt eine Art „eingeschränkte Host-Funktionalität“.

Partition

Eine Partition ist ein zusammenhängender Bereich auf einem Datenträger (unter Windows häufig mit einem Laufwerksbuchstaben verbunden).

Auf einem Android-System sind immer mehrere Partitionen in Benutzung, auch wenn nur der interne Speicher zur Verfügung steht (und keine SD-Karte eingelegt ist): So ist das `/system` in der Regel nur lesend eingebunden (um

Änderungen im Betrieb zu verhindern), während für Apps und Daten eine eigene Partition (`/data`) bereitsteht.

Die SD-Karte beinhaltet meist nur eine (in der Regel unter `/sdcard` eingebundene) Partition. Es sind aber auch hier mehrere Partitionen möglich (und werden z. B. mit App2SD+/Link2SD auch verwendet) – wobei Android bei Anschluss an den PC via USB nur jeweils die erste Partition davon freigibt.

Weitere Details können z. B. bei Wikipedia nachgelesen werden.

Peer-to-Peer

Das englische Wort „peer“ bedeutet soviel wie „Gleichgestellter“. Teilnehmer in einem Peer-to-Peer-Netzwerk sind sich also „gleichgestellt“ – im Gegensatz zu einem „Client-Server-Modell“. Jeder Teilnehmer kann also sowohl Dienste eines anderen in Anspruch nehmen, als auch eigene Dienste bereitstellen.

Mehr dazu kann auch in diesem Wikipedia-Artikel nachgelesen werden.

Permission

… meint hier die im Android-System definierten Zugriffsberechtigungen, wie im Kapitel Zugriffsrechte erklärt. Eine Übersicht dazu findet sich im Anhang Android Permissions.

Proxy

Ein Vermittler (von lateinisch *proximus*, der Nächste). Gemeint ist hier ein Service, der ihm übergebene Anfragen weiterleitet – da sie beispielsweise auf direktem Wege nicht zustellbar sind. Denkbar ist auch, dass ein Proxy Daten für den schnelleren Zugriff optimiert – meist indem er selbige zwischenspeichert („caching proxy“). Er könnte aber auch, wie im Falle des Web-Browsers *Opera* oder des „Mobilizers“ von *Google* Bilder für mobile Geräte herunterrechnen, um so den Traffic zu minimieren (und die Übertragung zu beschleunigen). Weitere Details finden sich u. a. in der Wikipedia.

RAM

Diese drei Buchstaben stehen für ***R****andom* ***A****ccess* ***M****emory* – also Speicher, auf den man nach Belieben an beliebiger Stelle zugreifen kann. Im Gegensatz nicht etwa zu ROM, sondern zu Dingern wie Bandlaufwerken (jaja, so alt ist der Begriff schon), bei denen man sich erst mühsam vom Start zur gewünschten Position (linear) vortasten muss.

Sowohl auf PCs wie auch auf unseren Androiden ist damit meist der Arbeitsspeicher gemeint, in den die Programme/Apps zur Ausführung geladen werden. Üblicherweise ist dies der Bereich, der generell zu klein ist – oder von dem man halt nie genug haben kann.

Recovery Menü

Ein separater Bereich des Boot-Menüs, aus dem heraus verschiedene Operationen wie Nandroid-Backup oder auch das Bereinigen des Dalvik-Caches möglich sind.

In das *Recovery Menü* gelangt man in der Regel durch eine spezielle Tasten-Kombination beim Einschalten. Diese ist aber zumindest von Hersteller zu Hersteller unterschiedlich. Am vertbreitetsten ist das Halten der „Leiser-Taste“ während des Einschaltens (und darüber hinaus, bis das Menü erscheint).

Reguläre Ausdrücke

Ein *regulärer Ausdruck* lässt sich am besten als Zeichenkette mit Suchmustern beschreiben. Dank komplexer Regeln lässt sich damit recht gezielt auch in größeren Textmengen suchen. Genauere Details lassen sich beispielsweise diesem Wikipedia-Artikel entnehmen.

ROM

Richtig **o**ffensichtlicher **M**ist ist diese **r**eal **o**fferierte **M**ehrdeutigkeit: Manchmal hat man den Eindruck, er wurde nur zur Verwirrung der Massen eingeführt. Wer einmal den Namen für eine Android-Komponente nicht kennt, sagt einfach „ROM“. Klingt, als wüsste man voll Bescheid – und die Chance, dass das auch noch Sinn ergibt, ist verdammt hoch …

Aber im Ernst: Worum geht es hier? Eigentlich steht der Begriff „ROM“ für ***R**ead **O**nly **M**emory* – also Speicher, auf den ausschließlich lesend zugegriffen werden kann. Ja, richtig: So wie bei CD-ROM, da steckt das ja auch drin. Nur bei Android, da kann das alles mögliche sein. Nicht selten sachlich falsch – aber wen kümmert’s? Schauen wir uns also die einzelnen Bedeutungen einmal an:

Systemspeicher: Teile des Android-Systems werden in der Tat „nur lesend“ eingebunden. Unter anderem eine Schutzmaßnahme, um Veränderungen zu erschweren (damit wir die dusseligen Apps, mit denen uns die Hersteller/Provider „beglücken“, nicht einfach löschen können). So heißt es z. B. in den Spezifikationen des HTC Wildfire: „384 MB RAM; 512 MB ROM“. Nonsens(e): Ein Blick hinter die Kulissen offenbart, dass nur 250MB read-only (/system) eingebunden sind. Die restlichen 250MB „ROM“ stehen zur Installation von Anwendungen zur Verfügung. Read-only? Mitnichten. De facto kann der ganze Bereich jederzeit schreibbar gemacht werden, sofern man root hat. Also eher irreführend – richtiger müsste es hier heißen: „interner Speicher“, oder – in Abgrenzung vom RAM – „interner Flash-Speicher“.

Das System selbst: Um die Verwirrung komplett zu machen, wird auch hier gern von „ROMs“ gesprochen. Das hat schon Tradition: Auch bei älteren Spiele-Konsolen sprach man davon, „ein ROM zu laden“. Hier war es aber

„seinerzeit“ tatsächlich eine Cartridge – also ein Speicher-Chip, den man an das Gerät ansteckte. Später kamen dann die Emulatoren, welche die „antiken Geräte“ auf moderner Hardware emulieren können. Und hier kommt diese „Cartridge“ natürlich in Form einer Datei daher. Den Begriff „ein ROM laden“ hat man beibehalten. Und schließlich weiter übertragen.

root

Aus Herstellersicht: Die Wurzel allen Übels. Objektiv betrachtet: Der Administrator (auch „SuperUser“) eines Linux-Systems. Der darf alles, und kann alles (kaputtmachen auch, ja).

RSA

Klingt wie NSA, ist aber etwas anderes (und kann der NSA ggf. auch die Schnüffelei erschweren): *RSA ist ein asymmetrisches kryptographisches Verfahren, das sowohl zur Verschlüsselung als auch zur digitalen Signatur verwendet werden kann* (Wikipedia). Hierbei kommt ein Schlüsselpaar (privater und öffentlicher Schlüssel) zum Einsatz. Der Name des Verfahrens setzt sich aus den Anfangsbuchstaben der Familiennamen seiner Erfinder zusammen: **R**ivest, **S**hamir, **A**dleman (und nein, hinter „NSA“ verbergen sich nicht die Familiennamen der amerikanischen (Vize-)Präsidenten Nixon, Sherman und Adams).

RUU

Radio **U**nit **U**pdate: Eine Art update.zip für das Radio-Image, also ein „Firmware-Upgrade“.

ROM **U**pgrade **U**tility: Wie der Name bereits sagt, ein Utility zum Upgrade des ROM, welches entweder vom Hersteller oder von Drittanbietern zur Verfügung gestellt und vom PC aus installiert wird.

Wenn nicht ganz klar ist, was von beidem gemeint ist, ist es in der Regel das erste – wobei das durchaus mit dem zweiten identisch sein kann, da die Begriffe oftmals gleichbedeutend verwendet werden. Was mancherorts als „ROM Upgrade Utility“ bezeichnet wird, ist nämlich nichts anderes als das Update des Radio-Images.

Salt

Salt (deutsch Salz) bezeichnet in der Kryptographie eine zufällig gewählte Zeichenfolge, die an einen gegebenen Klartext vor der Verwendung als Eingabe einer Hashfunktion angehängt wird, um die Entropie der Eingabe zu erhöhen. (Wikipedia)

SDK

Das SDK ist die Grundlage für die Android App Entwicklung und liegt für die jeweilige Version von Android vor. Es ist aber nicht nur dafür verwendbar, sondern bringt auch einige brauchbare Tools wie Fastboot, den Dalvik Debug

Monitor sowie ADB mit.

Ergänzt man das Ganze noch um Eclipse, kann es mit der Entwicklung von Apps losgehen!

SIP

Das ***S****ession* ***I****nitiation* ***P****rotokoll* findet in der Internet-Telefonie häufige Anwendung, und wird im allgemeinen Sprachgebrauch teilweise gar mit „VoIP“ (Voice-over-IP, also der Internet-Telefonie selbst) gleichgesetzt. Nähere Informationen finden sich u. a. bei der Wikipedia.

SMB

S*erver* ***M****essage* ***B****lock (kurz SMB, teils auch als LAN-Manager- oder NetBIOS-Protokoll bekannt) ist ein Kommunikationsprotokoll für Datei-, Druck- und andere Serverdienste in Netzwerken. Es ist der Kern der Netzwerkdienste von Microsofts LAN Manager, der Windows-Produktfamilie sowie des LAN Servers von IBM. Weiter wird es von den frei verfügbaren Softwareprojekten Samba und Samba-TNG verwendet, um Windows-Systemen den Zugriff auf Ressourcen von UNIX-basierten Systemen zu ermöglichen und umgekehrt.* (Wikipedia) Unter Linux/Unix wird das Kürzel übrigens mit zwei Vokalen angereichert, damit man es leichter aussprechen kann: SaMBa.

Sniffer

Kein NSA-Kollege – aber durchaus von selbigen verwendet: *Ein Sniffer (engl. to sniff, riechen, schnüffeln) ist eine Software, die den Datenverkehr eines Netzwerks empfangen, aufzeichnen, darstellen und ggf. auswerten kann. Es handelt sich also um ein Werkzeug der Netzwerkanalyse.* (Wikipedia)

Softreset

Ein „weiches“ herunterfahren des Systems, wenn nichts mehr geht – vergleichbar mit Strg-Alt-Entf am PC. Wird i. d. R. durch langes Drücken der „Power-Taste“ ausgelöst.

SSH

Das Kürzel *SSH* bezeichnet die **S**ecure **Sh**ell – was zum Einen ein Netzwerk-Protokoll, zum Anderen aber auch entsprechende Programme für die Herstellung/Nutzung einer sicheren Datenverbindung benennt. Es kommt i. d. R. eine starke Verschlüsselung zum Einsatz, die auch bereits beim Verbindungsaufbau und der Anmeldung (Login) gilt. Eine detailliertere Beschreibung kann der Wikipedia entnommen werden.

SSI

Server-**S**ide **I**ncludes bezeichnet einfache, i. d. R. in HTML-Dokumente eingebettete Skript-Befehle, welche der Webserver vor der Auslieferung der Seite an den Client interpretiert und durch entsprechende Inhalte ersetzt. Für weiterführendes, siehe Wikipedia.

SSID

Diese Abkürzung steht für den Begriff ***S**ervice **S**et **Id**entifier*, und sie bezeichnet den frei wählbaren Namen eines WLAN-Netzes. Für Details, siehe Wikipedia.

Symlink

Eine symbolische Verknüpfung, auch symbolischer Link, Symlink oder Softlink genannt, ist eine Verknüpfung in einem Dateisystem (Dateien und Verzeichnisse), die auf eine andere Datei oder ein anderes Verzeichnis verweist. Es ist also lediglich eine Referenz auf die Zieldatei bzw. das Zielverzeichnis. (Wikipedia)

Tethering

Das Bereitstellen einer bestehenden Netzverbindung für andere Geräte. In unserem Kontext handelt es sich dabei in der Regel um eine mobile Datenverbindung, die per WLAN, Bluetooth, oder USB weitergegeben wird.

TAR

Kurzform für **T**ape **Ar**chive – denn dafür wurde dieses Format ursprünglich eingesetzt. Obwohl Bandlaufwerke heutzutage weitgehend zum „alten Eisen“ gehören, ist das Format (zumindest in der Unix- und Linux-Welt) nach wie vor sehr beliebt – und kommt meist in Verbindung mit einem Kompressionsprogramm wie GZip oder BZip daher. Nähere Details lassen sich u. a. diesem Wikipedia-Artikel entnehmen.

TOR

Das hat jetzt nichts mit Fußball zu tun, und ich bin auch nicht Balla-Balla. Wer genau hinschaut erkennt, dass alle drei Buchstaben groß geschrieben sind – ganz offensichtlich handelt es sich hier also um eine Abkürzung.

Und die steht für ***T**he **O**nion **R**outer*. Zwiebel-Ruten? Gedacht ist hier an die unzähligen Schalen oder Schichten, die eine Zwiebel hat (und vielleicht auch an die Tränen, die ein Auseinandernehmen derselben verursacht). Beim TOR-Netzwerk steht jede Schicht hier für eine Erhöhung der Anonymität: Je mehr Router beteiligt sind, desto schwieriger lässt sich der Absender eines Paketes ausmachen.

Für eine genauere Beschreibung möchte ich jedoch auch hier wieder auf den zugehörigen Wikipedia-Artikel verweisen.

Traces

Die Ablaufverfolgung (englisch tracing) bezeichnet in der Programmierung eine Funktion zur Analyse von oder Fehlersuche in Programmen. […] Dabei wird z. B. bei jedem Einsprung in eine Funktion, sowie bei jedem Verlassen eine Meldung ausgegeben, sodass der Programmierer mitverfolgen kann, wann und von wo welche Funktion aufgerufen wird. (Wikipedia)

Tracker

Die Rede ist hier nicht vom „Trecker“, der laut knatternd die Landstraßen verstopft und eigentlich aufs Feld gehört. *Tracker* leitet sich vielmehr vom englischen Wort „track“, also „Spur“ ab. Ein *Tracker* in unserem Kontext versucht, Nutzerspuren aufzuzeichnen. Klassische Beispiele sind die Web-Bugs. Nicht verwechseln: Im Bereich „GPS & Navigation“ gibt es auch „Tracker“, die Spuren aufzeichnen – hier ist das aber definitiv gewünscht, um anschließend z. B. die zurückgelegte Route auf einer Karte ansehen zu können.

Tunnel

Ein *Tunnel* bündelt die Netzwerk-Kommunikation eines oder mehrerer Protokolle, und bettet diese ggf. in ein anderes Protokoll ein. Am Tunnelein- und Ausgang wird dabei das „originale Protokoll“ (z. B. HTTP zum Abruf einer Webseite) benutzt, während die Übertragung im Tunnel Verschlüsselt (beispielsweise über SSH) abläuft. Details dazu finden sich u. a. in der Wikipedia.

USB Massenspeicher

Taucht der Speicher des Androiden nach dem Verbinden mit dem Computer per USB-Kabel wie von Zauberhand als Wechseldatenträger auf letzterem auf, handelt es sich dabei um USB-Massenspeicher. Ein Teil des Androiden wird quasi zum „Kartenleser“ umfunktioniert, der Computer erhält somit „Vollzugriff“ – was sich auch zum Formatieren der Karte (oder zum Retten versehentlich gelöschter Daten) nutzen lässt. Da dies jedoch eine dedizierte Partition voraussetzt, die entweder nur am Computer oder nur auf dem Androiden bereitstehen kann, setzt Android ab Version 4.0 stattdessen auf MTP.

UMTS

Abkürzung für **Universal Mobile Telecommunications System**, auch bekannt als **3G** (Mobilfunkstandard der dritten Generation; Datenübertragungsraten bis zu 42 MBit/s).

Update.Zip

Ganz offensichtlich eine Datei. Und ebenso offensichtlich will diese etwas aktualisieren – nur was?

Es handelt sich hier um ein „flashbares Update“. Offizielle Firmware-Updates kommen meist unter diesem Namen daher. Und da das System eine solche Datei, so sie im Wurzel-Verzeichnis der SD-Karte liegt, als ein solches betrachtet, lässt sich auf diese Weise auch so einiges anderes ins System mogeln. Das nutzt z. B. Titanium Backup aus, wenn es ein update.zip erstellt.

Einspielen lässt es sich zum Beispiel über das Bootmenü.

VNC

***V**irtual **N**etwork **C**omputing, kurz VNC, ist eine Software, die den Bildschirminhalt eines entfernten Rechners (Server) auf einem lokalen Rechner (Client) anzeigt und im Gegenzug Tastatur- und Mausbewegungen des lokalen Rechners an den entfernten Rechner sendet. Damit kann man auf einem entfernten Rechner arbeiten, als säße man direkt davor.* (Wikipedia)

VPN

Ein ***V**irtuelles **P**rivates **N**etzwerk* bindet einen Computer (oder, in unserem Fall, einen Androiden) über eine verschlüsselte Verbindung in ein entferntes Netzwerk (Firmennetz, Uni-Netz, heimisches Netzwerk) ein, als wäre er „dort vor Ort“ (näheres dazu bei Wikipedia).

Web-Bug

Kleine Wanzen, die sich auf diversen Webseiten verstecken, um Besucher-Informationen zu sammeln. Mancherorts werden sie auch verharmlosend „Zählpixel“ genannt.

Man mag sich fragen: Welchen Schaden kann so ein 1x1 Pixel großes Bildchen denn anrichten? Kommt ganz darauf an. Denn beim Abruf desselbigen vom Server wird ja nicht nur diese kleine Grafik-Datei zum Benutzer übertragen. Auch in der Gegenrichtung fließen so einige Informationen: Auf welcher Webseite war das Bild eingebunden (Referrer)? Welche IP-Adresse/Browser/Betriebssystem/… hat der Anwender im Einsatz? Wo war er noch (Kekse/Cookies helfen gern dabei, Lücken zu schließen)?

Da kann so einiges zusammen kommen, und schon fast ein Profil des Anwenders erstellt werden. Mehr Details gefällig? Wie immer, hilft Wikipedia hier gern weiter.

WebDAV

WebDAV steht für *Web-based Distributed Authoring and Versioning* und ist ein offener Standard zur Bereitstellung von Dateien im Internet, über den auf Daten wie bei einer Online-Festplatte zugegriffen werden kann. Dabei ist auch eine Versionskontrolle spezifiziert. In der Regel wird *WebDAV* über einen Webserver umgesetzt, da es sich hier um eine Erweiterung des HTTP-Protokolls handelt. Nähere Details finden sich u. a. in der Wikipedia.

Wipe

Wörtlich übersetzt: (Weg)wischen, Löschen. Das „was“ ist hier allerdings die Frage. Und da kommt es darauf an, wen man fragt bzw. wie man den Wipe initialisiert.

Factory-Reset: Eine Form des Wipe ist mit diesem gleichbedeutend, denn sie löscht lediglich die Daten der `/data` sowie `/cache` Partitionen. Das ist

der Bereich, in dem die selbst installierten Apps sowie die Daten abgelegt werden.

Dalvik-Cache: Mit dem Wipe/Löschen des Dalvik-Caches erzwingt man eine Neu-Übersetzung des Programmcodes aller Apps. Damit geht kein Datenverlust einher: Lediglich der nächste Gerätestart dauert etwas länger.

Komplett-Wipe: Den gibt es in verschiedenen Versionen des Recovery-Menüs. Sollte nur ausgeführt werden, wenn man auch wirklich weiß, was man da tut: Der löscht nämlich alle Daten von allen Partitionen, auch von `/system`. Danach geht dann nichts mehr – es lässt sich lediglich ein neues ROM einspielen.

Sicheres Löschen: Auch dieses wird oftmals als „Wipe“ bezeichnet. Während sich „einfach gelöschte“ Dateien noch mit ein wenig Aufwand wieder herstellen lassen, werden sie beim „Wipe“ mehrfach mit „Datenmüll“ überschrieben – womit sie wirklich unwiderbringlich verschwunden sind.

XML

Die *e**X**tensible **M**arkup **L**anguage* stellt hierarchisch strukturierte Daten in Form von Textdaten dar, und wird u. a. für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt. Normalerweise enthalten XML-Dateien keinen Binärcode, auch wenn dies aufgrund der Erweiterbarkeit durchaus möglich ist. Daher kann man selbige notfalls auch im „Texteditor“ betrachten – wenn auch nicht unbedingt sehr komfortabel.
Weiterführende Informationen finden sich bei der Wikipedia.

Android Permissions – und was sie bedeuten

Normalerweise sieht man eine Kurzbeschreibung der Permission (z. B. bei der Installation einer App). Die technische Bezeichnung taucht selten im Klartext für den Anwender auf – man kann aber z. B. auch einen Blick auf das Manifest werfen, und da stehen sie im Klartext.

Nun werde ich aber hier nicht alle Permissions aufführen, das wäre einfach zu viel. Die findet man bei Interesse im Entwickler-Handbuch (allerdings auf Englisch). Ein Wiki zum Thema, an dem sich jeder beteiligen kann, existiert ebenfalls, und zwar bei Stack Exchange (wiederum auf Englisch), eine weitere gute (englische) Übersicht mit zusätzlichen Sicherheits-Tipps existiert bei AndroidForums.Com, und auch bei Go2Android findet sich ein gut erklärter Artikel von MaTT (übrigens Teil der Serie „anDROID für Anfänger“). Eine erklärte und ständig aktualisierte Übersicht, wer hätte das gedacht, gibt es natürlich auch bei IzzyOnDroid. Ein paar ausgewählte, die doch häufiger einmal auftauchen könnten, möchte ich aber hier kurz erklären.

Permission Groups

Zunächst einmal sind die Permissions in „Berechtigungs-Gruppen“ eingeteilt. Besucht man die Playstore-Seite einer App, findet man diese „Permission Groups“ als Überschriften unter „Diese App möchte auf folgendes zugreifen:“ wieder. Eine vollständige Auflistung gibt es in der API Dokumentation auf den Entwicklerseiten in englischer Sprache, sowie auf der gerade genannten Seite bei *IzzyOnDroid* auf Deutsch. Jede dieser Gruppen fasst zugehörige Berechtigungen zusammen – so etwa „ACCOUNTS“ (Konten) die Permissions für den direkten Zugriff auf vom Account Manager verwaltete Konten, „CALENDAR“ alles, was auf den Kalender zugreift, „LOCATION“ die für den Standort-Zugriff zuständigen Dinge, oder „NETWORK“ mit dem Netzwerk-Zugriff zusammenhängende Permissions.

Protection Level

Eine weitere Einteilung stellen die so genannten *Protection Level* dar (siehe Developers Reference: Permission Element). Fünf verschiedene Ebenen legen dabei fest, wie mit den entsprechenden Permissions umgegangen wird (die Kürzel in Klammern sind für die Referenz in der eigentlichen Permissions-Tabelle):

- **normal (no):** niedriges Risiko bzw. „Standard“. Diese Permissions werden dem Anwender bei der Installation nicht extra „vorgeführt“, sondern stillschweigend akzeptiert.

- **dangerous (da):** höheres Risiko: Zugriff auf persönliche Daten, oder Kontrolle über das Gerät mit potentiellem negativen Impakt für den User. Das sind die Permissions, die bei der Installation vom Anwender explizit bestätigt werden müssen.
- **signature (si):** Der Herausgeber des Zertifikats muss mit dem des ROM-Zertifikats übereinstimmen. In der Regel heißt dies: Eine solche App muss vom gleichen Hersteller stammen.
- **signatureOrSystem (sy):** Wie „Signature“; alternativ darf es aber auch eine beliebige „System-App“ sein. Da man eine solche nur mit root-Rechten installieren kann, ist dies für „normale Anwender“ eher uninteressant.
- **development (dv):** Das ROM muss mit einem Entwickler-Key signiert sein. Diese Permissions sind eigentlich nur für Entwickler/Entwicklung gedacht. I. d. R. scheinbar als „developmentOrSignatureOrSystem“ zu verstehen.

Permissions

Auch hier wieder ein Auszug einiger gerbäuchlicherer Kandidaten. Nochmals kurz zur Erklärung: „PL“ bezeichnet den gerade beschriebenen Protection Level. „Risk“ steht für das mit der jeweiligen Permission verbundene potentielle Risiko, sofern sich das grob abschätzen lässt:

- 0 = niedrig
- 1 = moderat
- 2 = medium
- 3 = hoch
- 4 = sehr hoch
- 5 = kritisch

Dieses Risiko muss jedoch auch immer im Zusammenhang mit dem Protection Level gesehen werden: Von einer „kritischen Permission“ ist der Normal-Anwender (ohne root) beispielsweise überhaupt nicht betroffen, wenn der Protection Level „Signature“ bzw. „SignatureOrSystem“ heißt (es sei denn, es handelt sich um eine vorinstallierte System-App). Gelegentlich ändert sich eine solche Zuordnung mit neueren Android-Versionen (Beispiel: `READ_LOGS` wanderte ab Android 4.1 nach „sy“). Quelle für die Risiko-Klassen ist hier übrigens der bereits genannte Artikel bei AndroidForums.Com, die meisten Protection-Level finden sich in einem Blog-Eintrag zugeordnet. Folgende Tabelle beschreibt einige der häufiger verwendeten Permissions:

Permission	PL	Risk	Erklärung
ACCESS_COARSE_LOCATION	da	?	*Ungefährer (netzwerkbasierter) Standort*. Hier kommt kein GPS zum Einsatz, sondern die Informationen von Funkmasten (Cell-ID) sowie WLANs
			Genauer (GPS-) Standort. Exakte, per GPS

ACCESS_FINE_LOCATION	da	?	ermittelte Standortdaten.
ACCESS_NETWORK_STATE	no	?	*Netzwerkstatus anzeigen*. Informationen über Netzwerke (besteht eine Verbindung, und wenn ja zu welchem Netzwerk?)
ACCESS_WIFI_STATE	no	?	*WLAN-Status anzeigen*. Informationen über WiFi-Netzwerke (besteht eine Verbindung, und wenn ja zu welchem Netzwerk? Welche Netzwerke sind verfügbar?)
ACCOUNT_MANAGER	sy	?	*Als Konto-Manager fungieren*. App darf mit Konto-Authentifizierern interagieren. (für Systemanwendungen reserviert).
AUTHENTICATE_ACCOUNTS	da	4	*Als Kontoauthentifizierer fungieren*. Konto-Authentifizierungsfunktionen verwenden, Konten erstellen, Abrufen und Einstellen der zugehörigen Passwörter. In der Regel stellt eine solche App eine Schnittstelle zu einem neuen Dienst bereit, der nicht von Haus aus in Android integriert ist (Beispiel: Dropbox) – implementiert also die Art und Weise, wie bei diesem die Anmeldung funktioniert. Darüber hinaus kann die App u. U. auch einschränken, was eine aufrufende App mit dem Zugang anstellen darf.
BLUETOOTH	da	?	*Bluetooth-Verbindungen herstellen*. Zugriff auf bereits „autorisierte“ Bluetooth-Geräte
BLUETOOTH_ADMIN	da	2	*Bluetooth-Verwaltung*. Bluetooth-Geräte „autorisieren“ (also „pairen“ und so). Eine mit dieser Permission ausgestattete App darf selbständig Bluetooth-Verbindungen aufbauen und etablieren – auch zu „wildfremden“ Geräten.
CALL_PHONE	da	3	*Telefonnummern direkt anrufen*. Anruf *ohne Bestätigung durch den Anwender* tätigen. Wird z. B. für Kontakt-Widgets benötigt, wenn ein „Tapp“ auf selbige direkt einen Anruf auslösen soll – macht aber bei einer Mal-App herzlich wenig Sinn. Gilt nicht für Notruf-Nummern.
CALL_PRIVILEGED	sy	?	*Alle Telefonnummern direkt anrufen.* Wie CALL_PHONE, aber inklusive Notruf-Nummern („Hallo, Polizei – Anwender ist gerade in Bank eingebrochen. Bitte Fußboden reparieren …“)
CAMERA	da	1-3	*Fotos aufnehmen*. Vollzugriff auf die Kamera. Nebenwirkung: Diese App lässt sich nicht auf Geräten installieren, die über keine Kamera verfügen. Die API-Referenz schreibt sinngemäß: „Wenn die App auch

			ohne Kamera bedienbar ist, diese Permission nicht anfordern.“ Da sei die Frage erlaubt: Braucht man sie dann überhaupt, wenn es auch ohne geht?
CHANGE_NETWORK_STATE	da	?	*Netzwerkkonnektivität ändern*. Netzwerk-Status ändern (also z. B. Verbindung trennen)
CHANGE_WIFI_MULTICAST_STATE	da	?	*WLAN-Multicast-Empfang zulassen*. WiFi MultiCast aktivieren. Damit können Datenpakete an mehrere Empfänger zeitgleich verschickt werden, ohne dass dies zusätzliche Bandbreite erfordert. Macht z. B. Sinn bei einem Streaming-Server, der mehrere Clients bedient („Radio“). Gleichzeitig ermöglicht dies auch den Empfang von Netzwerk-Paketen, die nicht an das eigene Gerät gerichtet sind (Netzwerk-Sniffer).
CHANGE_WIFI_STATE	da	?	*WLAN-Status ändern*. CHANGE_NETWORK_STATE für WiFi. Kann auch Änderungen an konfigurierten WLAN-Netzen vornehmen.
CLEAR_APP_USER_DATA	sy	?	*Alle Cache-Daten der Anwendung löschen*. Benutzerdaten beliebiger/aller Apps löschen (siehe *Einstellungen › Anwendungen verwalten*, der Button „Daten löschen“ bei jeder Anwendung) – richtiger wäre also *Anwendungsdaten* löschen oder CLEAR_APP_DATA, das „User“ verwirrt hier ein wenig.
DISABLE_KEYGUARD	da	2-3	*Tastensperre deaktivieren*. Tastensperre (inkl. deren Passwort-Schutz) deaktivieren, sodass der Bildschirm nicht mehr automatisch gesperrt wird. Sinnvoll z. B. bei Video-Apps und insbesondere bei Navis – und bei eingehenden Telefonaten.
EXPAND_STATUS_BAR	no	2-3	*Statusleiste ein-/ausblenden*. Status-Bar (Notification?) erweitern/kollabieren. Wohl die Lite-Version von STATUS_BAR
GET_ACCOUNTS	no	?	*Bekannte Konten suchen*. Liste konfigurierter Accounts abrufen (nur die Accounts, nicht die Zugangsdaten selber). Mit dieser Permission lässt sich lediglich feststellen, welche Accounts existieren. So kann beispielsweise eine App, die Dropbox verwenden möchte, feststellen, ob bereits ein passendes Zugangskonto eingerichtet ist.
			Laufende Anwendungen abrufen.

GET_TASKS	da	2-3	Informationen über laufende Anwendungen abrufen. Wird natürlich von Task-Managern und -Killern, aber auch von Akku-Statistik-Apps benötigt. „Böse Apps“ können dies nutzen um auszukundschaften, wo sich lohnende Daten zum Klauen finden lassen.
INSTALL_PACKAGES	sy	?	*Anwendungen direkt installieren*. Andere Apps installieren. Kann OK sein (App-Manager), muss aber nicht (Wallpaper etc. wollen vielleicht eher Schadsoft nachladen, wenn sie diese Permission anfordern)
INSTALL_SHORTCUT	?	1-3	*Verknüpfungen auf dem Homescreen erstellen.* Malware nutzt dies, um unerwünschte Dinge dort abzulegen. Beispielsweise ein Icon, dass wie das vom Playstore aussieht – allerdings ganz woanders hinführt.
INTERNET	da	?	*Uneingeschränkter Internetzugriff*. Öffnen von Netzwerk-Sockets. Die App kann also beliebige Internet-Verbindungen herstellen. Wird von allen Apps gebraucht, die Werbung anzeigen wollen.
KILL_BACKGROUND_PROCESSES	no	3	*alle Anwendungen im Hintergrund schließen*. Hintergrund-Prozesse „töten“, also beenden. Dabei kann es sich um die eigenen Prozesse handeln (was dem Anwender die Möglichkeit gibt, das Programm tatsächlich zu beenden, statt es nur in den Hintergrund zu schieben) – es können aber eben so gut fremde Prozesse beendet werden. i. d. R. handelt es sich dann um einen Task-Manager oder Task-Killer. Eine böswillige App könnte dies jedoch auch nutzen, um Schutzmechanismen auszuhebeln (etwa eine Anti-Malware-App abzuschießen, bevor sie mit ihrem eigentlichen Unwesen beginnt).
MANAGE_ACCOUNTS	da	?	*Als Konto-Manager fungieren*. Accounts/Zugangsdaten *verwalten* – also auch verändern. Die Doku ist leider wieder einmal sehr vage. Laut einem Post bei Stack Exchange heißt dies jedoch nur, dass die betreffende App ihre eigenen Konten (nicht aber andere) mit Unterstützung des (system-eigenen) Account-Managers verwalten darf.
			Telefonstatus ändern. *Status* der Telefonie anpassen: Power, MMI-Codes (z. B. Rufumleitung [de]aktivieren,

MODIFY_PHONE_STATE	sy	?	Rufnummernübermittlung ein/ausschalten) etc. – jedoch nicht Anrufe tätigen. Allerdings kann das Netzwerk (zu einem anderen Anbieter, Roaming) gewechselt oder die Mobilfunkverbindung ein- bzw. ausgeschaltet werden, ohne dass der Benutzer davon informiert wird. Auch können mit dieser Permission eingehende Anrufe abgefangen werden.
READ_ATTACHMENT	da	3	*Attachments lesen.* Bezieht sich auf Dateianhänge in E-Mails der Stock-Mail-App (trifft also weder auf die GMail-App, noch auf K-9 Mail zu). Dateianhänge können sensible Informationen enthalten, die eine böswillige App an andere Stellen weiterleiten könnte.
READ_CALENDAR	da	2	*Kalenderdaten lesen*. Sollte klar sein: Alle Termine können damit gelesen werden.
READ_CONTACTS	da	2-3	*Kontaktdaten lesen*. Damit ist das Adressbuch fällig. Was damit alles einsehbar ist, verrät die App permission.READ_CONTACTS.
READ_HISTORY_BOOKMARKS	da	2-3	Erlaubt lesenden Zugriff auf Lesezeichen und Browserverlauf (Chronik). Was das im Einzelnen bedeutet. lässt sich mit der READ_HISTORY_BOOKMARKS App ermitteln.
READ_OWNER_DATA	da	?	*Eigentümerdaten lesen*. Auslesen der auf dem Gerät gespeicherten Eigentümerdaten.
READ_PHONE_STATE	da	?	*Telefonstatus lesen und identifizieren*. Zugriff auf die Telefonfunktionen des Gerätes. Eine Anwendung erhält mit dieser Berechtigung unter anderem die Möglichkeit, die Telefon- und Seriennummer des Telefons zu ermitteln. Um festzustellen, ob ein Anruf aktiv ist,wird sie jedoch trotz gegenteiliger Behauptungen keineswegs benötigt, wie die App permission.READ_PHONE_STATE zeigt. Bei Werbung (z. B. AdMob) wird dies häufig zum Auslesen der IMEI/IMSI genutzt um festzustellen, welche Werbung auf dem Gerät bereits angezeigt wurde (eindeutige Identifizierung, Tracking – seit August 2014 darf die IMEI laut Playstore-Richtlinien dafür nicht mehr verwendet werden, und schon stoßen auch etliche Apps diese Permission ab …). Apps, die auch für Android 1.6 und früher kompatibel sein sollen, wird diese Permission automatisch

			gesetzt.
READ_PROFILE	da	2-3	*Das persönliche Profil des Anwenders lesen.* Bezieht sich auf den neuen „Me“ Kontakt für das eigene Profil. Macht beispielsweise Sinn für Messenger-Apps, die den Diskussionsverlauf darstellen.
READ_SMS	da	1-3	*SMS oder MMS lesen*. Damit lassen sich bereits gespeicherte Kurznachrichten lesen. Darunter können natürlich auch vertrauliche Informationen sein.
READ_SOCIAL_STREAM	da	3	*Den Social-Media-Stream lesen.* Diese Permission wurde mit Android 4.0 eingeführt. Eine damit ausgestattete App kann Status-Updates der sozialen Netzwerke lesen – sowohl eingehende, als auch ausgehende.
RECEIVE_BOOT_COMPLETED	no	1-3	*Automatisch nach dem Booten starten*. App möchte benachrichtigt werden, wenn der Bootvorgang abgeschlossen ist. i. d. R. heißt das: Sie möchte nach dem Booten automatisch gestartet werden.
RECEIVE_MMS, RECEIVE_SMS	da	3	*MMS empfangen, SMS empfangen*. Eingehenden MMS/SMS abfangen – da möchte wohl jemand mitlesen. Kann aber durchaus OK sein, wenn die App auf MMS/SMS reagieren *soll*. Auf der anderen Seite kann man damit eingehende Nachrichten auch „im Nirvana“ verschwinden lassen.
RECORD_AUDIO	da	1-3	*Audio aufnehmen*. Tonaufnahmen erstellen. Das kann sowohl für ein „Diktaphon“ genutzt werden – als auch zum Mitschneiden von Telefonaten.
SEND_SMS	da	3	*Kurznachrichten senden*. Und zwar ohne Zutun des Benutzers, auch an richtig teure Premium-Dienste (womit klar ist, wozu „böse Apps“ das gern hätten). Es gibt aber auch „gute“ Gründe für diese Permission: Natürlich die SMS-Apps, aber teilweise auch In-App-Käufe, die nicht über Google Checkout abgewickelt werden.
STATUS_BAR	sy	?	Kann die Status-Bar (Notification?) öffnen, schließen, und ausblenden. Meist will die App wohl letzteres, um einen „Vollbild-Modus“ zu ermöglichen.
			Warnungen auf Systemebene anzeigen. Fenster mit Systemwarnungen einblenden. Eine böswillige App kann so den gesamten Bildschirm blockieren. Sollte eigentlich nicht

SYSTEM_ALERT_WINDOW	da	3	benutzt werden, da dies für System-Meldungen gedacht ist. Erlaubt das anzeigen von „Alert Windows“, d. h. Nachrichtenfenstern, die immer im Vordergrund angezeigt werden.
USE_CREDENTIALS	da	?	*Authentifizierungsinformationen eines Kontos verwenden*. Möchte die konfigurierten Zugangsdaten verwenden. Das heißt nicht unbedingt, dass es sie „zu sehen bekommt“ – aber die App kann sich quasi „im Namen des Anwenders“ anmelden. Beim ersten Mal wird der Anwender jedoch vom Account Manager gefragt, ob er dies zulassen möchte.
USE_SIP	da	2-3	App kann Internet-Telefonie nutzen (SIP ist das Protokoll dafür)
WAKE_LOCK	no	?	*Standby-Modus deaktivieren*. App kann das System daran hindern, einen Ruhezustand einzunehmen (also den Bildschirm zu dimmen, die CPU „schlafen“ zu lassen, etc.) Wäre doch blöd, wenn die Navi-App läuft und plötzlich der Bildschirm ausgeht.
WRITE_APN_SETTINGS	sy	?	*Einstellungen für Zugriffspunktname schreiben*. App kann die Zugangsdaten zum Internet etc. (siehe APN) verändern. Meist geht es der App nur darum, den Namen des APN zu ändern – um die Verwendung des mobilen Datenverkehrs zu steuern (Beispiel: APNDroid).
WRITE_CALENDAR	da	2	*Kalenderdaten schreiben*. Diese Permission erlaubt lediglich den Schreib-, nicht aber den Lesezugriff auf den Kalender. Die damit versehene App kann also Termine hinzufügen, nicht aber lesen oder ändern.
WRITE_CALL_LOG	da	2-3	*Anruflisten schreiben.* Macht sicher Sinn für VoIP- und Backup-Apps. Andere Apps sollten hier jedoch nichts verloren haben.
WRITE_CONTACTS	da	1-3	*Kontaktdaten schreiben*. Wie WRITE_CALENDAR, nur in Bezug auf die Kontaktdaten bzw. Browser-History (Chronik) und Lesezeichen.
WRITE_HISTORY_BOOKMARKS	da	2-3	
WRITE_EXTERNAL_STORAGE	da	2	App darf beliebige Daten auf der (externen) SD-Karte lesen, schreiben, verändern und auch löschen – prinzipiell auch die Daten anderer Apps. Diese Permission ist aber beispielsweise essentiell für diverse Backup- und Kamera-Apps, die natürlich Daten auf der Karte manipulieren müssen. **Warnung:** Apps, die für Android 1.5 oder

			älter geschrieben wurden, erhalten diese Permission implizit!
WRITE_OWNER_DATA	da	?	*Eigentümerdaten schreiben*. Schreiben/verändern der auf dem Gerät gespeicherten Eigentümerdaten.
WRITE_PROFILE	da	1-3	*Das persönliche Profil des Anwenders schreiben.* Gegenstück zu READ_PROFILE.
WRITE_SECURE_SETTINGS	dv	4	*Allgemeine Systemeinstellungen ändern*. Lesen und Schreiben von Systemeinstellungen. „Secure Settings“ können nur von Systemanwendungen (also solchen, die ins „ROM“ integriert wurden) angefordert werden.
WRITE_SETTINGS	da	2	
WRITE_SMS	da	3	*SMS Nachrichten schreiben* (jedoch nicht senden). Eine böswillige App könnte darauf hoffen, dass der Anwender die „ungesendete Nachricht“ dann doch noch auf den Weg bringt.
WRITE_SYNC_SETTINGS	da	2	*Synchronisierungseinstellungen schreiben*. Schreibzugriff auf die Einstellungen der Synchronisation. Eine mit dieser Permission ausgestattete App kann u. a. die Synchronisation von Kontakten und Kalendern aktivieren bzw. deaktivieren. Sinn macht so etwas u. U. bei einer SMS Backup App.
com.android.vending.BILLING	?	5	In-App Payment (in der App integrierte Bezahldienste, die über den *Play Store* abgewickelt werden)

Wer noch immer „Bahnhof“ versteht, dem sei ein kleines Tutorial bei N-Droid.DE empfohlen. Etwas tiefgreifender wird das Thema bei LifeHacker beschrieben. Für Anfänger gut verständlich, sofern sie keine Probleme mit der englischen Sprache haben – definitiv empfohlene Lektüre!

Diebstahl-Schutz mit Tasker

Schrieb ich doch im Kapitel Diebstahlschutz, ein solcher ließe sich eventuell auch mit Tasker realisieren? In der Tat ist dies möglich. Eine Variante des „Device-Trackings“ (aka „Wo ist mein Androide?“) stellt Kevin Purdy in einem Artikel auf Lifehacker.COM mit einer bebilderten Schritt-für-Schritt-Anweisung vor. Zwar auf Englisch – aber dank der guten Bebilderung sollten schon geringfügige Sprachkenntnisse ausreichend sein, um dem How-To folgen zu können.

Damit wäre die Lokalisierung eines abhanden gekommenen Androiden abgedeckt. Damit dies nicht in der Absicht eines „bösen Menschen“ geschieht (sondern allenfalls wir das Gerät „irgendwo liegen gelassen haben könnten“), hier noch ein kleiner Anreiz zur Umsetzung eines „Diebstahl-Alarms“. Selbiger soll losgehen, wenn ein Unbefugter sich unseres Androidens bemächtigt – nicht aber, wenn wir selbst ihn in die Hand nehmen. Dazu könnte wie folgt vorgegangen werden:

1. **Profil 1: Bildschirm wurde entsperrt**
   - Bedingung: Event → Display → Display unlocked
   - Task: Variable → Variable Set, U_UNLOCKING = 1
2. **Profil 2: Alarm deaktivieren**
   - Bedingung 1: Event → Hardware → Button: Camera
   - Bedingung 2: State → Variable Value U_UNLOCKING > 0
   - Task: Variable → Variable Set, U_UNLOCKING = 0
3. **Profil 3: Alarm-Count-Down auslösen**
   - Bedingung: State → Variable Value U_UNLOCKING > 0
   - Task: XXX

Bei „XXX“ ist natürlich die entsprechende Aktion einzusetzen. Beispielsweise mit einer While-Schleife 10 Sekunden warten, ob doch noch entschärft wird – und andernfalls anschließend ein entsprechendes MP3 abspielen, welches laut „DIEBE! ICH BIN EIN GEKLAUTES TELEFON!“ brüllt, und dabei Polizeisirene, „Roten Alarm“ der Enterprise, o. ä. einblendet.

Wie ist das Ganze gedacht? Über eine Variable (U_UNLOCKING) schauen wir, ob überhaupt etwas passieren muss (ist sie ungleich 0?). Dazu wird sie bei Entsperrung des Bildschirms (ggf. durch andere Events wie „Gerät bewegt“ ersetzen) auf 1 gesetzt – dies erledigt „Profil 1“.

Wird nun beim/kurz nach dem Entsperren des Bildschirms der Kamera-Button gedrückt, gibt „Profil 2“ Entwarnung (setzt U_UNLOCKING=0; diese Aktion kann natürlich gegen einen beliebigen anderen Event ausgetauscht werden: Etwa Umdrehen oder Schütteln des Androiden). Passiert dies nicht, kümmert sich „Profil 3“ um die entsprechende Alarm-Aktion.

Dies ist ein rein theoretischer Ansatz – sollte das jemand einmal umgesetzt

haben, freue ich mich auf entsprechendes Feedback!

PS: Sollte das Gerät sich dennoch unerlaubt entfernt haben, so zeigt ein Video-Tutorial bei Youtube, wie sich auch die Ortung desselben mittels *Tasker* realisieren lässt.

Secret Codes oder Magische Nummern

Klar kann man mit einem Telefon telefonieren. Dazu gibt man eine Ziffernfolge ein, und drückt die Taste für „Abheben“. Was aber passiert, wenn man noch ein paar Sonderzeichen hinzufügt?

Beschränken wir uns dabei auf die Zeichen `#` und `*` erhalten wir – richtig kombiniert – so genannte GSM-Codes. Der verlinkte Wikipedia-Artikel beschreibt ganz gut, worum es dabei geht (kurz gefasst: Steuer-Codes für diverse Netzanbieter-Funktionen, die eigentlich auf allen Geräten gleich funktionieren sollten, aber nicht bei allen Anbietern in gleichem Umfang verfügbar sind). Unterteilen lassen sich diese Codes grob in mehrere Untergruppen:

USSD-Codes:

Diese folgen dem Muster `*1nn#` – also der * Taste, gefolgt von einer 1 und weiteren zwei Ziffern, abgeschlossen durch eine Raute (optional mit Parametern: `*1nn*<Parameter>#`). Sie stellen Zugangsnummern für einfachere Mobilfunkdienste dar, und stellen zum Beispiel Zugang zu vorkonfigurierten Diensten bereit, die für den Betreiber des jeweiligen Mobilfunknetzes spezifisch sind. Gibt man einen solchen USSD-Code auf dem Mobilfunk-Gerät ein, wird die Antwort des Betreibernetzes normalerweise innerhalb weniger Sekunden auf dem Bildschirm dargestellt.

Erweiterte Service-Codes:

Diese werden für erweiterte Service-Dienste wie etwa Anrufweiterleitungen oder die (De-)Aktivierung der Rufnummernübermittlung, aber auch zum Ändern bzw. Entsperren der PIN genutzt.

Geräte-, Hersteller- und Systemspezifische Codes:

Diese dienen i. d. R. dem zuständigen Service-Personal zur Abfrage/Anpassung diverser Geräte-Parameter, für Status-Tests, u. a. m.

Disclaimer: Naturgemäß kann es sein, dass einige dieser Codes nicht auf allen Geräten bzw. nicht bei allen Mobilfunkanbietern funktionieren. Sofern sie nicht im Handbuch des Gerätes aufgeführt sind, mag das durchaus seine Gründe haben: So kann der eine oder andere Punkt, unsachgemäss angewendet, u. U. Schäden verursachen. Ich übernehme keinerlei Garantien dafür, dass folgende Codes a) funktionieren, b) das tun, was da steht, oder c) “folgenfrei” genutzt werden können. Insbesondere übernehme ich keinerlei Verantwortung für etwaige negative Folgen! Für etwaige positive Folgen stelle ich natürlich gern mein Bankkonto zur Verfügung …

Ebenso kann es vorkommen, dass nach Eingabe des einen oder anderen Codes nichts passiert. Es kann aber auch sein, dass dies nur so scheint (bei meinen Tests fand ich anschließend – am nächsten Tag – einige der „magischen

Nummern“ in der „Verbraucherliste“ wieder). Es können hier durchaus Hintergrund-Dienste gestartet oder aber „versteckte Klassen/Menüs“ in Apps freigeschaltet werden.

USSD Codes

Code	Bedeutung
*100#	Prepaid-Guthaben anzeigen
*135#	Eigene Rufnummer anzeigen

Erweiterte Service-Codes

Code	Bedeutung
**21*<Rufnummer># / *21# / #21# / *#21#	Anrufweiterleitung einrichten / aktivieren / deaktivieren / überprüfen
*30# / #30# / *#30#	Eingehende Rufnummern anzeigen / unterdrücken / Status
#31#<Rufnummer>	Mit unterdrückter Rufnummer anrufen
*43# / #43# / *#43#	Anklopfen aktivieren / deaktivieren / Status abfragen
*76# / #76# / *#76#	COLP: Wenn der ausgehende Anruf weitergeleitet wird, Zielrufnummer anzeigen
*77# / #77# / *#77#	COLR: Bei eingehendem umgeleiteten Anruf die Ursprungsnummer anzeigen
*N# (TCom: *T#)	Wird eine SMS mit dieser Zeichenfolge begonnen, erfolgt eine SMS Empfangsbestätigung

Geräte-, Hersteller-, und Systemspezifische Codes

Die fett gedruckten Codes sollten nach Informationen bei StackExchange auf allen Android-Geräten (zumindest mit Android Version 4.1) gleichermaßen funktionieren. Ob das “Spielen” damit deswegen unbeding ratsam ist (zumal wenn man nicht genau weiß, was sich dahinter verbirgt), steht auf einem anderen Blatt. Daher habe ich die Codes, die mir als “absolut harmlos” bekannt sind, einmal kursiv hervorgehoben.

Code	Bedeutung
##0*#*#*	Test des LCD-Displays
*#0011#	GSM-Infos
*#0228#	ausfühliche Infos zum Barreriestatus
##0283#*#*	Loopback-Test
**05*<PUK Code>*<neue PIN>*<neue PIN zur Bestätigung>#	Entsperren des Telefons aus dem Notruf-Modus
##0588#*#*	Test des Annäherungs-Sensors

#06#	IMEI
##0673#*#* *#*#0289#*#*	Einer der beiden Codes führt zu einem Audio-Test
*#0782#	RTC (Real-Time-Clock) auslesen und anzeigen
##0842#*#*	Test für Vibrator (oh ja!) und Hintergrund-Beleuchtung
##1111#*#*	FTA Software Version
##1234#*#*	Firmware-Info
##1472365#*#*	ein kurzer GPS-Test (und Einstellungen)
##1575#*#*	ein weiterer GPS-Test
##197328640#*#*	Service-Menü mit verschiedenen Test-Möglichkeiten
##2222#*#*	FTA Hardware Version
#*#225#*#	Kalender-Debug
*#2263#	Bandnutzung
##232331#*#*	BlueTooth-Test
##232337#*#	MAC-Adresse des BlueTooth-Interfaces
##232338#*#*	MAC-Adresse des WLAN-Interfaces anzeigen
##232339#*#* *#*#526#*#* *#*#528#*#*	Einer dieser Codes führt zu den WLAN-Tests…
##2432546#*#* (*#*#CHECKIN#*#*)	Nach OTA-Updates suchen
##2663#*#*	Touch-Screen Version anzeigen
##2664#*#*	Touch-Screen Test
##273283*255*663282*#*#*	angeblich für eine schnelle Sicherung der Medien-Dateien (Fotos, Videos..) gut. Von wo und nach wo? Habe ich nicht probiert…
*2767*3855#	Factory Format (Wipe). Vorsicht damit!!!
##3264#*#*	RAM Test / RAM-Version anzeigen
##3424#*#*	HTC Function Test
##34971539#*#*	Kamera-Menü mit folgenden Punkten: Update mit Firmware aus Bild (keinesfalls! Sonst futsch!); Update mit Firmware von SD-Karte; Versions-Info; Update-Counter
##36245#*#*	EMail Debug
##44336#*#*	Herstellungszeitpunkt und Laufende Nummer
#*#4636#*# ***(*#*#INFO#*#*)***	Öffnet ein Menü, aus dem sich wählen lässt: Telefon-Infos, Akku-Infos, Akku-History, Verbrauchsstatistiken.
#4646#	Feldtest (Details zu Akkustand + Funknetz, Wechsel UMTS/GSM)
##4986*2650468#*#*	Diverse Hardware-Infos
##564#*#*	QXDM (Qualcomm eXtensible Diagnostic Monitor) Logging FrontEnd
##7262626#*#*	Ein Bett im Kornfeld… äh, Feld-Test
##7269#*#*	Standard Device-Logging (Device [logcat?], AT-Befehle, Kernel [dmesg?] — HTC?)
##7378423#*#* *#*#SERVICE#*#*	Ein weiteres Service-Menü: Service-Information, Service-Settings, Service-Tests…

*#7465625#	Netz- und Subnetzsperre, Simlook, Service Provider und Corporate Lock (Galaxy-S?)
#*#759#*#	GooglePartnerSetup
##7594#*#*	Verhalten des Einschalt-Knopfes ändern (z. B. direktes Abschalten ohne Menü)
##7780#*#*	Zurücksetzen auf Werkseinstellungen
#*#8255#*#	GoogleTalk Service Überwachung
#*#8350#*#	Logging der Anrufe deaktivieren
#*#8351#*#	Logging der Anrufe aktivieren
*#9090#	Service-Modus UART/USB
##9696#*#*	FTP Test

Weitere spezielle magische Nummern gibt es für Geräte von …

- … LG
- … Motorola
- … Samsung
- … Samsung (SGS)

Und dann wären da noch diverse Listen mit GSM und *USSD* Codes …

- … bei den XDA-Developers
- … bei SMSMich.DE (auch Provider-spezifische Codes)
- … und sicher noch an vielen anderen Stellen.

Leistungsaufnahme verschiedener Komponenten

Heise hat für seinen Artikel Energiesparplan ein Motorola Milestone angepasst und ausführlich getestet, was welche Komponente so verbraucht. Und wie man das Einschränken kann (das war jetzt eine klare Empfehlung, den Artikel zu lesen!). Die Daten davon werden einerseits ein „war ja klar“, aber bei einigen Daten auch ein „oh, das hätte ich jetzt nicht gedacht“ hervorrufen. Passende Angaben zum Galaxy S3 hat die c’t in ihrer Ausgabe vom August 2012 gesammelt – und den Test gleich noch ein wenig ausgeweitet – und dabei den Test gleich noch ein wenig ausgeweitet, um sie schließlich im c’t Special Android (Januar 2016) auch noch um die Werte des Galaxy S5 zu erweitern.

Anmerkungen zum Galaxy S3/S5 (*): Der Energieverbrauch des Displays hängt hier auch stark von den dargestellten Inhalten ab, was auf das verwendete Amoled-Display zurückzuführen ist. Bei vollständig schwarzem Display entspricht der Verbrauch auf allen Stufen nicht mehr als dem Minimum.

Anmerkungen zu den Messwerten des Galaxy S5: Anders als bei den vorigen Messreihen, wurde hier die Grundlast nicht abgezogen. Im Folgenden habe ich das (nach Rücksprache mit dem Redakteur) entsprechend angepasst, damit es vergleichbar bleibt. Die „nackten Zahlen“ weichen daher geringfügig von denen des Artikels ab.

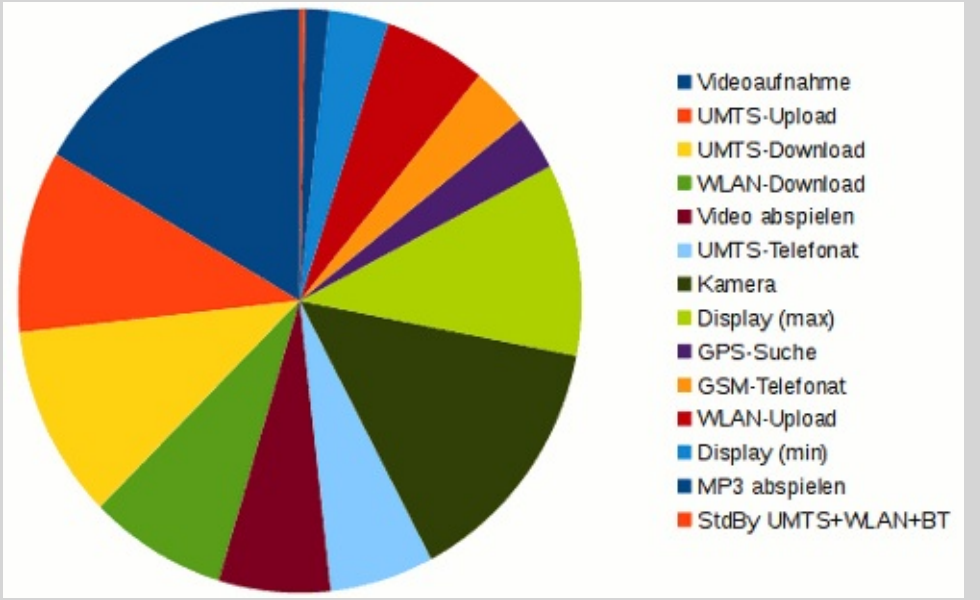
Weiterhin sollte man beim Vergleich der Werte auch die unterschiedliche Hardware-Ausstattung im Hinterkopf behalten: So werkelt beispielsweise im Galaxy S3 ein mit 1,4 GHz getakteter Quad-Core Prozessor, und zur Anzeige dient ein 4,8 Zoll Display – im Milestone waren es noch eine Single-Core CPU mit 550 MHz und ein 3,7 Zoll Display.

Betriebszustand	**zusätzliche Leistungsaufnahme**			
	Motorola Milestone	**Samsung Galaxy S3**	**Galaxy S5**	**ø**
Videoaufnahme[1]	1557 mW	1683 mW	2277 mW[4]	1839 mW
UMTS Upload	1410 mW	1033 mW	908 mW	1117 mW
UMTS Download	1349 mW	1074 mW	1138 mW	1187 mW
EGDE Upload	1179 mW			
WLAN Download	1158 mW	549 mW	906 mW[5]	871 mW
Video abspielen (fullscreen)[1]	1135 mW	597 mW	395 mW[6]	709 mW
UMTS-Telefonat	983 mW	637 mW	362 mW	660 mW
Kamera[1]	934 mW	1460 mW	2335 mW[7]	1576 mW
EGDE Download	853 mW		635 mW	
Bluetooth empfangen	751 mW	487 mW		
Display (höchste Stufe)	730 mW	1568 mW*	1227	1175

			mW*	mW
GPS Suche	550 mW	263 mW	191 mW	335 mW
GSM-Telefonat	511 mW	297 mW	310 mW	373 mW
Bluetooth senden	487 mW	454 mW		
WLAN Upload	479 mW	488 mW	987 mW[8]	651 mW
Display (niedrigste Stufe)	310 mW	567 mW	260 mW*	379 mW
WLAN Tether [2]		372 mW	542 mW[9]	
MP3 abspielen über Bluetooth		296 mW	255 mW	
MP3 abspielen	160 mW	153 mW	140 mW	151 mW
UMTS Standby	18,3 mW	13,8 mW	16,2 mW	16,1 mW
GSM/EDGE Standby	11,6 mW	9.5 mW	15 mW	12 mW
WLAN Standby 2,4 GHz	7,8 mW	9.3 mW	18 mW	11,7 mW
WLAN Standby 5 GHz	N/A	14.6 mW	20 mW	
NFC Standby	N/A	4 mW	0,1 µW	
Bluetooth Standby	2,8 mW	1.8 mW	2,9 mW	2,5 mW
GPS Standby	0,4 mW	0.7 mW	0,2 mW	0,4 mW
WLAN Tether Download[3]		1254 mW	1871 mW	

[1] Leistungsaufnahme des Displays bereits abgezogen
[2] Tethering aktiv, 1 Benutzer
[3] Download vom Notebook per WLAN-Tether
[4] Hauptkamera, Full-HD; bei 4K wären es 4079 mW
[5] im 2.4 GHz Band; im 5 GHz Band 2064 mW
[6] HD Auflösung; SD: 307 mW, Full-HD@60fps: 843 mW, 4K: 1520 mW
[7] Hauptkamera im Normalmodus; im Nachtmodus: 1342 mW; Frontkamera: 809 mW
[8] im 2.4 GHz Band; im 5 GHz Band 1298 mW
[9] hier mit LTE

Damit man sich ein besseres Bild machen kann, habe ich in der letzten Spalte die theoretischen Durchschnittswerte der drei Geräte berechnet, und daraus eine Tortengrafik erstellt – sodass die Relationen besser erfassbar sind:



Grafische Darstellung der größten Stromverbraucher

Für die Gesamt-Leistungsaufnahme muss natürlich noch die Grundlast hinzugerechnet werden (was das Gerät verbraucht, wenn alles in der Tabelle genannte abgeschaltet ist; also „Flugmodus“). Das wären beim Motorola Milestone sowie beim Galaxy S3 ganze fette 6,4 mW. Wird das Gerät also des Nachts in diesen versetzt, spart das bereits enorm:

Betriebszustand	zusätzliche Leistungsaufnahme			
	Motorola Milestone	Samsung Galaxy S3	Galaxy S5	ø
Flugmodus	6,4 mW	6,4 mW	10,7 mW	7,8 mW
GSM Bereitschaft	18 mW	9,5 mW	15 mW	14,2 mW
GSM Bereitschaft + WLAN Standby	25,8 mW	18,8 mW	26,4 mW	23,7 mW
GSM Bereitschaft + WLAN Standby + Bluetooth Standby	28,6 mW	20,6 mW	40 mW	29,7 mW
UMTS Bereitschaft	24,7 mW	10,9 mW	16,2 mW	17,3 mW
UMTS Bereitschaft + mobile Daten aktiv		13,8 mW		
UMTS Bereitschaft + WLAN Standby	32,5 mW	20,2 mW	27,6 mW	26,8 mW
UMTS Bereitschaft + WLAN Standby + Bluetooth Standby	35,3 mW	22,0 mW	41,2 mW	32,8 mW

Die Werte sind natürlich alle spezifisch für o. g. Motorola Milestone bzw. das Samsung Galaxy S3/S5, und können auf anderen Geräten abweichen. Die Größenordnungen sollten aber zumindest ähnlich sein.

Nur so nochmal gesagt: Selbst wenn WLAN etc. alles aus sind, und nur Telefonate (und SMS) noch durchkommen (den Datenverkehr also mal ganz außen vorgelassen), reduziert sich der Verbrauch im Flugmodus auf ein Drittel

(GSM) oder gar ein Viertel (UMTS). Einmal 6 Stunden angenommen (von 0 Uhr bis 6 Uhr), hält der Akku damit (theoretisch) für bis zu gut 2 Stunden länger. Natürlich wieder nur, wenn man dann anschließend auch nichts damit macht – also wieder so ein „Laborwert“. Trotzdem kann man sich leicht ausrechnen: Sechs Stunden Flugmodus (statt GSM Bereitschaft) schaffen Raum für zusätzliche ca. 8 Minuten GSM Telefonat…