Team der Chemnitzer Linux-Tage:

Chemnitzer Linux-Tage 2013

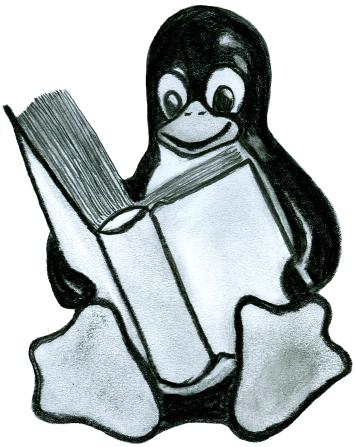
– Tagungsband –

16. und 17. März 2013

Team der Chemnitzer Linux-Tage

Chemnitzer Linux-Tage 2013

16. und 17. März 2013



– Tagungsband –

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Universitätsverlag Chemnitz
2013

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über http://dnb.d-nb.de abrufbar.

Technische Universität Chemnitz/Universitätsbibliothek
Universitätsverlag Chemnitz

Herstellung und Auslieferung:
Verlagshaus Monsenstein und Vannerdat OHG
Am Hawerkamp 31
48155 Münster
http://www.mv-verlag.de

ISBN 978-3-941003-80-4

URL: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-105403
URN: urn:nbn:de:bsz:ch1-qucosa-105403

Satz und Layout: Jens Pönisch und Monique Kosler
Coverfoto: Petra Pönisch
Graphik Innentitel: Petra Pönisch
URL der Chemnitzer Linux-Tage: http://chemnitzer.linux-tage.de

Premiumsponsoren

Weitere Sponsoren

Medienpartner

Inhaltsverzeichnis

1 Vorwort **11**

2 Inhalt der Hauptvorträge **13**

2.1 AsciiDoc – medienneutrales, technisches Publizieren 13
2.2 Erzeugen einfacher OpenStreetMap-Karten 21
2.3 Gnuplot für Anfänger 31
2.4 Himbeertorte – Raspberry Pi zum Anfassen 41
2.5 Linux-Versteher dank strace 49
2.6 Nutzung des OCRA-Algorithmus und LinOTP beim Online-Banking . 61
2.7 Petabyte-Storage leicht gemacht: Ceph 69
2.8 Sensordaten per SNMP verteilen 77
2.9 Vertraulich kommunizieren mit OTR 85
2.10 WeLoupe Next Generation Web-Scanner 95
2.11 Zuverlässigkeit von GNU/Linux-basierten eingebetteten Systemen . . 103

3 Zusammenfassungen der weiteren Vorträge **111**

3.1 3D-Visualisierung mit Blender 111
3.2 Aktuelle Entwicklungen beim Linux-Kernel 111
3.3 Aktuelles vom Supercomputing – Storage Cluster mit ZFS 111
3.4 All about FreeYourAndroid! 111
3.5 Anständiges Werkzeug: emacs 112
3.6 Aptitude – bekannt und doch unbekannt 112
3.7 Best Practice OTRS – Serviceverträge mit OTRS planen, verwalten und überwachen 112
3.8 Building Email Server Cluster with Dovecot 113
3.9 Community Matters 113
3.10 Computer und Hacking überall mit Raspberry Pi & Co 113
3.11 Das fast nützliche Perl-5-Quiz 113
3.12 Datenschutzrecht für Linux-Admins in Betrieben und Behörden 114
3.13 Debian GIS – GIS und OpenStreetMap mit Debian-Bordmitteln 114
3.14 Der mount-Befehl, die Wahlfreiheit für das Dateisystem 114
3.15 Der NAO – ein humanoider Roboter auf Linux-Basis 114
3.16 Die Digitalisierung von Büchern 115
3.17 DNS im RZ-Betrieb mit PowerDNS und MySQL 115
3.18 Ein hochverfügbarer iSCSI Storage mit Linux-Tools 115
3.19 Einführung in BigData (Hadoop/Pig/Hive) 115
3.20 Einführung in das Dateisystem BTRFS 115
3.21 Einführung ins Cross Media Publishing 116
3.22 Einsatz von Postgres im Bereich der Verkehrstelematik 116

3.23 Embedded GNU/Linux mit Gnublin . . . 116
3.24 Fast alles ist Text. Datenformate unter UNIX/Linux. . . . 116
3.25 Forth überall . . . 117
3.26 Freiheit leben für Anfänger – vier Freiheiten der Freien Software als Meilensteine auf dem Linux-Weg . . . 117
3.27 GROUP-E Collaboration Software – best practice aus Aue / Sachsen . . 117
3.28 Grundkonfiguration des Mailclients Mutt . . . 117
3.29 Host-based disk-mirroring with MD . . . 118
3.30 I2P, Tor, Freenet, GNUnet, Retroshare – Kurzvorstellung der Techniken 118
3.31 Ich weiß, was du letzte Woche getan hast! Seitenkanäle in der Gebäudeautomation. . . . 118
3.32 ICINGA – Open Source Monitoring Today and Tomorrow . . . 118
3.33 Integration: Fluch oder Segen – integrierst du noch oder arbeitest du schon? . . . 119
3.34 IPv6 – Migration, Betrieb und Deployment . . . 119
3.35 IT-Management mit Open Source Tools . . . 119
3.36 KDE-Grundeinstellungen: Desktophintergrund, Plasmoids und mehr 120
3.37 LibreOffice im Unternehmenseinsatz . . . 120
3.38 Linux auf ARM-Servern . . . 120
3.39 Linux Essentials – Einsteigerzertifizierung des LPI . . . 120
3.40 Linux im Büro . . . 121
3.41 Linux-Dienstleister stellen sich vor . . . 121
3.42 Managing larger projects with TaskJuggler . . . 121
3.43 Mercurial: Experimente für Fortgeschrittene . . . 121
3.44 Modellierung einer virtuellen DMZ . . . 122
3.45 MySQL HA mit Galera . . . 122
3.46 Nach dem ersten RPM gehts erst richtig los 122
3.47 Nebenläufige Programme mit Python . . . 122
3.48 Netzwerksicherheit durch aktive Sicherheitsscans . . . 123
3.49 Neues aus der BSD-Welt . . . 123
3.50 Nutzung der aktuellen Version des Notensatzprogramms LilyPond . . 123
3.51 Nutzung von PCs und Thin Clients für VDI- und Cloud-Anwendungen 123
3.52 Open choice beim Dokumentenformat: Ist der Anwender überfordert? 124
3.53 Open eCard . . . 124
3.54 Open KnowledgeWorker – Wissen ist frei . . . 124
3.55 Open Source und die Zukunft . . . 125
3.56 Open Source für Klein(st)unternehmen – die vergessene Zielgruppe . 125
3.57 Open-Source-Grafiktreiber für eingebettete Systeme . . . 125
3.58 Open-Source-Projekte einer breiten (Business-)Community verfügbar machen . . . 125
3.59 openSUSE on ARM . . . 126
3.60 Performancetuning für iSCSI . . . 126
3.61 Picosafe: Open Source USB-Stick für Sicherheitsanwendungen . . . 126
3.62 Plan 9, ein Rasberry Pi und Linux . . . 126
3.63 Portable Build-Systeme . . . 127

3.64 POWERLINK + Open Source IP-Core => FPGA . . . 127
3.65 Rechtsfragen für Postmaster und Unternehmen . . . 127
3.66 Replikation mit PostgreSQL . . . 127
3.67 Sieben Mythen über Open Source – Was wir schon immer mal richtig (durch-) denken wollten … . . . 128
3.68 SSH und unzuverlässige Leitungen . . . 128
3.69 Systemadministration in einer agilen Welt . . . 128
3.70 Technische Dokumentation sicher und schick mit zpub . . . 128
3.71 Tippen statt klicken – die Welt der Linux-Kommandozeile . . . 129
3.72 Total Testing . . . 129
3.73 Totalschaden: Ein gehackter Server auf dem OP-Tisch . . . 129
3.74 Transparenz und Transformation: auf dem Weg in die offene Gesellschaft 129
3.75 Tupi – wie die Indios laufen lernen . . . 130
3.76 Tux als Bruchpilot – Vom Höhenflug ins Ministerium zur Bruchlandung im Klassenzimmer . . . 130
3.77 Vom Aussterben bedroht: Die Universalmaschine Computer . . . 130
3.78 Wenn Geeks Langeweile haben . . . 130
3.79 Zarafa als Exchange-Ersatz in der (private | public) Cloud . . . 131
3.80 Zur Funktionsweise von Schadsoftware . . . 131
3.81 Zutrittskontrolle im lokalen Netzwerk . . . 131

4 Zusammenfassungen der weiteren Workshops 132
4.1 3D-Visualisierung mit Blender . . . 132
4.2 Datenintegration mit Talend Open Studio . . . 132
4.3 Die interaktive Programmier-Sprache «J» . . . 132
4.4 Django: Schnell performante Web-Applikationen entwickeln . . . 133
4.5 Firewallkonfiguration mit Shorewall . . . 133
4.6 Inkscape – Sexy Jeans . . . 133
4.7 LaTeX–Einsteigerworkshop . . . 133
4.8 LibreOffice für Umsteiger . . . 134
4.9 OpenFOAM: Numerik mit Freier Software am Beispiel . . . 134
4.10 Python: Programmieren macht Spaß . . . 134
4.11 Setup eines hochverfügbaren Virtualisierungsclusters . . . 134
4.12 ZFS-Einsteigerworkshop . . . 135

5 Personen 137

1 Vorwort

Mit Software lässt sich kein Geld verdienen! Diese Einstellung war weit verbreitet, als die Zeit begann, in der Computer erstmals industriell gehandelt wurden. Geld verdiente man damals durch Produktion und Installation von Hardware. Software war nichts anderes als eine Service-Leistung. Denn schließlich kaufte man eine universelle Maschine, die doch sowieso erst noch durch entsprechende Programme den eigenen Bedürfnissen angepasst werden musste. Wenn man sich vor Augen hält, wie sich die Geschichte dann entwickelte, entbehrt dies nicht einer gewissen Komik. Denn Geld wird doch heutzutage hauptsächlich durch Software verdient. Hardware ist meist eine Service-Leistung.

Dennoch ist es gang und gäbe, dass sich unter unseren Schreibtischen auch heute noch Exemplare dieser universellen Maschinen finden. Sie geben uns die Freiheit auszuprobieren, Erfahrungen zu sammeln und selbst zu experimentieren – kurz, sie bieten uns Wahlfreiheit. Wären Computer nur für die Zwecke einsetzbar, die der Hersteller vorsieht, bliebe es zumindest fraglich, ob in der Vergangenheit eine Reihe von Enthusiasten ihre Wahlfreiheit verwirklicht hätten, die schließlich in freie Compiler, Betriebssysteme, Benutzeroberflächen, … mündete.

Doch stellen wir uns einmal die Frage: Sind Computer heute wirklich noch universell? Rein technisch sicherlich! Aber praktisch? Wir telefonieren mit Computern, auf denen wir nur bestimmte Apps installieren dürfen, kaufen PCs, die nur bestimmte Betriebssysteme booten können, und bekommen erzählt, dass graue Desktop-PCs Anachronismen sind, die doch besser durch schicke Tablet-PCs und ihre hilfreichen Apps ersetzt werden sollten. Die Wahlfreiheit wird damit zum Privileg von uns Experten, die wissen, wie man «universelle Computer» bedient.

Sie fragen, ob ich mit dieser Aussage den Untergang der freien Softwarebewegung an den Horizont malen will? Nein, keinesfalls! Ich bin mir sicher, dass wir – die Community – dafür sorgen werden, dass auch zukünftige Generationen die Möglichkeit haben, ihre Wahlfreiheit zu verwirklichen, indem wir zeigen, welchen Nutzen die Gesellschaft aus Freier Software ziehen kann. Trotz aller Bescheidenheit denke ich, dass unser diesjähriger Konferenzband einen kleinen Beitrag zu diesem Ziel leistet. Viel Spaß bei der Lektüre.

Mario Haustein im Februar 2013

AsciiDoc - medienneutrales, technisches Publizieren

Wilhelm Meier
w.meier@unix.net
http://mozart.informatik.fh-kl.de

AsciiDoc ist eine sehr leichtgewichtige Auszeichnungssprache. Sie eignet sich hervorragend zur Dokumentation von technischen Produkten, insbesondere von Softwareartefakten. Mit dem `asciidoc`-Prozessor kann aus *einer* Quelle (single-source publishing) ein sehr große Vielzahl von Medien erzeugt werden (z.B. eBook-Formate, Print-Formate, Online-Formate). Im Vortrag soll zum einen *AsciiDoc* als Sprache besprochen werden, zum anderen soll aber auch ein besonderer Editor (`aedit`) inkl. Präprozessor(`app`) vorgestellt werden. Dieser ermöglicht durch eine Vielzahl von kontextsensitiven Hilfsfunktionen ein ganz besonders schnelles Erzeugen von Softwaredokumentation.

1 Technisches Dokumentieren

Technische Dokumentationen zu erstellen ist nicht selten eine recht große Herausforderung. Gilt es doch einerseits einen äußerst präzisen Text zu schreiben, der aber auch andererseits dadurch gekennzeichnet ist, dass viele nicht-immanente Inhalte wie etwa Bilder, spezielle Grafiken, Quelltext bzw. Fremddateien und Systemausgaben sowie etwa Fremddokumente in den Text zu integrieren sind. Alles dies zusammen bildet eine semantische Einheit und soll gemeinsam verwaltet und auch versioniert werden können. Weiterhin soll das Endprodukt in den unterschiedlichsten Formaten publiziert werden können: von der reinen Druckversion etwa als `pdf` oder *PostScript* über Online-Formate (`html`, `html5`) bis zu *eBook*-Versionen (`epub`, `mobi`), wobei selbstverständlich jedwedes Format aus ein und derselben Quelle automatisch erzeugt werden soll.

Selbstverständlich soll der Autor eines Textes dabei in optimaler Form unterstützt werden. Geht es um das Schreiben einfacher Texte ohne erhöhte Layoutanforderungen, so sind sicher die allgemein bekannten, universellen *Office*-Produkte wie etwa *LibreOffice* [7] gut geeignet und ermöglichen eine gute Produktivität.

Schwierigere Aufgaben, etwa das Verfassen langer (> 100 Seiten) und mit typografischen Herausforderungen durchsetzten Dokumenten, benötigen allerdings schon besondere Werkzeuge. In diesem Fall bietet das *WYSIWYG*-Konzept der o.g. *Office*-Produkte nicht die notwendige Vielfalt bzw. Produktivität. In diesem Fall wird von den meisten Autoren auf LaTeX[8] zurückgegriffen. Obgleich etwas in die Jahre gekommen, bietet es doch gerade für Autoren aus dem mathematischen / informatischen Umfeld exzellente Möglichkeiten, sehr ansprechende Dokumente in den unterschiedlichsten Formaten zu erzeugen.

Mit der Abkehr vom *WYSIWYG*-Konzept kommen weitere Möglichkeiten ins Blickfeld, die im folgenden beleuchtet werden sollen.

2 Dokumentationswerkzeuge

Die Palette der verfügbaren Dokumentationswerkzeuge - F/OSS wie auch kommerziell / closed-source - ist schier unübersichtlich. Daher ist es hilfreich, zunächst die Anforderungen festzulegen, die bei der Auswahl eines Werkzeuges zugrunde gelegt werden sollen. Diese Anforderungen erwachsen selbstverständlich aus der Domäne, für die Dokumentation erzeugt werden soll. Zusätzlich ergeben sich auch Anforderungen aus dem Umfeld (etwa projekt- oder unternehmensspezifische Vorgaben) und nicht zuletzt spielt eine gewisse Portion persönlicher Vorliebe mit hinein.

Ohne einen Anspruch auf Vollständigkeit und mit einer nicht zu vernachlässigenden Portion persönlicher Vorliebe des Autors sollen die folgenden allgemeinen Anforderungen für die Auswahl des Dokumentationswerkzeuges gelten:

- Technische Dokumentation ist stark *strukturierte* Dokumentation. Gleiche Strukturen sollten sich auch später im Satz gleich darstellen, allerdings ist bei der Erstellung der Texte der tatsächliche Satz eher hinderlich.
- Die technische Dokumentation ist direkt an ein technisches Produkt geknüpft, etwa ein Software-Artefakt. Im Falle von Software sollten daher die Quellen für die Dokumentation zusammen mit den Quellen der Software gleichförmig verwaltet werden können.
- Oft wird die Dokumentation von unterschiedlichen Autoren erzeugt / gepflegt. Gute Dokumentation entsteht aber nur, wenn die Autoren in ihrer Produktivität nicht eingeschränkt werden. Daher sollte das Werkzeug zur Erstellung der Dokumentationsquellen nicht mit dem Werkzeug zum Erstellen der endgültigen Dokumentationsformate gekoppelt sein.

Fokussiert man nun noch die Anwendungsdomäne auf die Erstellung von Dokumentation für Software (etwa Handbücher für Applikationen oder Software-Bibliotheken, Bücher, Skripte oder Tutorien zu Programmiersprachen oder Softwaresystemen), so konkretisieren sich die Anforderungen nach Ansicht des Autors zusätzlich in den folgenden Punkten:

- Die Auszeichnung der Strukturelemente sollte im Text vorgenommen werden können. Dies muss möglichst *leichtgewichtig* sein, um die Schreibgeschwindigkeit nicht zu bremsen (eine Mausbedienung ist wiederum eher hinderlich).
- Nicht im Text immanente Elemente (Bilder, Grafiken, Quelltext, Systemausgaben, Fremddateien, etc.) müssen einfach zu inkludieren sein.

- Ebenso sollte es möglich sein, nicht-Text-Elemente trotzdem *textuell* zu beschreiben (Beispiel: *UML*-Diagramm). Der Nachteil der notwendigen zusätzlichen Abstraktion vom Text etwa zum fertigen Diagramm wird nach Ansicht des Autors durch die enge Koppelung von Text und Diagramm sowie den Verzicht auf einen Werkzeugwechsel mehr als aufgehoben.

Fasst man nun die obige Menge der Anforderungen zusammen und sucht nach Werkzeugen, die dem entsprechen können, so scheiden *WYSIWYG*-Werkzeuge ebenso aus wie etwa LaTeX, was nach Ansicht des Autors eine zu schwergewichtige Auszeichnungssprache besitzt. Damit rücken die *Wiki*-sprachbasierten Werkzeuge ins Blickfeld, denn die hier benutzte Auszeichnungssprache ist naturgemäß sehr leichtgewichtig. In dieser Gruppe findet man die generellen *Wiki*-Sprachen wie etwa *MarkDown* [4], *Restructured Text* [6] oder auch *AsciiDoc* [1] und *Textile* [5]. Weiterhin existieren noch die für die Software-Dokumentation speziell geschaffenen Werkzeuge wie etwa *QuickBook* [2] aus dem *Boost*-Framework oder auch *QDoc* [3] aus dem *Qt*-Framework [10].

Die dazugehörige Werkzeugkette sollte es einfach ermöglichen, alle heute typischen Zielformate, etwa `html`, `pdf`, `epub` oder `mobi` zu erzeugen. Als bewährtes Vorgehen hat es sich erwiesen, durch das originäre Werkzeug das Dokument zunächst in ein universelles Zwischenformat zu transformieren. Hier kommt aus Sicht des Autors nur noch *DocBook* [9] in Frage.

Damit scheidet *QDoc* [3] aus der Liste der o.g. Werkzeuge aus: zum einen produziert es ausschließlich `html` und zum anderen ist der Einsatz zu sehr auf die reine Dokumentation von Software in Handbuchform beschränkt, da es den Text in Form von Quelltext-Kommentaren erwartet.

Diese Schwachstelle ist etwa in dem sehr mächtigen Werkzeug *QuickBook* [2] nicht vorhanden. Darüberhinaus hat *QuickBook* gerade für die Dokumentation von Quelltext ganz zentrale Merkmale, etwa das einfache Inkludieren / Verlinken von Quellcode-Elementen (Klassen, freie Funktionen, etc.) anhand ihres vollständig qualifizierten Namens und es bietet auch die Möglichkeit, Quellcode-*Snipptes*, also Fragmente bestehender externer Quelltextdateien zu inkludieren. Allerdings entfällt bei *QuickBook* (genauso wie bei *QDoc*) die Möglichkeit, nicht-Text-Elemente (etwa Diagramme) in der Dokumentationsquelle direkt zu beschreiben (s.u. *Asciidoc*).

Die allgemeinen *Wiki*-basierten Werkzeuge haben zwar fast alle die Möglichkeit, `docbook` als Zwischenformat zu produzieren, allerdings bietet nur *AsciiDoc* hier die Möglichkeit, nicht-Text-Elemente *inline* durch entsprechende *plugins* zu erzeugen. Damit ist *AsciiDoc* in der Abdeckung und Gewichtung der o.g. Anforderung nach Ansicht des Autors das geeignetste Werkzeug, um schnell und einfach, umfangreiche technische Dokumentation mit dem Schwerpunkt softwaretechnische Systeme zu erzeugen.

3 AsciiDoc

AsciiDoc als Werkzeug und Sprache soll in diesem Artikel nicht näher beschrieben werden. Dazu existiert ausreichend Dokumentation [1] sowie genügend Beispiele für ansprechende Endprodukte, um die Fähigkeiten zu demonstrieren. An dieser Stelle soll lediglich das Alleinstellungsmerkmal *plugin*-Konzept hervorgehoben werden.

Als Beispiel dienen ein *plugin* zum Erzeugen einfacher bis komplexer Blockdiagramme (*BlockDiag* [11]) sowie eines zum Erzeugen von UML-Diagrammen (*PlantUML* [12]). Genauso wie etwa die Syntaxhervorhebung für inkludierte Quelltextfragmente durch ein externes Werkzeug (hier *GNU Source Highlighter* [13]) mit Hilfe eines *plugins* erledigt wird, können auch Diagramme erzeugt werden. Dies erfolgt durch einen sog. *Filter*, das durch das *plugin* installiert wird. Bei der Verarbeitung des Dokumentes durch `asciidoc` wird der entsprechend gekennzeichnete Paragraph oder auch ein ganzer Block (s.u.) durch das externe Filterprogramm in das ausgewählte Zwischenformat wie etwa `docbook` bzw. `svg` (zur weiteren Inklusion) umgewandelt.

Der Text im folgenden Kasten erzeugt ein einfaches Blockdiagramm:

```
Asciidoc-Text davor.

.Ein Titel
["blockdiag"]
----
blockdiag {
   A -> B -> C -> D;
   A -> E -> F -> G;
}
----
Weiterer Asciidoc-Text dahinter.
```

Mit dem Text in diesem Kasten wird eine einfaches UML-Diagramm erzeugt:

```
["plantuml"]
----
class Date {
      +Date(int)
      --
      +Date date()
      .. optional ..
      #void foo()
      --
      -int geheim
}
interface Timeable
abstract class Nonsense
```

```
Timeable <|-- Date
Nonsense <|-- Date
----
```

Natürlich können auch externe Quellcode-Dateien eingebunden werden. Auch zum Extrahieren von Quellcode-Fragmenten (s.a. *QuickBook*) existieren Filter (allerdings s.a. unten *Werkzeuge*).

4 Werkzeuge

Als *Wiki*-Auszeichnungssprache lassen sich *AsciiDoc*-Dokumente mit jedem beliebigen Texteditor erstellen. Für `vim`, `emacs` und `katepart` existieren zudem Definitionen zur Syntaxhervorhebung. Diese lassen sich zudem auch in vielen Entwicklungsumgebungen (etwa *QtCreator*) verwenden, so dass das Edieren von Software-Quelltext und auch Dokumentations-Quelltext immer mit demselben Werkzeug erfolgen kann. Zusammen mit dem o.g. *plugin*-Konzept sichert dies allein schon entscheidende Produktivitätsvorteile.

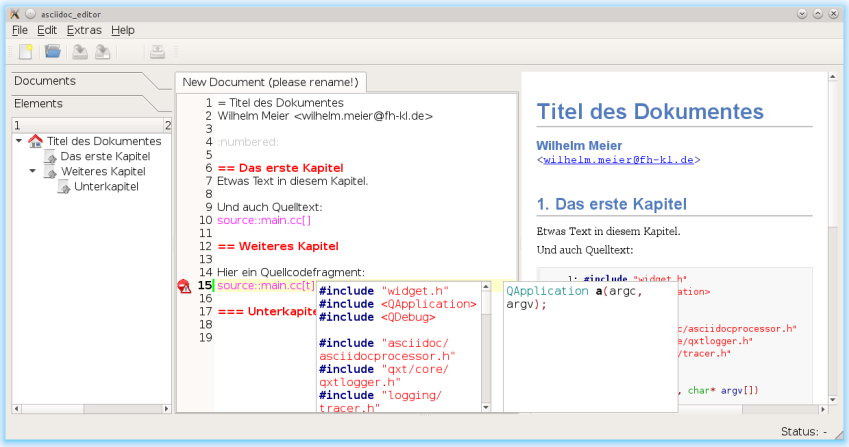


Abb. 1: Der Editor *aedit* mit Anzeige der Dokumentstruktur (links), des Dokumentquelltextes und Vervollständigung bei der Inklusion eines Quelltextfragmentes (Snippet) (mitte) sowie rechts die Anzeige des Zielformates (`html`), die synchron zu den Änderungen des Dokumentquelltextes erfolgt.

Allerdings bleiben hier noch zwei Punkte offen:

- Definition vom Macros für `asciidoc` in der Sprache *AsciiDoc* selbst

- Ein kontextsensitiver Editor, der neben Syntaxhervorhebung über ein Dokument-Modell verfügt und somit in der Lage ist, u.a. eine Autovervollständigung für Sprachelemente (etwa Querverweise, URLs oder Text-Variablen, etc.) und vor allem aber auch für die Inklusion von Software-Quelltexten / Fragmenten anzubieten.

Als ein *nice-to-have* Merkmal sollte der Editor aber auch in der Lage sein, das Zielformat schritthaltend mit dem Editiervorgang darzustellen. Dazu wurde der Editor `aedit` (s.a. Abb. 1) als *Qt*-Applikation entwickelt. Um eine Text-Autovervollständigung anzubieten, muss der Editor die Dokumentstruktur parsen und ein Dokumentmodel aufbauen. Da die Autovervollständigung auch für die Software-Quelltexte bzw. Fragmente daraus möglich sein soll, müssen ebenso die zu inkludierenden Software-Quelltexte geparsed werden.

Weil nun schon ein Dokument-Parser entwickelt werden musste, lag es nahe, damit auch die Möglichkeit zu schaffen, *neue* Sprachelemente für *AsciiDoc* zu schaffen (s.o. *Macros*). Um die Unabhängigkeit bei der Wahl des Erstellungswerkzeuges beizubehalten, wurde der *AsciiDoc*-Parser zusammen mit *Macro-Engine* und *Code-Snippet-Extractor* in einen eigenen *AsciiDoc*-Präprozessor ausgelagert. Damit können diese Merkmale auch ohne den speziellen Editor `aedit` genutzt werden.

Beispielhaft sei die oft genutzte Inklusion von Quelltext-Fragmenten beschrieben. Für die Inklusion von Quelltextfragmenten wird diesselbe Annotation wie bei *QuickBook* benutzt: die zu inkludieren Fragmente werden durch spezielle Kommentare der Form `//[<name> ... //]` wie in dem folgenden Kasten definiert:

```
//[snippet1
  int main() {}
//]
```

Die Inklusion im Dokument erfolgt dann durch eine spezielles Macro `source::<filename>[<snippetPath>]`. Im Editor wird dafür auch eine Autovervollständigung sowohl des Dateinamens `<filename>` angeboten wie auch der darin enthaltenen Quelltextfragmente in `<snippetPath>`. Beim Auswahlprozess der Autovervollständigung wird neben dem Quellcode auch noch das jeweilige Fragment angezeigt. Diese Anzeige ist auch später als *ToolTip* in der Dokumentanzeige verfügbar.

Die Autovervollständigung ist darüber hinaus auch für Dokumentvariablen (*AsciiDoc*: Attribute), Querverweise, Fremddateien, URLs verfügbar. Bei URLs wird einerseits die Verfügbarkeit bei jeder Dokumentverarbeitung durch den Präprozessor geprüft und es werden Inhalts-Zusicherungen geprüft, um sicherzustellen, das tatsächlich auf den gewünschten Inhalt verlinkt wird.

Als Alternative zum dedizierten Auszeichnen der Software-Quelltexte durch die *Snippet*-Markierungen wäre etwa über das Projekt *srcML* [15] die Auswahl von Fragmenten über *XPath*-Ausdrücke denkbar. Dies hätte gegenüber der von *QuickBook* eingesetzten Möglichkeit, Quelltext-Dokumente über vollständig qualifizierte Namen in C++ Form wie `<namespace>::<class>::<classElement>` zu verlinken (nicht zu inkludieren) den Vorteil, dass dies auch für andere Sprachen als C++ möglich ist.

Weiterhin wurde eine schritthaltende Anzeige des Zielformats `html` im Editor realisiert. Damit eine *bidirektionale* Navigation zwischen dem Dokument-Quelltextfenter und der `html`-Darstellung ermöglicht wird, wurde ein Web-Browser (*QWebKit*) in den Editor integriert. Die bidirektionale Navigation wird durch spezielle Tags im Zielformat möglich gemacht, die durch den `app`-Präprozessor eingebracht werden. Geplant ist dies auch für das Zielformat pdf.

Literatur

[1] *AsciiDoc*, Stuart Reckham, Jan. 2013, http://www.methods.co.nz/asciidoc/index.html

[2] *QuickBook*, Boost-C++-Libraries, Jan. 2013, http://www.boost.org/doc/libs/1_52_0/doc/html/quickbook.html

[3] *QDoc*, Digia, Jan. 2013, http://doc.qt.digia.com/qdoc/index.html

[4] *MarkDown*, WikiPedia, Jan. 2013, http://de.wikipedia.org/wiki/Markdown

[5] *Textile*, WikiPedia, Jan. 2013, http://de.wikipedia.org/wiki/Textile

[6] *Restructured Text*, WikiPedia, Jan. 2013, http://de.wikipedia.org/wiki/ReStructuredText

[7] *LibreOffice*, Jan. 2013, http://de.libreoffice.org/

[8] *LaTeX*, WikiPedia, Jan. 2013, http://de.wikipedia.org/wiki/LaTeX

[9] *DocBook*, Jan. 2013, http://www.docbook.org/

[10] *A cross-platform application and UI framework*. Digia, Jan. 2013 http://qt.digia.com/

[11] *BlockDiag - a block diagram generator*, Jan. 2013, http://blockdiag.com/en/blockdiag/index.html

[12] *PlantUML - UML Diagram generator*, Jan. 2013, http://plantuml.sourceforge.net/

[13] *GNU Source Highlighter*, Lorenzo Bettini, Jan. 2012 http://www.gnu.org/software/src-highlite/

[14] *Standard ECMA-262: ECMAScript Language Specification*. ECMA-International, Jun. 2011
http://www.ecma-international.org/publications/standards/Ecma-262.htm

[15] *srcML: A document-oriented XML representation of source code*, SDML, Jan. 2013, http://www.sdml.info/projects/srcml/

[16] *KMUX - Linux und F/OSS für kleine und mittelständische Unternehmen*, Apr. 2011, http://www.kmux.de

Erzeugen einfacher OpenStreetMap-Karten

Jens Pönisch*
TU Chemnitz

OpenStreetMap

Das 2004 von Steve Coast gegründete Projekt hat sich zum Ziel gesetzt, geographische Daten wie Landschaftsnutzung, Besiedelungs- und Infrastruktur umfassend und unter einer freien Lizenz (ODBL) in Form einer Datenbank zur Verfügung zu stellen.

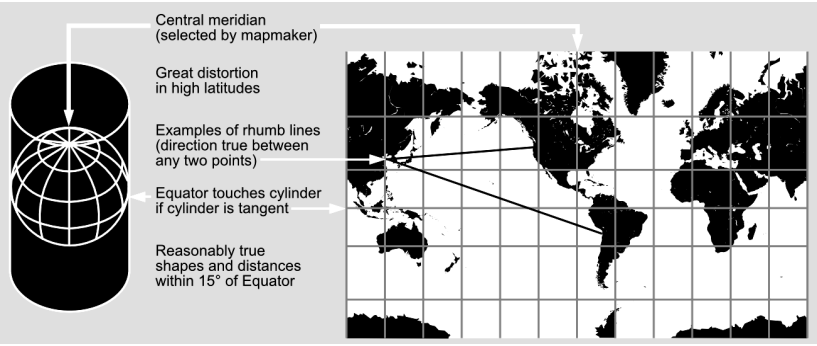
Die sicher wichtigste Nutzungsform ist die Erzeugung von geographischen Karten. So bietet das Projekt selbst einige Online-Karten in verschiedenen Stilen an. Aus diesen bereits aufbereiteten Daten sollen mit möglichst einfachen Mitteln eigene Karten für verschiedene Anwendungsszenarien erzeugt werden.

Nicht eingegangen wird hier auf *routingfähige* Karten, wie sie für Navigationssysteme benötigt werden.

Sphärische Mercatorprojektion und Slippy Map

Da die Erdoberfläche näherungsweise ein Ellipsoid ist, Karten aber flach sind, ist eine *Projektion* von dieser Oberfläche in die Kartenebene erforderlich. Sollen die erzeugten Karten zur Navigation dienen, ist es sinnvoll, wenn Himmelsrichtungen und Winkel an Kreuzungen unverändert bleiben, was mithilfe der *Mercatorprojektion* möglich ist.

Die Erde wird dazu auf einen übergestülpten Zylinder, der in Richtung der Erdachse ausgerichtet ist, projiziert. Anschließend wird der Zylinder an der Datumslinie (180° Ost = 180° West) aufgeschnitten und geeignet zu einem Quadrat verzerrt.



*poenisch@isym.tu-chemnitz.de

Mercatorprojektion (Quelle: Wikimedia Commons)

Zur Beschleunigung der Rechnung wird diese zur *Sphärischen Mercatorprojektion* vereinfacht, die von einer exakten Kugelgestalt der Erde ausgeht und dabei minimale Abweichungen in Kauf nimmt.

Die Berechnung erfolgt nach folgender Formel:

$$
\begin{aligned}
x &= s \cdot \lambda \\
y &= s \cdot \ln\left(\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right) \\
&= s \cdot \ln\left(\tan\varphi + \frac{1}{\cos\varphi}\right)
\end{aligned}
$$

λ – Längengrad in Radiant

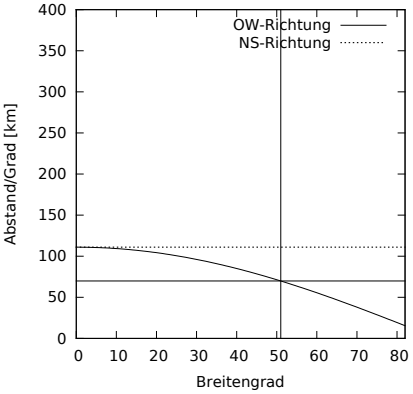
φ – Breitengrad in Radiant

s – Skalierungsfaktor

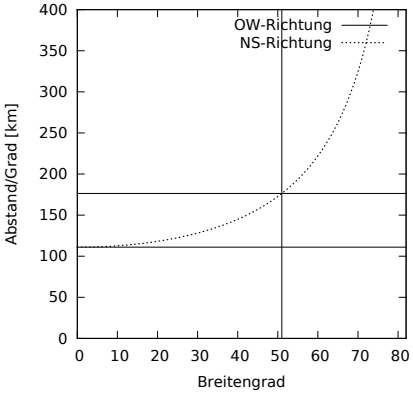
Die Längen- und Breitengrade (im WGS84-Referenzsystem) müssen zunächst in das Bogenmaß umgerechnet werden. Östliche Längen und nördliche Breiten haben dabei ein positives, westliche Längen und südliche Breiten ein negatives Vorzeichen:

$$\text{Radiant} = \frac{\pi}{180^\circ} \cdot \text{Grad}$$

Das Ergebnis dieser Projektion ist ein Quadrat, das die Erdoberfläche bis zum 82-ten Breitengrad nach Norden und Süden abbildet. Abstände werden dabei umso mehr vergrößert, je weiter man sich vom Äquator entfernt, da die Längengrade in der Projektion parallel verlaufen und dafür der Abstand der Breitengrade vergrößert wird. Auf dem 51-ten Breitengrad (Höhe Chemnitz) ist dieser Verzerrungsfaktor etwa 1,6.



Tatsächlicher Abstand je Grad



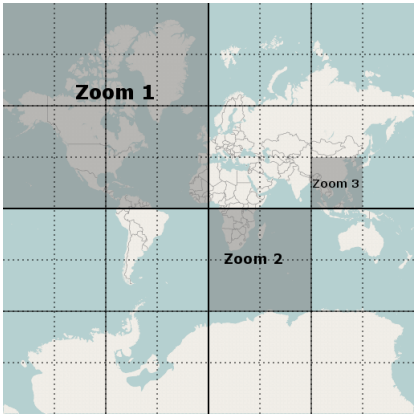
Abstand je Grad in Mercatorprojektion

Dieser Umstand führt bei großflächigen Karten zu einer verfälschten Darstellung von Flächenverhältnissen, bei regionalen Karten muss er lediglich bei der Angabe eines Kilometerrasters oder einer Entfernungsskala berücksichtigt werden.

Für die Kartendarstellung wird dieses Quadrat je nach Zoomstufe umskaliert und in quadratische Kacheln der Größe von 256×256 Pixel zerlegt. Diese werden, beginnend in der linken oberen Ecke, in x- und y-Richtung durchnumeriert. Eine bestimmte Kachel wird also durch das Tripel *(Zoomstufe, x-Koordinate, y-Koordinate)* charakterisiert.

Die Umrechnung von geographischen in Kachelkoordinaten erfolgt durch Anpassung obiger Formel:

$$x = \left\lfloor 2^z \cdot \frac{\lambda + \pi}{2 \cdot \pi} \right\rfloor \qquad y = \left\lfloor 2^z \cdot \frac{\pi - \ln(\tan\varphi + \frac{1}{\cos\varphi})}{2 \cdot \pi} \right\rfloor$$



Konstruktion der Slippy-Map

Dieses Konzept wurde von *Google Maps* übernommen und wird im *OpenStreetMap*-Umfeld als *Slippy Map* bezeichnet. Mit jeder höheren Zoomstufe verdoppelt sich dabei die Zahl der Kacheln in waagerechter und senkrechter Richtung, es wird also insgesamt die vierfache Kachelzahl benötigt. Üblicherweise werden die Zoomstufen bis 16 oder 18 bereitgestellt.

Für die Darstellung regionaler Karten sind besonders folgende Zoomstufen interessant. Die Größenangaben beziehen sich wieder auf den 51-ten Breitengrad.

Zoom	Kachellänge/-breite [km]	Maßstab bei 4 cm Druckbreite
11	12,5	1:300 000
12	6	1:150 000
13	3	1:75 000
14	1,5	1:38 000
15	0,8	1:19 000

Karten für den Ausdruck

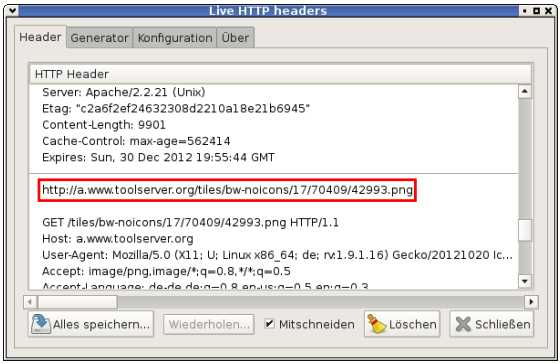
Um eine Karte für den Ausdruck zusammenzustellen, müssen die Kacheln für die gewünschte Zoomstufe besorgt und diese zu einem Bild zusammengestellt werden. Einige Online-Karten bieten diese Funktionalität über ihr Web-Interface selbst an. Fehlt diese Funktion, können diese Schritte manuell durchgeführt werden.

Die von Online-Karten verwendeten Kacheln sind praktisch immer über eine *URL* erreichbar, die jedoch nicht unbedingt mit der *URL* der Kartenanwendung übereinstimmt. Um die benötigten Daten zu ermitteln, kann man entweder den *JavaScript*-Quelltext der Kartenanwendung oder die von ihr ausgelösten *HTML*-Requests analysieren.

Die zweite Variante lässt sich bequem mithilfe des Firefox-Add-ons *Live HTTP headers* [4] realisieren, wobei die Kachel-Requests üblicherweise den Aufbau

http://pfad/*zoom*/*x*/*y*.png

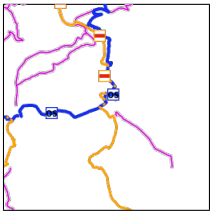
haben und so leicht zu erkennen sind.



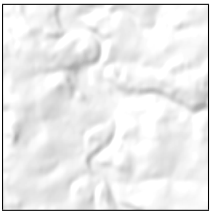
Kachel-Anfrage in Live HTTP headers

Es gibt Basiskacheln, die die eigentliche Kartendarstellung enthalten, sowie häufig *Overlays* mit Zusatzinformationen wie Wanderwegen, Höhenschattierungen und -linien. Letzere sind weitestgehend transparent und können die Basiskacheln überlagern.

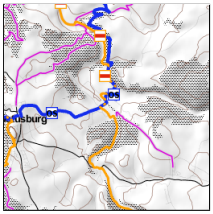
Mapnik-Basiskarte



Overlay Wanderwege (Lonvia)



Overlay Höhen (Hike-and-Bikemap)



Basiskarte mit Overlays kombiniert

Nach Festlegung der Zoomstufe können die x- und y-Werte der benötigten Kacheln durch obige Umrechnung aus den geographischen Koordinaten der darzustellenden Region berechnet werden.

Der Download kann nun mit *curl* oder *wget* erfolgen, das Zusammensetzen der Einzelkacheln zu einer Gesamtgraphik wird durch die Graphikbibliotheken vieler Scriptsprachen unterstützt. Eine mögliche Vorlage ist das Python-Script `bigmap.py` des Autors [5].

Ein interaktives graphisches Tool zum Zusammensetzen derartiger Karten ist das vom Autor entwickelte Programm *QBigMap* [6]. Zusätzlich können hier auch GPX-Tracks dargestellt und -Routen entworfen werden.

Ein Nachteil der auf diese Weise generierten Karten soll nicht verschwiegen werden: Da die Kacheln für die Online-Darstellung optimiert sind, erscheinen Beschriftungen im Ausdruck häufig sehr klein oder nur schwer lesbar.

Eigene Online-Karten

Für Weg- und Ortsbeschreibungen auf Webseiten besteht häufig der Wunsch, eigene Online-Karten bereitzustellen. Dazu ist etwas *JavaScript*-Programmierung erforderlich. Die Kartenfunktionalität wird hier von den Bibliotheken *OpenLayers* [7] und *Leaflet* [8] bereitgestellt, wobei die letztere sehr einfach zu nutzen, die erste allerdings leistungsfähiger ist.

Für die Bereitstellung der Karten genügt ein einfacher Webspace, es wird keinerlei zusätzliche Serverfunktionalität benötigt.

Eine eigene Karte unter Benutzung der *Leaflet*-Bibliothek besteht aus einem HTML- und einem Javascript-Teil, der hier auf zwei Dateien verteilt wird.

Die HTML-Datei bindet die benötigten JavaScript-Bibliotheken und Styles ein und stellt die Kartenfläche in Form eines `div`-Elements mit vorgegebener Größe bereit.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Leaflet-Karte 1</title>
<!-- Leaflet-Symbole -->
<link rel="stylesheet"
      href="http://cdn.leafletjs.com/leaflet-0.4/leaflet.css"/>
<!-- Leaflet-Bibliothek -->
<script src="http://cdn.leafletjs.com/leaflet-0.4/leaflet.js"></script>
<!-- Eigene Kartenfunktionalität -->
<script src="myleaflet1.js"></script>
</head>
<!-- Initialisierung mit Breite, Länge, Zoom -->
<body onload="javascript:map(50.8,12.9,12)">
<!-- Kartenfläche, Größe muss gesetzt sein -->
<div id="map" style="width:100%;height:100%;"></div>
</body>
```

Die JavaScript-Datei erzeugt das Kartenobjekt und verknüpft es mit einer oder mehreren Kartenquellen:

```
function map(lat, lon, zoom) {
  // Kartenobjekt mit Position und Zoomfaktor erzeugen
  var myMap = L.map('map').setView([lat, lon], zoom);
  // Attributionstring (Copyright) setzen
  var attribution = 'Map data &copy; <a href="http://openstreetmap.org">'
    +'OpenStreetMap</a> contributors, '
    +'<a href="http://opendatacommons.org/licenses/odbl/">ODbL</a>';
  // Kartenlayer hinzufügen, Platzhalter {z}, {x}, {y}
  L.tileLayer('http://tile.openstreetmap.org/{z}/{x}/{y}.png',
```

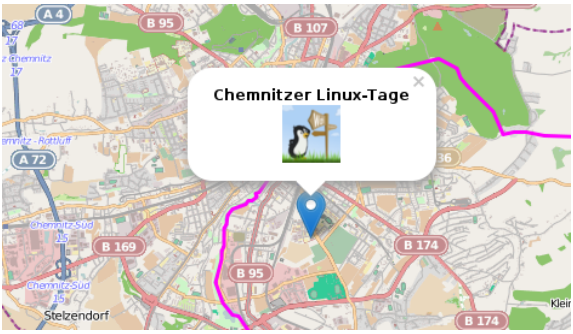
```
  { attribution: attribution,
    maxZoom: 18
  }).addTo(myMap);
}
```

Um einen Punkt (*POI* = Point of Interest) zu markieren, wird folgender Code ergänzt:

```
// Setze Marker
var marker = L.marker([50.8135, 12.9293]).addTo(myMap);
// Beim Anklicken Popup anzeigen
marker.bindPopup('<b>Chemnitzer Linux-Tage<br/><img src="clt-logo.png"/>');
```

Um einen Track anzuzeigen, muss dieser mittels `XMLHttpRequest`-Objekt geladen und anschließend geparst werden.

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.open('GET', 'track.gpx', true);       // URL des Tracks
xmlhttp.overrideMimeType('application/xml'); // Rückgabetyp setzen
xmlhttp.onreadystatechange = function () {
  if (xmlhttp.readyState != 4) return;        // Request unvollständig
  if (xmlhttp.status != 0 && xmlhttp.status != 200 && xmlhttp.status != 304)
    return; // Fehler
  // Request erfolgreich, GPX-Punkte ermitteln.
  // Für Waypoints "wpt", für Routenpunkte "rtept" statt "trkpt" verwenden.
  var nodes = xmlhttp.responseXML.getElementsByTagName('trkpt');
  drawNodes(map, nodes);
}
xmlhttp.send();

function drawNodes(map, nodes) {
  poly = [];    // Koordinatenfeld
  for (var i = 0; i < nodes.length; i++) {
    var lat = parseFloat(nodes[i].getAttribute('lat'));
    var lon = parseFloat(nodes[i].getAttribute('lon'));
    poly.push(new L.LatLng(lat, lon)); // Koordinate hinzufügen
  }
  var track = new L.Polyline(poly, {}).addTo(map);  // Linienzug hinzufügen
  track.bindPopup('Trackbeschreibung');             // Popup für Linienzug
}
```



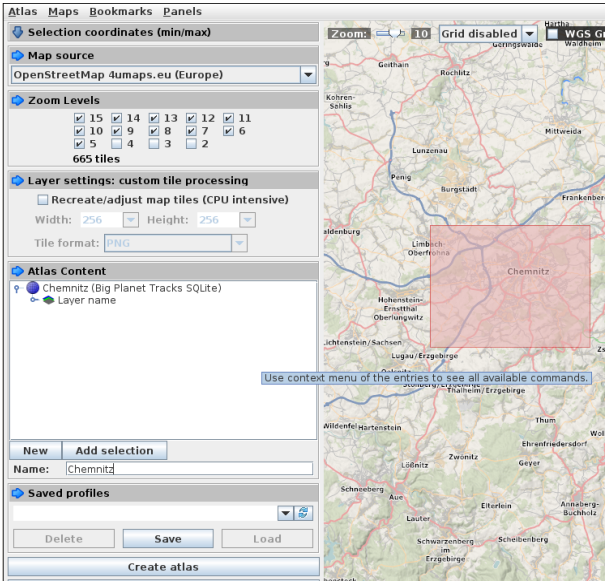
Online-Karte mit Leaflet, Popup und Track

Android-Karten

Für Android-Geräte existieren eine Vielzahl freier und kommerzieller Apps auf der Basis von OpenStreetMap-Daten. Hier wurde die quelloffene, kachelbasierte Anwendung *Big Planet Tracks* [9] ausgewählt, die Offline-Nutzung und das Aufzeichnen von Tracks ermöglicht.

Die Atlanten werden in Form von SQLite3-Datenbanken bereitgestellt, die Auswahl zwischen mehreren Atlanten ist möglich. Beim Online-Betrieb werden diese Datenbanken als lokaler Cache benutzt, im Offline-Betrieb dienen sie als Datenquelle.

Offline-Karten können auf dem PC mit der Java-Anwendung *Mobile Atlas Creator* (MOBAC) [10] für gewünschte Regionen und Zoomstufen erzeugt werden.



MOBAC-Kartenerzeugung

Nach dem Start dieser Java-Anwendung mit dem Befehl

```
java -jar Mobile_Atlas_Creator.jar
```

werden ein Atlasname und das Ausgabeformat *Big Planet Tracks SQLite* gewählt. Sind Kartenquelle, die gewünschten Zoomstufen und die Region gewählt, kann der Atlas erzeugt werden, was je nach Zahl der benötigten Kacheln etwas Geduld erfordert. Die Karte wird im Verzeichnis `~/atlases` abgelegt und muss nun in das Massenspeicherverzeichnis `RMaps/maps` übertragen werden, was über die USB-Verbindung erfolgen kann. Wenn das Gerät für die Datenübertragung nur den MTP-Modus unterstüzt, muss unter Debian das Paket `mtpfs` installiert und anschließend das Gerät manuell montiert werden:

```
sudo mtpfs -o allow_other /mnt
```

Nach dem Kopieren der Datei wird das Gerät wieder ausgehängt:

```
sudo umount /mnt
```

MOBAC stellt jedoch nur bestimmte Kartenstile zur Auswahl, eine Konfigurationsmöglichkeit für weitere Datenquellen konnte nicht gefunden werden. Deshalb soll das Datenformat der SQLite-Datenbank kurz erläutert werden, um erfahrenen Anwendern die Bereitstellung eigener Atlanten aus anderen Kachelquellen und mit Overlays zu ermöglichen.

Die Datenbank besteht aus drei Tabellen. Die Tabelle `android_metadata` enthält lediglich die Spalte `locale` mit einem einzigen Eintrag der Sprache (`de_DE`), die Tabelle `info` die beiden Spalten `minzoom` und `maxzoom`, die den kleinsten und größten Zoomfaktor der gespeicherten Daten enthalten. Die eigentlichen Daten befinden sich in der Tabelle `tiles`. Hier enthalten die Spalten `x`, `y` und `z` die Kachelnummer der jeweiligen Zoomstufe und die Spalte `image` die Kacheldaten als JPEG- oder PNG-Bild. Die Spalte `s` (Strategy) enthält bei Offline-Karten immer den Wert `0`.

Karten für Garmin-GPS-Empfänger

Viele Garmin-GPS-Empfänger ermöglichen das Laden eigener Karten, die nach Vorstellung des Herstellers von diesem gekauft werden sollen. Inzwischen wurde jedoch das Datenformat weitgehend entschlüsselt, so dass eigene Karten erstellt werden können. Im *OpenStreetMap*-Wiki [1] stehen für eine Vielzahl von Regionen solche Karten zum Download zur Verfügung. Ist eine Karte für eine gewünschte Region jedoch nicht verfügbar oder aufgrund der Größe für ein etwas älteres Gerät nicht geeignet, so kann diese selbst berechnet werden. Voraussetzung für die Nutzung der Werkzeuge ist eine installierte Java-Runtime-Umgebung.

Im Gegensatz zu den bisher beschriebenen Anwendungen benötigen *Garmin*-Geräte vektorbasierte Karten, die aus den *OpenStreetMap*-Vektordaten erzeugt werden. Diese werden einmal pro Woche für den ganzen Planeten und täglich für einzelne Kontinente, Länder und Regionen bereitgestellt [11]. Aufgrund der Datenmenge sollte man sich auf ein möglichst kleines Gebiet beschränken, das die Region der benötigten Garmin-Karte umfasst, und das `PBF`-Dateiformat verwenden.

Aus diesem Gebiet wird mithilfe des Java-Werkzeugs *Osmosis* die Kartenregion herausgeschnitten.

```
osmosis --rb file="region.osm.pbf" \
  --bb left=x0 bottom=y0 right=x1 top=y1 clipIncompleteEntities=true \
  --wx file="karte.osm"
```

Da diese Operation sehr lange dauert, bietet sich für kleinere Regionen (Urlaubsgebiete, Bundesländer) alternativ eine Online-Abfrage mithilfe der Overpass-API [12] an.

```
curl --data-urlencode "data=\
( node(süd,west,nord,ost);\
  <;\
);\
out;" \
-o karte.osm http://overpass-api.de/api/interpreter
```

Diese muss mit dem Werkzeug *Splitter* weiter zunächst in Teilregionen zerlegt werden.

```
java -Xmx2048M -jar splitter.jar karte.osm
```

Gemeinsam mit den Datendateien wird eine Steuerdatei `template.args` erzeugt, die für das Generieren der Karte benötigt wird.

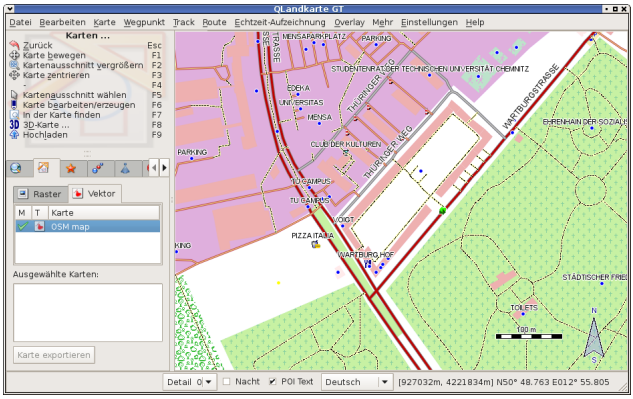
Für die eigentliche Kartenerzeugung werden noch einen Kartenstil und eine zugehörige `TYP`-Datei, die das Kartenlayout definieren, benötigt. Da eine eigene Erstellung nicht ganz einfach ist, bietet es sich an, auf vorhande Vorlagen zurückzugreifen, hier werden die des OpenStreetMap-Nutzers *Computerteddy* [13] genutzt. Die Dateien werden im gleichen Verzeichnis wie die Kartendaten abgelegt bzw. ausgepackt.

Der Aufruf

```
java -Xmx2048M -jar /opt/mkgmap/mkgmap.jar \
  --max-jobs \
  --remove-short-arcs \
  --latin1 \
  --reduce-point-density=10 \
  --mapname="Kartenname" \
  --gmapsupp -c template.args \
  --family-id=42 --style-file=teddy teddy.typ
```

erzeugt eine Datei `gmapsupp.img` mit der fertigen Karte.

Ist die Karte generiert, kann sie mit dem Programm *QLandkarteGT* kontrolliert werden. Dazu muss zunächst die Datei `osmmap.tdb` und anschließend die eigentliche Karte `gmapsupp.img` geladen werden.



Garmin-Karte in QLandkarteGT

Diese fertige Karte wird nun im Massenspeichermodus auf das Garmin-Gerät übertragen und im Verzeichnis `garmin` der SD-Karte abgelegt. Nach einem Reboot des Gerätes kann die neue Karte genutzt werden.

Literatur und Online-Ressourcen

[1] http://wiki.openstreetmap.de – OpenStreetMap Wiki.

[2] Ramm, Topf: *OpenStreetMap. Die freie Weltkarte nutzen und gestalten.* 3. Auflage 2010. Lehmanns New Media.

[3] Robinson et al: *Elements of Cartography.* 6th ed. 1995. John Wiley & Sons.

[4] https://addons.mozilla.org/de/firefox/addon/live-http-headers/ – Live HTTP Headers

[5] http://ruessel.in-chemnitz.de/osm/bigmap.html – Python-Script `bigmap.py`.

[6] http://ruessel.in-chemnitz.de/osm/qbigmap – QBigMap.

[7] http://www.openlayers.org – JavaScript-Bibliothek OpenLayers.

[8] http://leafletjs.com – JavaScript-Bibliothek Leaflet.

[9] http://code.google.com/p/big-planet-tracks/ – Android-App Big Planet Tracks.

[10] http://mobac.sourceforge.net/ – Mobile Atlas Creator.

[11] http://download.geofabrik.de/ – Download von OpenStreetMap-Vektordaten.

[12] http://overpass-api.de – Overpass API.

[13] http://wiki.openstreetmap.org/wiki/User:Computerteddy – Computerteddys Typ- und Styledateien für Garmin-Geräte.

[14] http://www.mkgmap.org.uk/ – mkgmap.

[15] http://ruessel.in-chemnitz.de/osm/clt2013 – Webseite zum Vortrag.

Gnuplot für Anfänger

Uwe Berger
bergeruw@gmx.net

Ein bekanntes Sprichwort meint: „Ein Bild sagt mehr als tausend Worte...“. In Verbindung mit nüchternen und grauen Datenreihen bietet das Kommandozeilen-Tool gnuplot schier unerschöpfliche Möglichkeiten diese auf einfache Art und Weise anschaulichen zu visualisieren. Dass man dazu nicht unbedingt tief greifende Vorkenntnisse haben muss, soll diese kleine Einführung zeigen. An Hand eines grundlegenden Beispiels wird die Verwendung von gnuplot von einem Anfänger für Anfänger erklärt.

1 Motivation

Seit einiger Zeit experimentiert der Autor dieses Beitrages mit diversen Messsensoren. Dabei standen ursprünglich nur die elektronische Ansteuerung und Messwertabfrage unter dem Betriebssystem Linux im Vordergrund[1]. Nachdem neben einigen Temperatursensoren aber auch Messfühler zur Bestimmung des Luftdrucks, der Luftfeuchtigkeit und der Helligkeit dazu kamen, entwickelte sich das Projekt zu einer Langzeitaufzeichnung verschiedener wetterrelevanter Kenngrößen[2]. Spätestens ab hier reifte der Wunsch heran den Verlauf der permanent ermittelten Datenreihen in geeigneter Form zusammenzufassen und in Diagrammform auf einer kleinen Webseite[3] zu visualisieren.

Es wurde also vor allem eine performante und ressourcensparende Lösung zur automatischen Generierung von Diagrammen gesucht, die der dafür vorgesehenen Rechentechnik[4] gerecht wird. Nach einigen interessanten Versuchen mit verschiedenen php-Erweiterungen (z.B. pChart[5]), erwies sich gnuplot[6] als ideales und flexibles Tool zur dynamischen Erzeugung von „Wetterkurven“.

1 Vortrag CLT2012: „Busverkehr unter Linux“ (http://chemnitzer.linux-tage.de/2012/vortraege/853)
2 http://bralug.de/wiki/Wetterdaten_mit_Linux_aufzeichnen_und_verarbeiten
3 soweit erreichbar: http://bergeruw.dyndns.org/~bergeruw/temp/
4 http://bralug.de/wiki/Wetterdaten_mit_Linux_aufzeichnen_und_verarbeiten#Rechner-Hardware
5 http://www.pchart.net/
6 http://www.gnuplot.info/

Lässt man sich nicht von der sehr umfangreichen Dokumentation[7] abschrecken, findet man auch als Anfänger schnell einen Einstieg in das Kommandozeilen-Tool gnuplot. Binnen weniger Minuten zaubert man das erste Diagramm in Form einer Grafikdatei auf die Festplatte, die z.B. mit der üblichen HTML-Anweisung[8] in eine Webseite integriert werden kann. Dabei erleichtern diverse gnuplot-Eigenschaften das Einbinden in eigene Programmumgebungen, um diese Prozesse zu automatisieren.

2 Grundlagen

Wie in der gnuplot-FAQ[9] erläutert, handelt es sich bei dem Tool um eine urheberrechtlich geschützte, aber frei verteilbare Software. Obwohl der Name „gnuplot" dies zwar suggeriert, hat das Programm nichts mit dem GNU-Projekt[10] oder der GPL[11] zu tun. Es stehen Versionen für verschiedene derzeit geläufige Betriebssysteme zur Verfügung, so u.a. auch für Linux, Windows und OSX.

Nach einer erfolgreichen Installation des entsprechenden Softwarepakets, meldet sich das Kommandozeilen-Tool nach seinem Aufruf mit folgender bzw. ähnlicher Ausgabe zurück:

```
$ gnuplot

   G N U P L O T
   Version 4.2 patchlevel 6
   last modified Sep 2009
   System: Linux 2.6.32-45-generic

   Copyright (C) 1986 - 1993, 1998, 2004, 2007 - 2009
   Thomas Williams, Colin Kelley and many others

   Type `help` to access the on-line reference manual.
   The gnuplot FAQ is available from http://www.gnuplot.info/faq/

   Send bug reports and suggestions <http://sourceforge.net/projects/gnuplot>


Terminal type set to 'wxt'
gnuplot>
```

Listing 1: Aufruf gnuplot-Shell

Innerhalb der nunmehr geöffneten gnuplot-Shell könnte man eigentlich sofort mit der Erstellung des ersten Diagramms loslegen. Aber vorab sollen einige Grundlagen

7 http://www.gnuplot.info/documentation.html
8 Grafik in HTML-Seite einbinden: <img src="diagramm.png" alt="Wetterkurve">
9 http://www.gnuplot.info/faq/faq.html
10 http://www.gnu.org/
11 General Public License

zur Arbeit mit gnuplot erläutert werden.

Steuerung von gnuplot über interne Parameter und Befehle

Die darzustellende Diagrammform, Linienarten, Achsenbeschriftungen, Legenden usw. werden über zahlreiche interne Parameter und Variablen festgelegt. Dabei sind deren Inhalte mit sinnvollen Anfangswerten vordefiniert. Das Setzen von Werten erfolgt mit dem set-Befehl, gefolgt vom jeweiligen Variablennamen, dem neuen Inhalt sowie weiteren kontextabhängigen Optionen. Beispiel:

```
gnuplot> set title 'Das ist ein Diagramm', offset 20, 20
```

Neben dem eigentlichen Aussehen des Diagramms können u.a. auch eigene Variablen und Funktionen definiert, spezielle Ein- und Ausgabeformate der darzustellenden Datenpunkte festgelegt und komplexe Rechenoperationen ausgeführt werden. Eine detaillierte Beschreibung aller Parameter und Möglichkeiten findet man in der Dokumentation[12]. Mit dem help-Befehl in der gnuplot-Shell lässt sich auch eine Online-Hilfe aufrufen, z.B.:

```
gnuplot> help set title
```

Die momentanen Inhalte von internen Einstellungen, selbst definierten Variablen, Funktionen usw. sind mit dem show-Befehl einsehbar. Dabei listet ein:

```
gnuplot> show all
```

sämtliche Parameter und Definitionen auf. Ersetzt man das „all“ durch einen konkreten Parameternamen, wird nur dessen Wert angezeigt:

```
gnuplot> show title
```

Eine ausführliche Übersicht zu den vielfältigen vom Tool angebotenen Diagrammarten, mit den jeweils dabei zu setzenden Einstellungen, findet man ebenfalls auf den entsprechenden Webseiten[13] des gnuplot-Projektes.

Sind alle Einstellungen getroffen worden, können mit dem plot-Befehl die Datenreihen in einem entsprechenden Diagramm dargestellt werden.

Ausgabeformate

Recht bald kommt die Frage auf, in welchem Format gnuplot die Diagramme ausgeben kann. Wirft man einen Blick in die Dokumentation oder bemüht die Online-Hilfe mittels des Befehls:

```
gnuplot> help set terminal
```

findet man neben den gängigen Grafikformaten auch zahlreiche interessante Exoten. So ist es z.B. auch möglich, Diagramme als ASCII-Grafik auf einer Textkonsole

12 http://www.gnuplot.info/documentation.html
13 http://www.gnuplot.info/screenshots/index.html

auszugeben.

gnuplot-Skript und Plot-Daten übergeben

Zur Erstellung eines gnuplot-Diagramms sind, wie oben erläutert, diverse interne Parameter und Variablen zu setzen sowie geeignete Befehlsfolgen aufzurufen. Je nach Gegebenheit können dabei mehr oder weniger umfangreiche Skripte entstehen, die dem Kommandozeilen-Tool übergeben werden müssen. Folgende Methoden sind dafür vorgesehen:

- Interaktion in der gnuplot-Shell: gnuplot bringt eine eigene Shell (siehe Listing 1) mit, in der Befehle direkt abgesetzt werden können. Diese Methode bietet sich vor allem zum Testen einzelner Passagen umfangreicher Skripte an.
- Skript-Übergabe bei Programmstart mittels Kommandozeile: Sämtliche Befehle werden in einer einfachen Textdatei aufgeführt, welche als Aufrufparameter angegeben wird. Weiterhin bietet die Aufrufoption -e die Möglichkeit, Befehle direkt an gnuplot zu übergeben:

  ```
  $ gnuplot -e 'f(x)=sin(x); plot f(x); pause 5'
  ```

- Pipes[14]: Wie bei UNIX-Programm üblich, kann auch gnuplot Befehle über eine Pipe empfangen:

  ```
  $ echo 'f(x)=sin(x); plot f(x); pause 5' | gnuplot
  ```

- Standard-Eingabe (stdin)[15]: Vor allem für den gnuplot-Aufruf aus eigenen Programmen heraus, ist diese Methode interessant:

  ```
  $ echo 'f(x)=sin(x);plot f(x);pause 5' > plot.txt; gnuplot < plot.txt
  ```

Neben den Befehlen zur Steuerung der gnuplot-Ausgabe werden die darzustellenden Datenreihen benötigt, wenn nicht gerade ein Graph einer mathematisch berechenbaren Funktion gezeichnet werden soll. Die Übergabe der Daten kann in zwei Formen erfolgen, die jeweils durch entsprechende Optionen des plot-Befehls bestimmt wird.

Daten im gnuplot-Skript selbst:

```
# ...
plot '-'
1 1
3 4
5 6
e
```

Listing 2: Datenreihen am Ende des gnuplot-Skript

Das „-“-Zeichen im plot-Befehl (Listing 2) gibt an, dass ab der nächsten Zeile die

14 http://de.wikipedia.org/wiki/Pipe_%28Informatik%29
15 http://de.wikipedia.org/wiki/Stdin

einzelnen Datenreihen im Skript enthalten und von dort einzulesen sind. Das Ende des Datenabschnittes wird durch ein „e“ markiert.

Daten in einer eigenen Textdatei:

```
# Datei: daten.txt
1 1 5
3 4 8
5 6 9
```

Listing 3: Inhalt Datei „daten.txt“

```
# ...
plot 'daten.txt' using 1:3
```

Listing 4: plot-Befehl zum Darstellen von Daten aus Textdatei „daten.txt“

Dem plot-Befehl (Listing 4) wird der Name der Textdatei mitgegeben, in der die darzustellenden Daten zu finden sind (Listing 3). Die Option „using 1:3“ definiert beispielhaft, dass die 1. und 3.Spalte der Messreihen zu verwenden sind.

Beide Formen haben gemeinsam, dass sich jede Datenreihe auf einer Zeile befinden muss und die einzelnen Werte jeweils durch ein Leerzeichen oder Tabulator getrennt werden. Verwendet man ein davon abweichendes Trennzeichen, kann man dies durch eine entsprechende gnuplot-Anweisung mitteilen, z.B.:

```
set datafile seperator '|'
```

Damit sollten die wichtigsten Grundlagen zur Erstellung eines gnuplot-Skriptes erklärt worden sein. Weiterführende Informationen sind in der Dokumentation zu finden.

3 Pflicht

In der Folge werden die notwendigen gnuplot-Befehle zur Generierung eines einfachen Diagramms beispielhaft entwickelt.

Inhaltlich soll der Verlauf der Innen- und Außentemperatur über die Zeit, also zwei Kurven in einem Diagramm dargestellt werden. Die Werte liegen in einer externen Textdatei vor, die einzelnen Werte sind durch ein „|“-Zeichen separiert und der Zeitstempel hat ein spezielles Format:

```
# Timestamp|Innentemp.|Aussentemp.
2013-01-01T21:28:42|18.75|5.9375
2013-01-01T21:29:43|18.6875|5.9375
2013-01-01T21:30:45|18.75|5.9375
2013-01-01T21:31:46|18.8125|5.9375
...
```

Listing 5: Inhalt Datei temperatur.txt

Grafisches Ausgabeformat, Gesamtgröße und Schriftart/-größe des Diagramms sowie Name der Zieldatei werden festgelegt:

```
set terminal png font arial 8 size 900, 400
set output 'diagramm.png'
```

Definition des Trennzeichens zwischen den Datenspalten, x-Werte sind Zeitangaben sowie deren Ein- und Ausgabeformat (zur späteren Beschriftung der x-Achse):

```
set datafile separator '|'

set xdata time"

set timefmt '%Y-%m-%dT%H:%M:%S'
set format x "%H.%M\n%d.%m"
```

Gitternetzlinien in den angegebenen Intervallen anzeigen (eine sinnvolle Einteilung in x-Richtung wird von gnuplot aus den Daten selbst berechnet). Die Beschriftung der x- und y-Skala erfolgt automatisch. Abschließend werden die darzustellenden Wertebereiche in x- und y-Richtung gesetzt:

```
set ytics 5"
set xtics autofreq"
set grid ytics xtics

set autoscale y"
set autoscale x"

set xrange ['01.12.2013T21:04:22':'03.12.2013T21:04:22']
set yrange [-15:30]"
```

Den Titel der y-Achse und des Diagramms insgesamt setzen (die x-Achse wird im vorliegenden Beispiel ohne Titel dargestellt):

```
set ylabel '°C'
set title "Temperaturverlauf über 48 Stunden\naktuell: 03.12.2013 21:04:22"
```

Diagramm mit den Werten aus Datei „temperatur.txt“ (Listing 5) zeichnen. Dabei werden jeweils Außen- und Innentemperatur (Spalten 2 und 3) über die Zeit (Spalte 1) als geschlossene Linien abgetragen. Mit der Option „title“ wird der Text innerhalb der Legende festgelegt. Die Farbwahl der beiden Kurvenlinien wird gnuplot überlassen:

```
plot \
'temperatur.txt' using 1:3 title 'aussen' with lines, \
'temperatur.txt' using 1:2 title 'innen'  with lines
```

Sämtliche anderen internen gnuplot-Parameter gehen mit ihren Default-Werten in die Berechnung des Diagramms ein. Als Ergebnis wird eine Grafikdatei „diagramm.png“ erzeugt (Bild 1).

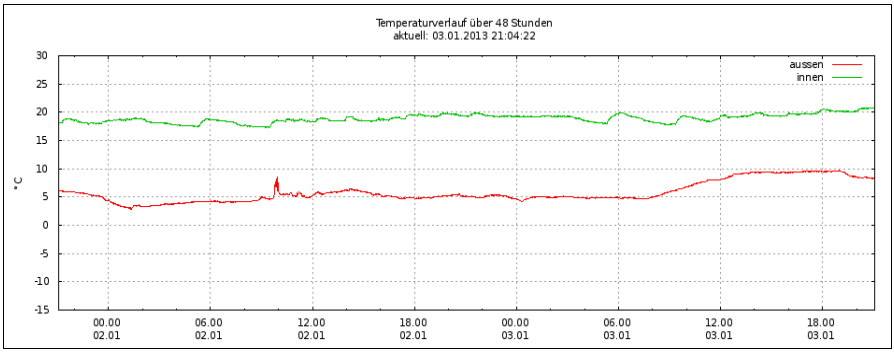


Bild 1: Diagramm „Temperaturverlauf“

Abschließend ist anzumerken, dass dieses und weitere Diagramme auf der Webseite[16] des Autors mit Hilfe von mehreren Tcl[17]-Skripten dynamisch, in einem Zyklus von ca. 2 Minuten, generiert werden[18]. Dabei wird gnuplot via Standard-Eingabe (stdin) gesteuert. Die darzustellenden Daten werden jeweils aus einer SQLite[19]-Datenbank gelesen und, analog dem obigen Beispiel, mittels einer temporären Textdatei an gnuplot übergeben.

4 Kür

Bei intensiverer Beschäftigung mit gnuplot selbst und/oder den verschiedensten Diagrammarten, findet man immer wieder Dinge, die zu interessanten Experimenten anregen.

Wärmediagramme (Heatmaps)

Als Wärmediagramm[20] (engl. Heatmap) bezeichnet man die dreidimensionale Darstellung eines Sachverhaltes in einem zweidimensionalen Diagramm. Die Werte der 3.Dimension werden dabei als Punkte oder Flächen mit den entsprechenden Farbwerten im xy-Koordinatensystem eingezeichnet. Landkarten mit farbigen Höheninformationen entsprechen z.B. dieser Darstellungsform.

16 http://bergeruw.dyndns.org/~bergeruw/temp/index.php
17 http://www.tcl.tk/
18 http://bralug.de/wiki/Wetterdaten_mit_Linux_aufzeichnen_und_verarbeiten#Scripte
19 http://www.sqlite.org/
20 http://de.wikipedia.org/wiki/Heatmap

Die Erzeugung eines solchen Diagramms ist mit wenigen gnuplot-Befehlen erledigt und wird an einigen einfachen Beispielen[21] in der Dokumentation beschrieben. Neben den bereits bekannten Anweisungen (z.B. zur Definition des Ausgabeformats, Formatierung der x- und y-Achse und Diagrammtitel,) spielen folgende Einstellungen eine entscheidende Rolle bei der Erzeugung von Wärmediagrammen:

```
# ...
set view map

set cbrange \[-25.0:50.0\]
set palette defined (0 "blue",17 "#00ffff",33 "white",55 "yellow",\
70 "red",100 "#990000", 101 "grey")
set cblabel '°C'

plot 'heatmap_data.txt' using 1:2:3 with image
```

Dabei wird zuerst die 3D-Sicht als „Landkarte mit Höheninformationen“ eingestellt. Danach erfolgt die Definition des Farbverlaufs für die 3.Dimension, deren Wertebereich sowie die Beschriftung der Farbskala. Mit dem letzten Befehl werden die speziell aufbereitetet Daten (1.Spalte: x-Wert; 2.Spalte: y-Wert, 3.Spalte: z-Wert, repräsentiert durch die Farbe) als „Bild“ im Koordinatensystem abgetragen.

Trägt man z.B. den Temperaturverlauf über Kalendertag (x-Achse) und Tagesuhrzeit (y-Achse) mit prägnanten Farben für die jeweiligen Temperaturwerte ab, entstehen sehr interessante und anschauliche Bilder[22].

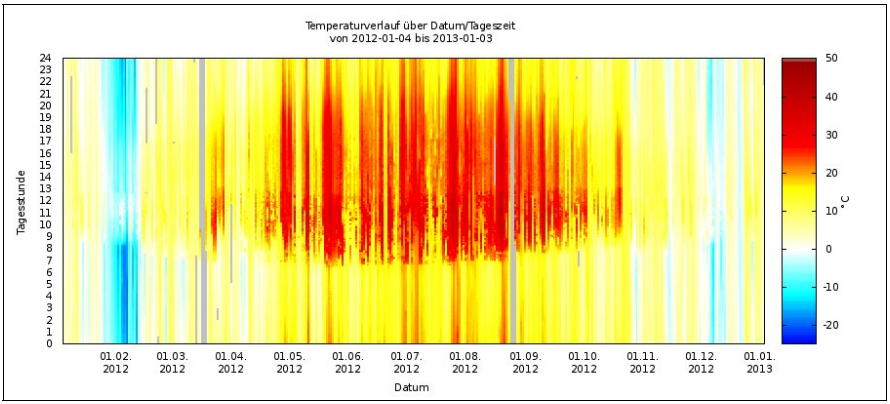


Bild 2: Temperaturverlauf über ein Jahr als Heatmap

Man erkennt beispielsweise auf einen Blick an welchen Tagen und zu welcher Uhrzeit die kältesten oder wärmsten Temperaturen gemessen wurden. Hat man Diagramme aus mehreren Jahren vorliegen, kann man diese sehr leicht nach verschiedenen Kriterien untereinander vergleichen.

21 http://gnuplot.sourceforge.net/demo/heatmaps.html
22 http://bergeruw.dyndns.org/~bergeruw/temp/heatmaps.php

Ausgabeformat tkcanvas

Mindestens einer der gnuplot-Entwickler war Tcl/Tk-Fan! Anders ist nicht zu erklären, dass es das Ausgabeformat „tkcanvas“ gibt:

```
# ...
set terminal tkcanvas size 600, 400
set output 'canvas.tk'
# ...
```

Listing 6: Setzen Terminaltyp „tkcanvas“

Ein darauf folgender plot-Befehl erzeugt eine Textdatei mit dem Namen „canvas.tk“, welche den Quelltext der Prozedur „gnuplot“ enthält. Innerhalb dieser Prozedur sind alle Tk-Anweisungen zur Erzeugung sämtlicher Elemente des entsprechend vorher definierten Diagramms aufgeführt. Einziges Übergabeparameter der Prozedur ist der Name eines Canvas-Widget[23], auf das die Generierungsbefehle der Grafikelemente angewendet werden.

```
# ...
canvas .diagramm
pack .diagramm

# ...Erzeugung des Diagramms via gnuplot (beachte Listing 6)

source canvas.tk
gnuplot .diagramm
# ...
```

Mit solchen oder ähnlichen Programmzeilen sind interessante Tcl/Tk-Anwendungen[24] realisierbar, die ohne spezielle Bibliotheken zur Erzeugung von Diagrammen auskommen.

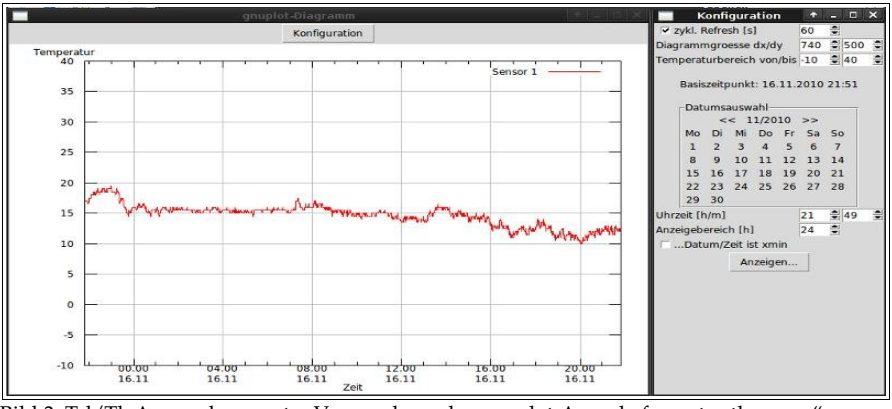


Bild 3: Tcl/Tk-Anwendung unter Verwendung des gnuplot-Ausgabeformats „tkcanvas“

23 http://www.tcl.tk/man/tcl8.5/TkCmd/canvas.htm

24 http://bralug.de/wiki/Tcl/Tk-Benutzeroberfl%C3%A4chen_f%C3%BCr_gnuplot_programmieren

Himbeertorte - Raspberry Pi zum Anfassen

Andreas Heik
andreas.heik@linux-tage.de

Aus der Vision, Computertechnik für den schmalen Geldbeutel technisch interessierten Jugendlichen verfügbar zu machen entstand ein kreditkartengroßen Einplatinencomputer, der *Raspberry Pi* [1].
Mit diesem Artikel zum Workshop möchten wir den Gedanken aufgreifen und mit praktischen Hinweisen zum anfassen und experimentieren anregen.

1 Raspberry Pi

Der Raspberry Pi ist als kreditkartengroßer Einplatinen-Rechner erhältlich. Passende Gehäuse werden als Zubehör angeboten.
Auf der Platine arbeitet ein Prozessor der *ARM11-Familie* zusammengefasst mit einer *VideoCore IV* Grafikeinheit in einem SoC *(System on a Chip)*. Das Betriebssytem, natürlich ein Linux sowie Anwendungen und Daten finden auf einer SD-Karte Platz. Über einen Ethernetport und 2 USB-Anschlüsse kommuniziert der Raspberry Pi mit seiner Umgebung. In der GPU dekodierte Videos in HD-Auflösung werden über HDMI oder Composite Video ausgegeben. [2]
An einer Stiftleiste herausgeführte Low-Level Peripherie wie GPIO, UART, I^2C- und SPI-Bus laden zum basteln und programmieren ein.

2 Quickstart

Für die Inbetriebnahme des Raspberry Pi ist ein Netzteil mit Micro-USB Stecker und einer Belastbarkeit von wenigstens *1 A* erforderlich. Das Betriebssystem wird auf einer SD-Karte mit mindestens 2 GB, besser 4 GB Speicherkapazität installiert. Bei der Auswahl empfiehlt sich eine SD-Karte mit hoher Schreib-/Lesegeschwindigkeit. Eine eingekreiste Zahl auf der SD-Karte gibt dazu Auskunft.
Der HDMI-Ausgang des Raspberry Pi verbindet man am einfachsten mit einem Monitor oder TV-Gerät. Alternativ kann die Grafikausgabe auch über den Composite-Ausgang erfolgen. Tastatur und gegebenenfalls eine Maus werden an die USB-Ports angeschlossen. Eine Funktastatur mit Maus reduziert den Stromverbrauch und hält den zweiten USB-Port beispielsweise für einen USB-WLAN Stick frei.

Als Betriebssystem wird von der Raspberry Pi Foundation die auf Debian basierte Distribution *Raspbian Wheezy* [3] empfohlen. Die Installation des entpackten Image auf eine SD-Karte erfolgt am einfachsten mit dem Werkzeug *dd*.

```
sudo dd if=2012-12-16-wheezy-raspbian.img of=/dev/SD-Karte bs=1M
```

Das vorgeschaltete *sudo* beschafft die erforderlichen Rechte für das blockweise Schreiben auf die Gerätedatei der SD-Karte.
Nach übertragen des Image befinden sich auf der SD-Karte eine 56 MB große */boot-Partition* (vfat-formatiert) und eine ca. 2 GB große Partition mit dem *Root-Filesystem* (ext4-formatiert).
SD-Karten mit vorinstalliertem *wheezy*-Image sind auch erhältlich.

Beim ersten Start des Raspberry Pi erscheint ein Konfigurationsmenü. Ist eine Tastatur angeschlossen empfiehlt sich jetzt die Konfiguration der *Tastatur*, der *Spracheinstellung* und der *Zeitzone*.
Wurde der Raspberry Pi an ein kabelgebundenes Netzwerk mit DHCP-Server angeschlossen ist zu diesem Zeitpunkt auch die Anmeldung mittels Secure Shell (*SSH*) möglich.
Die Anmeldung erfolgt mit dem Loginnamen *pi* und dem Passwort *raspberry*.
Das Konfigurationsmenü kann auch in einer SSH-Sitzung mit dem Kommando *sudo raspi-config* aufgerufen werden. Mit der Option *expand_rootfs* wird das Root-Filesystem an die Größe der SD-Karte angepasst. Die Aktualisierung des Systems erfolgt anschließend mit den Kommandos:

```
sudo apt-get update
sudo apt-get upgrade
```

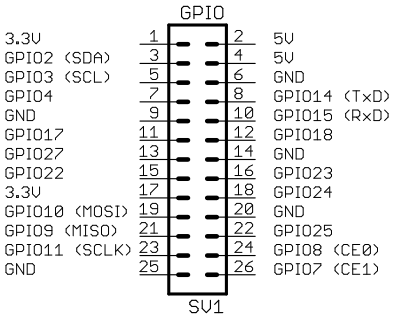
An dieser Stelle sind alle Voraussetzungen erfüllt, den Raspberry Pi zu erkunden.
Wer sich von den Multimedia-Fähigkeiten der GPU überzeugen möchte spielt ein Video im H.264-Format, vielleicht *Big Bug Bunny*[1] ab.
Die Kommandozeile mit Ausgabe des Tons über den HDMI-Ausgang lautet dazu:

```
omxplayer --adev hdmi big_buck_bunny_1080p_h264.mov
```

[1] http://www.bigbuckbunny.org/

Die an einer Stiftleiste herausgeführten GPIO-Pins [4] machen den Raspberry Pi zu einer interessanten Plattform für hardwarenahe Projekte. Doch bevor diese Anschlüsse mit Tastern, LEDs, Sensoren oder anderen Bausteinen verbunden werden beachten Sie unbedingt die Hinweise zu Belastbarkeit und Spannungsfestigkeit.

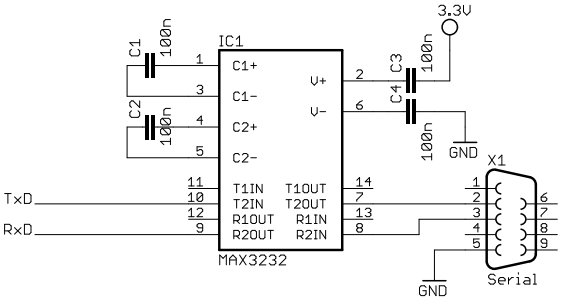
- 3.3V Spannung mit 50mA belastbar
- 5V wird vom Netzteil versorgt
- Signalpegel der GPIOs ist 3.3V (Eingänge sind nicht 5V-tolerant)
- Pinbelegung von der Board-Revision abhängig



Pinbelegung der Stiftleiste (Board Revision 2)

Serielle Schnittstelle

Soll der Raspberry Pi ohne Tastatur und Monitor betrieben werden erweist sich ein Zugang über die serielle Konsole als recht nützlich. Die erforderlichen Anschlüsse sind über die Pins *TxD* und *RxD* an der Stiftleiste herausgeführt. Für eine Verbindung mit der RS232-Schnittstelle eines PC ist jedoch eine Pegelanpassung erforderlich. Diese realisiert der IC *MAX 3232* (*3.3V-Typ*) mit minimaler Außenbeschaltung.

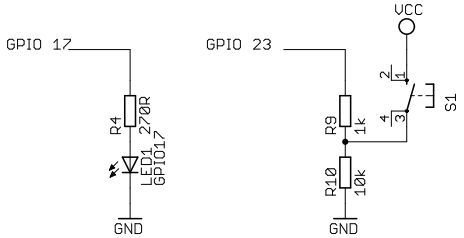


Als Terminalprogramme auf der PC-Seite eignen sich beispielsweise *minicom* oder *screen*. Die Datenrate ist auf 115200bit/s und das Datenformat auf 8N1 voreingestellt. Der Raspberry Pi gibt die Meldungen des Kernel auf der seriellen Schnittstelle aus und der init-Prozess bindet ein *getty* an die serielle Schnittstelle.

```
screen /dev/ttyUSB0 115200
```

GPIO-Pins

Die über die Stiftleiste herausgeführten GPIO-Pins sind frei programmierbar. Damit lässt sich jeder dieser Anschlüsse als Eingang oder Ausgang nutzen. Um bei offenen Eingängen einen definierten Pegel zu erhalten können integrierte Pullup-Widerstände softwareseitig aktiviert werden.
Zum Schutz vor Überlastung empfiehlt sich an den Ausgängen der Einsatz von Low-Current-LEDs oder Treibertransistoren.



Für die Programmierung der GPIO-Pins sind Module und Bibliotheken für viele Programmiersprachen verfügbar. Einfache Tests lassen sich in Shell-Skripten beispielsweise über das */sys-Filesystem* implementieren.

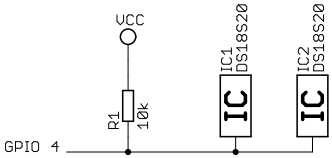
```
# GPIO 17 als Ausgang
echo "17" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio17/direction

# GPIO 17 auf High-Pegel
echo "1" > /sys/class/gpio/gpio17/value
```

Bussysteme

Bei der Anbindung von Bausteinen wie Speicher, Controller, Sensoren und dergleichen kommen oft Bussysteme zum Einsatz. Zu den typischen Vertretern gehören neben I^2C- und SPI- auch der 1-Wire-Bus. Die beiden erstgenannten Bussysteme werden hardwareseitig im SoC des Raspberry Pi unterstützt und deren Anschlüsse sind über die Stiftleiste herausgeführt.

1-Wire-Bus

Bausteine am 1-Wire-Bus werden neben der Stromversorgung mit nur einer Signalleitung angeschlossen. Die Datenübertragung erfolgt asynchron wobei der Raspberry Pi die Rolle des Masters übernimmt. Die Adressierung der Slaves basiert auf eindeutigen 64bit ROM-Adressen.

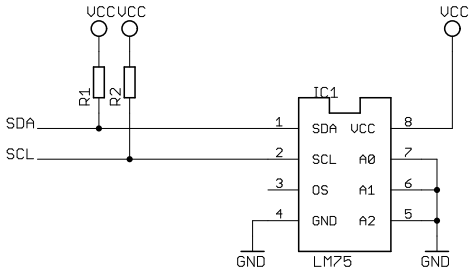


Der Kernel der *wheezy*-Distribution enthält Module für die Unterstützung des 1-Wire-Bus am *GPIO 4* und für verschiedene 1-Wire-Geräteklassen.
Das folgende Beispiel zeigt das Laden der Kernelmodule für den 1-Wire-Bus und einen Temperatursensor vom Typ *DS18S20*. Der 1-Wire-Bus präsentiert sich danach mit seinen Slaves im */sys-Filesystem*.

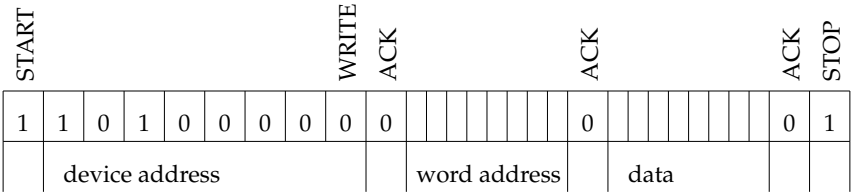
```
modprobe wire
modprobe w1-gpio
modprobe w1-therm
cat /sys/bus/w1/devices/22-00000022a744/w1_slave
 58 01 4b 46 7f ff 08 10 f9 : crc=f9 YES
 58 01 4b 46 7f ff 08 10 f9 t=21500          <-- 21.5 Grad Celsius
```

I²C-Bus

Der auch als Two-Wire-Interface (*TWI*) bezeichnete Bus wird häufig für die Kommunikation zwischen Bausteinen benutzt. Zwei Signalleitungen für Takt und Daten (*SCL* und *SDA*) bilden den Bus. Die Datenübertragung erfolgt synchron nach dem Master-Slave-Prinzip und unterstützt Taktraten zwischen 100 kHz und 3.4 MHz.



Die Pullup-Widerstände der Signalleitungen sind bereits auf dem Raspberry Pi-Board integriert.
Die Adressierung der Slaves erfolgt meist mit 8bit Adressen wobei das niederwertigste Bit über Lese- oder Schreiboperation auf dem Slave entscheidet. Das Beispiel zeigt die Kommunikationssequenz mit einem EEPROM-Baustein *24CXX* an der Geräteadresse $0x50$ beim Beschreiben einer Adresse mit einem Datenbyte.



In der *wheezy*-Distribution enthaltene Werkzeuge wie die *i2c-tools* benötigen als Schnittstelle zum I²C-Bus die zugehörigen Gerätedateien im */dev-Filesystem*.
Durch laden der Kernelmodule *i2c-bcm2708* und *i2c_dev* wird diese Schnittstelle angelegt. Damit die Module beim nächsten Start automatisch geladen werden entfernen Sie *i2c-bcm2708* aus der Blacklist */etc/modprobe.d/raspi-blacklist.conf* und fügen Sie *i2c_dev* der Liste der zu ladenden Module */etc/modules* hinzu.

Das Werkzeug *i2cdetect* scannt den I²C-Bus auf angeschlossene Slaves. Die Kommunikation mit den Slaves unterstützen Programme wie *i2cget*, *i2cdump* und *i2cset*.

```
apt-get install i2c-tools
modprobe i2c-bcm2708 i2c_dev
# scan bus 1
i2cdetect -y 1
```

Hinweis: Der Rapberry Pi Revision 2 stellt den I²C-Bus mit der *ID 1* an der Stiftleiste bereit. Die Adressierung der Slaves erfolgt *ohne* Schreib/Lese-Bit.

Das Python-Module *python-smbus*[2] setzt ebenfalls auf die I²C-Gerätedateien */dev/i2c-?* auf und stellt Funktionen für die Kommunikation mit I²C-Slaves bereit.
Im folgenden Code-Beispiel wird ein EEPROM-Baustein *24CXX* mit der Geräteadresse $0x50$ programmiert. Die Funktion *write_i2c_block_data()* füllt den EEPROM mit der übergebenen Daten-Liste ab Adresse *reg*. Da der Schreibzyklus den EEPROM laut Datenblatt für maximal 5ms blockiert wartet ein *sleep*. Die Iterationsfunktion über die Länge der Daten-Liste liest nun byteweise ab Adresse *reg* und gibt die Werte wieder aus.

[2] http://wiki.erazor-zone.de/wiki:linux:python:smbus:doc

```
# EEPROM 24C02
import sys, time, smbus

# bus, address, register
bus = 1
address = 0x50
reg = 0x01
data  = [ 0x48, 0x61, 0x6c, 0x6c, 0x6f, 0x20, 0x43, 0x4c, 0x54 ]

eeprom = smbus.SMBus(bus)

print "I2C: write to device 0x%02X" % address
try:
  eeprom.write_i2c_block_data(address, reg, data)
except IOError, err:
  print "Error accessing I2C address: 0x%02X" % address
  exit(-1)
time.sleep(0.1)

print "I2C: read from device 0x%02X" % address
for i in range(len(data)):
  try:
    ret = eeprom.read_byte_data(address, reg+i)
  except IOError, err:
    print "Error accessing I2C address: 0x%02X" % address
    exit(-1)
  print "%c" % ret
```

Erweiterte Funktionalität bietet das Python-Module *quick2wire* mit Funktionen für die Programmierung von GPIO-Pins, den I^2C- und SPI-Bus.
Mit *bcm2835* ist eine C-Bibliothek für die Programmierung der Low-Level Peripherie des Raspberry Pi verfügbar. Dabei erfolgt der Hardwarezugriff über in den Speicher gemappte Register (*mmap()*).

Literatur

[1] *Raspberry Pi*
http://www.raspberrypi.org

[2] *Wikipedia Artikel über Raspberry Pi*
http://en.wikipedia.org/wiki/Raspberry_Pi

[3] *Raspbian Distribution*
http://www.raspbian.org/

[4] *Low-level peripherals*
http://elinux.org/RPi_Low-level_peripherals

Linux-Versteher dank `strace`

Harald König
H.Koenig@science-computing.de
koenig@linux.de
http://www.science-computing.de/

`strace` ist ein wahres Wundertool in Linux, man muss es nur einsetzen – und kann damit sehr viel über die Abläufe und Interna von Linux lernen. Mit `strace` können einzelne oder mehrere Prozesse zur Laufzeit auf System-Call-Ebene „beobachtet“ werden. Damit lassen sich bei vielen Problemen sehr einfach wertvolle Informationen zum Debuggen gewinnen: welche Config-Dateien wurden wirklich gelesen, welches war die letzte Datei oder Shared-Library vor dem Crash usw. Im Unterschied zu interaktiven Debuggern läuft das zu testende Programm mit `strace` mehr oder weniger in Echtzeit ab, und man erhält schon während des Programm-Laufs jede Menge Ausgaben zu allen erfolgten Kernel-Calls, so dass man den Ablauf des Prozesses „live“ mitverfolgen bzw. den abgespeicherten Output nachträglich bequem auswerten kann.

Auch bei Performance-Problemen kann man mit `strace` interessante Informationen gewinnen: wie oft wird ein Syscall ausgeführt, wie lange dauern diese, wie lange „rechnet“ das Programm selbst zwischen den Kernel-Calls. Man kann auch den kompletten I/O eines Programms (Disk oder Net) mit `strace` recht elegant mitprotokollieren und später offline analysieren (oder auch „replay“en, bei Bedarf sogar in „Echtzeit“ Dank präziser Time-Stamps).

Der Vortrag soll anregen, rätselhafte UNIX-Effekte, -Probleme oder Programm-Crashes neu mit `strace` zu betrachten und damit hoffentlich (schneller) zu Lösungen und neuen Erkenntnissen zu gelangen.

1 Let's use `strace`

`strace` kennt viele Optionen und Varianten. Hier können nur die in meinen Augen wichtigsten angesprochen und gezeigt werden – die Man-Page (RTFM: `man strace`) birgt noch viele weitere Informationsquellen und Hilfen.

1.1 Erste Schritte

Je nachdem, was man untersuchen will, kann man (genau wie mit Debuggern wie `gdb`) ein Kommando entweder mit `strace` neu starten, oder aber man kann sich an einen bereits laufenden Prozess anhängen und diesen analysieren:

```
$ strace emacs
```

bzw. für einen schon laufenden `emacs`

```
$ strace -p $( pgrep emacs )
```

und wenn es mehrere Prozesse gleichzeitig zu tracen gibt (z.B. alle Instanzen des Apache `httpd`):

```
$ strace $( pgrep httpd | sed 's/^/-p/' )
```

Das an einen Prozess angehängte `strace -p ...` kann man jederzeit mit `CTRL-C` beenden, das untersuchte Programm läuft dann normal und ungebremst weiter. Wurde das zu testende Programm durch `strace` gestartet, dann wird durch `CTRL-C` nicht nur `strace` abgebrochen, sondern auch das gestartete Programm beendet.

`strace` gibt seinen gesamten Output auf `stderr` aus, man kann jedoch den Output auch in eine Datei schreiben bzw. die Ausgabe umleiten, dann jedoch inklusive dem `stderr`-Outputs des Prozesses (hier: emacs) selbst:

```
$ strace -o OUTFILE emacs
$ strace emacs 2> OUTFILE
```

Üblicherweise wird jeweils eine Zeile pro System-Call ausgegeben. Diese Zeile enthält den Namen der Kernel-Routine und deren Parameter, sowie den Rückgabewert des Calls. Der Output von `strace` ist sehr `C`-ähnlich, was nicht sehr wundern dürfte, da das Kernel-API von Linux/UNIX ja als ANSI-C-Schnittstelle definiert ist (meist didaktisch sinnvoll gekürzte Ausgabe):

```
$ strace cat /etc/HOSTNAME
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], [/* 132 vars */]) = 0
open("/etc/HOSTNAME", O_RDONLY)         = 3
read(3, "harald.science-computing.de\n", 32768) = 28
write(1, "harald.science-computing.de\n", 28) = 28
read(3, "", 32768)                      = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```

Selbst ohne `C`-Programmierkenntnisse lässt sich diese Ausgabe verstehen und vernünftig interpretieren. Details zu den einzelnen Calls kann man in den entsprechenden Man-Pages nachlesen, denn alle System-Calls sind dokumentiert in Man-Pages (`man execve ; man 2 open ; man 2 read write` usw.). Nur aus der Definition des jeweiligen Calls ergibt sich, ob die Werte der Argumente vom Prozess an den Kernel übergeben, oder aber vom Kernel an den Prozess zurückgegeben werden (bspw. den String-Wert als zweites Argument von `read()` und `write()` oben).

Für einige System-Calls (z.B. `stat()` und `execve()`) erzeugt `strace` mit der Option `-v` eine „verbose“ Ausgabe mit mehr Inhalt. Auf der Suche nach mehr Informationen (`stat()`-Details von Dateien, oder komplettes Environment beim `execve()`) kann dies oft weiterhelfen.

```
$ strace -e execve cat /etc/HOSTNAME
```

```
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], [/* 132 vars */]) = 0
harald.science-computing.de
$
$ strace -v -e execve cat /etc/HOSTNAME
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], ["LESSKEY=/etc/lesskey.bin", "MANPATH=/usr/
local/man:/usr/loca"..., "XDG_SESSION_ID=195", "TIME=\\t%E real,\\t%U user,\\t%S sy"...,
 "HOSTNAME=harald", "GNOME2_PATH=/usr/local:/opt/gnom"..., "XKEYSYMDB=/usr/X11R6/lib/X11
/XKe"..., "NX_CLIENT=/usr/local/nx/3.5.0/bi"..., "TERM=xterm", "HOST=harald", "SHELL=/bi
n/bash", "PROFILEREAD=true", "HISTSIZE=5000", "SSH_CLIENT=10.10.8.66 47849 22", "VSCMBOO
T=/usr/local/scheme/.sche"..., "CVSROOT=/soft/.CVS", "GS_LIB=/usr/local/lib/ghostscrip".
.., "TK_LIBRARY=/new/usr/lib/tk4.0", "MORE=-sl", "WINDOWID=14680077", "QTDIR=/usr/lib/qt
3", "JRE_HOME=/usr/lib64/jvm/jre", "USER=harald", "XTERM_SHELL=/bin/bash", "TIMEFORMAT=\
t%R %3lR real,\t%U use"..., "LD_LIBRARY_PATH=/usr/local/nx/3."..., "LS_COLORS=no=00:fi=0
0:di=01;34:l"..., "OPENWINHOME=/usr/openwin", "PILOTPORT=usb:", "XNLSPATH=/usr/share/X11
[ ... 22 lines deleted ...]
rap"..., "_=/usr/bin/strace", "OLDPWD=/usr/local/nx/3.5.0/lib/X"...]) = 0
```

Ein besonderer Modus von `strace` bekommt man mit der Option `-c`. Hier wird nicht mehr jeder einzelne Kernel-Call ausgegeben, sondern nur ganz am Ende eine Statistik über die Anzahl der einzelnen Calls sowie die jeweils verbrauchte CPU-Zeit. Diese Ausgabe mag für einen ersten Überblick bei Performance-Problemen ganz nützlich sein, jedoch erfährt man nur wenige Details über den Programmablauf und sieht den Ablauf des Programms nicht mehr „live“ (für die Kombination von Live-Trace und abschließender Statistik gibt es dann die Option `-C`).

1.2 Ausgabe von `strace` einschränken

Der Output von `strace` kann schnell *sehr* umfangreich werden, und das synchrone Schreiben der Ausgabe auf `strerr` oder auch eine Log-Datei kann erheblich Performance kosten. Wenn man genau weiß, welche System-Calls interessant sind, kann man die Ausgabe auf einen bzw. einige wenige Kernel-Calls beschränken (oder z.B. mit `-e file` alle Calls mit Filenames!). Das verlangsamt den Programmablauf weniger und vereinfacht das spätere Auswerten des `strace`-Outputs erheblich. Allein die Option `-e` bietet sehr viel mehr Möglichkeiten. Hier nur ein paar einfache Beispiele, welche sehr oft ausreichen, alles Weitere dokumentiert die Man-Page:

```
$ strace -e open             cat /etc/HOSTNAME
open("/etc/HOSTNAME", O_RDONLY)         = 3

$ strace -e open,read,write  cat /etc/HOSTNAME
open("/etc/HOSTNAME", O_RDONLY)         = 3
read(3, "harald.science-computing.de\n", 32768) = 28
write(1, "harald.science-computing.de\n", 28harald.science-computing.de
) = 28
read(3, "", 32768)                      = 0



$ strace -e open,read,write  cat /etc/HOSTNAME > /dev/null
open("/etc/HOSTNAME", O_RDONLY)         = 3
read(3, "harald.science-computing.de\n", 32768) = 28
write(1, "harald.science-computing.de\n", 28) = 28
```

```
read(3, "", 32768)                      = 0

$ strace -e file             cat /etc/HOSTNAME
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], [/* 132 vars */]) = 0
open("/etc/HOSTNAME", O_RDONLY)         = 3
```

1.3 Mehrere Prozesse und Kind-Prozesse tracen

`strace` kann auch mehrere Prozesse gleichzeitig tracen (mehrere Optionen `-p PID`) bzw. auch alle Kind-Prozesse (Option `-f`) mit verfolgen. In diesen Fällen wird in der Ausgabe am Anfang jeder Zeile die PID des jeweiligen Prozesses ausgegeben. Alternativ kann man die Ausgabe auch in jeweils eigene Dateien je Prozess schreiben lassen (mit Option `-ff`).

Wenn man nicht genau weiß, was man denn eigentlich tracen muss, dann ist die Verwendung von `-f` oder `-ff` angesagt, da man ja nicht ahnen kann, ob evtl. mehrere (Unter-) Prozesse oder Skripte involviert sind. Ohne `-f` oder `-ff` könnten sonst beim Trace wichtige Informationen von weiteren Prozessen entgehen. In meinen einfachen Beispielen mit `emacs` ist dieser selbst auch schon ein Wrapper-Shell-Skript. Hier nun ein paar Varianten als kleine Denksportaufgabe zum Grübeln, wie die `bash` intern so tickt:

```
$ strace -e execve bash -c true
execve("/bin/bash", ["bash", "-c", "true"], [/* 132 vars */]) = 0

$ strace -e execve bash -c /bin/true
execve("/bin/bash", ["bash", "-c", "/bin/true"], [/* 132 vars */]) = 0
execve("/bin/true", ["/bin/true"], [/* 130 vars */]) = 0
$ strace -e execve bash -c "/bin/true ; /bin/false"
execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"], [/* 132 vars */]) = 0

$ strace -f -e execve bash -c "/bin/true ; /bin/false"
execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"], [/* 132 vars */]) = 0
execve("/bin/true", ["/bin/true"], [/* 130 vars */]) = 0
execve("/bin/false", ["/bin/false"], [/* 130 vars */) = 0

$ strace -o OUTFILE -f -e execve bash -c "/bin/true ; /bin/false" ; grep execve OUTFILE
21694 execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"], [/* 132 vars */]) = 0
21695 execve("/bin/true", ["/bin/true"], [/* 130 vars */]) = 0
21696 execve("/bin/false", ["/bin/false"], [/* 130 vars */]) = 0

$ strace -o OUTFILE -ff -e execve bash -c "/bin/true ; /bin/false" ; grep execve OUTFILE*
OUTFILE.22155:execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"], [/*...*/]) = 0
OUTFILE.22156:execve("/bin/true", ["/bin/true"], [/* 130 vars */]) = 0
OUTFILE.22157:execve("/bin/false", ["/bin/false"], [/* 130 vars */]) = 0
```

1.4 Time-Stamps und Performance-Betrachtungen

Bei Performance-Fragen ist es oft hilfreich, dass jede Ausgabe von `strace` zu einem Kernel-Call mit einem Time-Stamp versehen werden kann. Gibt man die Option `-t`

ein-, zwei- oder dreimal an, kann man verschiedene Formate des Zeit-Stempels wählen: HH:MM:SS, HH:MM:SS.Mikrosekunden, sowie die „UNIX-Zeit“ in Sekunden und Mikrosekunden seit 1970 GMT/UTC. Die letzte Form ist für spätere Berechnungen mit den Time-Stamps am einfachsten, die anderen beiden Formen meist leichter lesbar und einfacher mit anderen Time-Stamps z.B. aus `syslog` oder anderen Log-Files abzugleichen, wenn es um die Analyse ganz bestimmter Events geht.

Mit der Option `-r` bekommt man wahlweise auch die Zeit-Differenz zum letzten Kernel-Call in Mikrosekunden (das kann man mit `-ttt` auch später noch selbst ausrechnen und ergänzen). Alle Time-Stamps beziehen sich immer auf den Aufruf des Kernel-Calls (Kernel Entry), man bekommt direkt keinen Time-Stamp vom Zeitpunkt des Rücksprungs vom Kernel zurück in das Programm.

Die Option `-T` fügt am Ende der Ausgabe-Zeile noch in spitzen Klammern die Verweildauer im Kernel an. Nur mit dieser Angabe kann man sich bei Bedarf den absoluten Time-Stamp der Rückkehr aus dem Kernel selbst errechnen, denn bis zum nächsten Kernel-Call könnte viel Zeit im User-Mode verbracht werden, so dass der nächste Kernel-Entry-Time-Stamp nicht immer aussagekräftig ist.

```
$ strace -t    -e open,read,write,close  cat /etc/HOSTNAME > /dev/null
22:22:25 open("/etc/HOSTNAME", O_RDONLY) = 3
22:22:25 read(3, "harald.science-computing.de\n", 32768) = 28
22:22:25 write(1, "harald.science-computing.de\n", 28) = 28
22:22:25 read(3, "", 32768)             = 0
22:22:25 close(3)                       = 0

$ strace -tt   -e open,read,write,close  cat /etc/HOSTNAME > /dev/null
22:22:46.314380 open("/etc/HOSTNAME", O_RDONLY) = 3
22:22:46.314566 read(3, "harald.science-computing.de\n", 32768) = 28
22:22:46.314698 write(1, "harald.science-computing.de\n", 28) = 28
22:22:46.314828 read(3, "", 32768)      = 0
22:22:46.314945 close(3)                = 0

$ strace -ttt  -e open,read,write,close  cat /etc/HOSTNAME > /dev/null
1325971387.860049 open("/etc/HOSTNAME", O_RDONLY) = 3
1325971387.860143 read(3, "harald.science-computing.de\n", 32768) = 28
1325971387.860186 write(1, "harald.science-computing.de\n", 28) = 28
1325971387.860249 read(3, "", 32768)    = 0
1325971387.860283 close(3)              = 0

$ strace -r    -e open,read,write,close  cat /etc/HOSTNAME > /dev/null
     0.000067 open("/etc/HOSTNAME", O_RDONLY) = 3
     0.000076 read(3, "harald.science-computing.de\n", 32768) = 28
     0.000042 write(1, "harald.science-computing.de\n", 28) = 28
     0.000039 read(3, "", 32768)        = 0
     0.000033 close(3)                  = 0

$ strace -ttT  -e open,read,write,close  cat /etc/HOSTNAME > /dev/null
22:24:09.339915 open("/etc/HOSTNAME", O_RDONLY) = 3 <0.000010>
22:24:09.340001 read(3, "harald.science-computing.de\n", 32768) = 28 <0.000012>
22:24:09.340052 write(1, "harald.science-computing.de\n"
```

Meist wird man zunächst den `strace`-Output in eine Datei schreiben und anschließend „Offline“ verschiedene Auswertungen machen. Hier aber ein kleines „Live“-Beispiel zur Frage „was treibt denn der `acroread` die ganze Zeit im Hintergrund?“:

```
$ strace -p $( pgrep -f intellinux/bin/acroread ) -e gettimeofday -tt
22:48:19.122849 gettimeofday({1325972899, 122962}, {0, 0}) = 0
22:48:19.123038 gettimeofday({1325972899, 123055}, {0, 0}) = 0
22:48:19.140062 gettimeofday({1325972899, 140154}, {0, 0}) = 0
22:48:19.140347 gettimeofday({1325972899, 140482}, {0, 0}) = 0
22:48:19.142097 gettimeofday({1325972899, 142261}, {0, 0}) = 0
22:48:19.146188 gettimeofday({1325972899, 146335}, {0, 0}) = 0
22:48:19.159878 gettimeofday({1325972899, 160023}, {0, 0}) = 0
CTRL-C
$ strace -p $( pgrep -f intellinux/bin/acroread ) -e gettimeofday -t 2>&1 \
  | cut -c-8 | uniq -c
    136 22:48:26
    138 22:48:27
    140 22:48:28
CTRL-C
```

oder als weiteres Beispiel eine kleine Statistik über 10 Sekunden:

```
$ strace -p $( pgrep -f intellinux/bin/acroread )  -c & sleep 10 ; kill %1
[1] 24495
Process 18559 attached - interrupt to quit
[ Process PID=18559 runs in 32 bit mode. ]
Process 18559 detached
System call usage summary for 32 bit mode:
$ % time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 79.13    0.010992           5      2114           poll
  7.88    0.001094           0      2939           clock_gettime
  7.00    0.000972           0      2124      1716 read
  3.07    0.000426           0      1388           gettimeofday
  2.93    0.000407           1       406           writev
  0.00    0.000000           0         1           restart_syscall
  0.00    0.000000           0         2           time
------ ----------- ----------- --------- --------- ----------------
100.00    0.013891                  8974      1716 total
```

1.5 Warum bricht denn XY ab, welche Dateien sind relevant?

Wenn ein Programm abbricht, ob nun mit einer „segmentation violation“ oder einer erkennbaren vernünftigen(?) Fehlermeldung, hat man schon halb gewonnen: jetzt muss man „nur“ das Programm mit `strace` laufen lassen, hoffen, dass damit der Fehler noch reproduzierbar ist und schon findet man kurz vor Ende der `strace`-Ausgabe vielleicht den ganz offensichtlichen Hinweis auf einen der letzten Schritte, der zum Problem geführt hat.

So könnte das Problem daran liegen, dass eine benötigte Shared-Library nicht gefunden wird. Hier ein Beispiel von `emacs` der seine `libX11.so.6` nicht mehr finden kann:

```
25243 open("/usr/lib64/x86_64/libX11.so.6", O_RDONLY) = -1 ENOENT (No such file or directory)
25243 stat("/usr/lib64/x86_64", 0x7fff4f3568c0) = -1 ENOENT (No such file or directory)
25243 open("/usr/lib64/libX11.so.6", O_RDONLY) = -1 ENOENT (No such file or directory)
25243 stat("/usr/lib64", {st_mode=S_IFDIR|0755, st_size=212992, ...}) = 0
25243 writev(2, [{"/usr/bin/emacs", 14}, {": ", 2}, {"error while loading shared libra"...
25243 exit_group(127)                   = ?
```

Oft bekommt man bei Problemen postwendend als Antwort „Wir haben seit dem letzten erfolgreichen Lauf gar nichts geändert“. Hier kann man mit `strace` leicht eine komplette Liste aller geöffneten oder auch gesuchten und nicht gefundenen Dateien erstellen und dann deren Time-Stamps im Filesystem überprüfen, ob denn wirklich nichts neu installiert oder geändert wurde:

```
$ strace -o OUT -e open,execve -f emacs
$ grep \" OUT | cut -d\" -f2 | sort | uniq -c | sort -nr | less
     35 /opt/kde3/share/icons/hicolor/icon-theme.cache
      9 /home/harald/.icons/DMZ/index.theme
      6 /usr/share/emacs/site-lisp/anthy/leim-list.el
      5 /usr/share/emacs/23.3/etc/images/splash.xpm
      4 /usr/share/icons/hicolor/icon-theme.cache
$ ls -ltcd $( grep \" OUT | cut -d\" -f2 | sort -u )  | less
-r--r--r--  1 root   root         0 Jan  7 23:43 /proc/filesystems
-r--r--r--  1 root   root         0 Jan  7 23:43 /proc/meminfo
drwx------  2 harald users    28672 Jan  7 23:40 /home/harald/.emacs.d/auto-save-list/
drwxrwxrwt 48 root   root     12288 Jan  7 23:33 /tmp//
-rw-r--r--  1 root   root   3967384 Jan  7 14:29 /usr/local/share/icons/hicolor/icon-theme.cache
-rw-r--r--  1 root   root  94182076 Jan  7 11:34 /usr/share/icons/hicolor/icon-theme.cache
-rw-r--r--  1 root   root    289877 Jan  7 11:34 /etc/ld.so.cache

$ strace -o OUT -e open,execve -e signal=\!all  -f emacs /etc/HOSTNAME
$ cut -d\" -f2 OUT | sort -u > FILELIST

$ strace -o OUT -e open -f emacs /etc/HOSTNAME
$ grep open OUT | grep -v ENOENT | cut -d\" -f2 | sort -u > FILELIST
```

oder eine Liste aller Zugriffe, die fehlschlugen (vgl. Unterschied zwischen `-e open` und `-e file`!):

```
$ strace -o OUT -e file,execve -f emacs
$ grep ENOENT OUT | cut -d\" -f2 | sort -u
$ grep ' = -1 E'  OUT | cut -d\" -f2 | sort -u
```

Wenn man eine komplette `FILELIST` erstellt, lässt sich leicht prüfen, welche Config-Dateien beispielsweise gelesen wurden, und kann dann nach der Datei suchen, aus der ein magischer String (Makro, Config-Setting, komischer Text in der Ausgabe o.ä.) kommt, um das Problem einkreisen zu können. Man sollte jedoch zuvor noch Einträge mit `/dev/` oder `/proc/` etc. entfernen:

```
$ strace -o OUT -e open,execve  -f emacs /etc/HOSTNAME
$ grep -v ENOENT OUT | grep \" | cut -d\" -f2 | sort -u > FILELIST
$ egrep -v '^/dev/|^/proc/' FILELIST | xargs grep "Welcome to GNU Emacs"
Binary file /usr/bin/emacs-gtk matches
```

1.6 Welche Login-Skripte wurden ausgeführt

... ist hiermit auch ganz allgemein zu prüfen und zu verifizieren. Entweder kann man der entsprechenden Shell vertrauen, dass diese sich z.B. mit deren Option `--login` wirklich identisch zu einem „echten“ Login verhält. Mit dieser Annahme reicht es,

eine File-Liste zu erstellen und diese zu analysieren. Im gezeigten Beispiel bleibt die Datei FILELIST unsortiert, denn auch die Reihenfolge der Login-Skripte ist natürlich interessant:

```
$ strace -o OUT -e open -f bash --login -i
exit
$ grep -v ENOENT OUT | grep \" | cut -d\" -f2 > FILELIST1
$ strace -o OUT -e file -f bash --login -i
exit
$ grep \" OUT | cut -d\" -f2 > FILELIST2

$ egrep "^/etc|$HOME" FILELIST1
/etc/profile
/etc/bash.bashrc
/etc/venus/env.sh
/etc/bash.local
/home/koenig/.bashrc
/etc/bash.bashrc
/etc/venus/env.sh
/etc/bash.local
/home/koenig/.bash_profile
/home/koenig/.profile.local
/home/koenig/.bash_history
/etc/inputrc
/home/koenig/.bash_history
```

Wenn man der Shell nicht so ganz traut, dann sollte man die „Quelle“ eines Logins selbst tracen! Als Nicht-root kann man dies z.B. noch mit `xterm -ls` versuchen. Oder man traced als root direkt den SSH-Daemon an Port 22 oder ein getty-Prozess mit allen Folge-Prozessen (s.u. zu Sicherheitsaspekten eines strace-Logs von sshd o.ä.):

```
$ lsof -i :22
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd    2732 root    3u  IPv4  15551      0t0  TCP *:ssh (LISTEN)
$ strace -p 2732 -ffv -e file -o sshd.strace ...
```

1.7 Noch mehr Daten sehen

Bei lokalen Dateien reicht eine Datei-Liste und grep, um nach der Quelle eines bestimmten Strings zu suchen. Mit Netzwerkverbindungen o.ä. geht das leider nicht so einfach. Für lange Strings bei `read()`, `write()`, usw. werden per default nur die ersten 32 Zeichen ausgegeben, damit kann man im Output von strace nicht gut suchen. Hier ist es nützlich, die maximale Länge der Strings zu erhöhen. Oft reichen die ersten 999 Zeichen, manchmal braucht man wirklich die kompletten Records („viel hilft viel“):

```
$ strace -s999    -o OUTFILE -f wget http://www.google.de/
```

(oder gleich `-s 999999`) um nach einem bekannten String zu suchen (z.B. HTTP). Damit findet man den System-Call `recvfrom()` sowie `read()` und kann sich auf den

read()-Call konzentrieren, wenn man nur die empfangenen Daten untersuchen will (für den gesendeten Daten-Stream entsprechend sendto()). Einen guten Überblick über die Netz-Verbindung liefert

```
$ strace -o OUT -e connect,bind,recvfrom,sendto,read wget http://www.google.de/
```

Um nun alle empfangenen Daten zu extrahieren, kann man z.B. die Strings der Rückgabewerte des recvfrom()-Calls extrahieren, die ordentlich gequoteten Strings mit dem Shell-Kommando ausgeben und die Quotes konvertieren lassen:

```
$ strace -o OUT -s 999999 -e read wget http://www.google.de/
$ grep 'read(4, "' OUT | cut -d, -f2- | sed -e '1,/google.com/d' \
     -e s'/. [0-9]*) * = [0-9]*$//'  -e 's@^@/bin/echo -en @' | sh
```

Nun stören nur noch die Längen-Records des HTTP-Protokolls, aber der interessante Inhalt ist komplett und durchsuchbar.

2 Probleme

Doch selbst ein so traumhaftes Werkzeug wie strace ist nicht ganz ohne Schatten. Hier ein paar Hinweise und Warnungen, den Rest erzählt ihnen die Man-Page oder google ;-)

2.1 strace stört den Prozess-Ablauf

Die Verwendung von strace ist meist transparent für den überwachten Prozess, so sollte eine „Messung“ mit strace das Messergebnis meist nicht verfälschen.

Jedoch sollte man immer berücksichtigen, dass die Kommunikation zwischen strace und den überwachten Prozess(en) zusätzlichen Rechenaufwand und viele Kontext-Wechsel bedingen. Zumindest bei Performance-Aussagen auf Systemen mit hoher Last sollte man daher besonders vorsichtig sein.

Ab und an hatte ich auch schon Hänger in den Prozessen, meist bei IPC oder mit Realtime-Signalen. „Früher“ (alte Linux-Kernel, vielleicht um 2.6.<klein> und noch älter?!) trat dies häufiger und dann leider reproduzierbar auf – inzwischen eher nicht mehr.

Sollte jedoch ein angehängter Prozess weiter hängen, obwohl strace abgebrochen wurde (strace lässt sich ab und an nur mit einem beherzten kill -9 noch bremsen), dann kann es lohnen, den hängenden Prozessen ein SIGCONT zu schicken (kill -CONT ...), damit diese vielleicht weiterlaufen.

2.2 `ptrace()` ist nicht kaskadierbar

Der System-Call-Tracer `strace` verwendet, ebenso wie `gdb` oder andere Debugger, den System-Call `ptrace()`, um den zu debuggenden Prozess zu kontrollieren und von diesem Informationen zu erhalten. Da immer nur ein einziger Prozess mittels `ptrace()` einen anderen Prozess kontrollieren kann, ergibt sich automatisch die Einschränkung, dass Prozesse, welche schon mit `gdb`/`strace` o.ä. debuggt werden, nicht nochmals mit einem weiteren `strace`-Kommando beobachtet werden können:

```
$ strace sleep 999 &
$ strace -p $( pgrep sleep )
attach: ptrace(PTRACE_ATTACH, ...): Operation not permitted
```

2.3 SUID-Programme tracen

SUID-Programme lassen sich aus Sicherheitsgründen von normalen Benutzern in Linux/UNIX nicht mit SUID debuggen oder (s)tracen, denn damit könnte der ausführende und tracende Benutzer ja beliebig in die Programmausführung unter einer anderen UID eingreifen und z.B. Daten zu/von einem Kernel-Call ändern. SUID-Programme werden daher ohne SUID, also unter der eigenen UID ausgeführt (was im Einzelfall auch nützlich sein kann). Wird `strace` unter `root` ausgeführt, ist dies kein Problem mehr: `root` darf ohnehin alles, damit ist auch das Ausführen unter anderen Benutzer-IDs kein erweitertes Sicherheitsproblem.

Will man nun jedoch in einem komplexeren Skript, welches *nicht* unter `root` laufen soll/darf, ein einzelnes SUID-Kommando (z.B. `sudo`) starten, so kann man dies mit `strace` erreichen, wenn man das `strace`-Binary *selbst* auf `SUID root` setzt.

Aber **Vorsicht**: jedes mit diesem SUID-präparierten `strace` ausgeführten „getracete“ Programm läuft damit automatisch unter `root` – eine große Sicherheitslücke, falls andere Benutzer dieses SUID-`strace` ausführen dürfen! Daher sollte man entweder alle Zugriffsrechte für nicht-Owner entfernen (`chmod go= strace` bzw. `chmod 4100 strace`), oder diese „gefährliche“ Variante von `strace` als Kopie in einem für andere Benutzer unzugänglichen Verzeichnis „verwahren“.

2.4 Für alle Benutzer lesbarer Output von `strace`

Der Output von `strace` kann durchaus „interessante“ Inhalte offenlegen und sollte daher besser nicht für andere Benutzer lesbar sein! So können Teile von Dateien im Output von `read()` angezeigt werden, die nicht lesbar sind (z.B. der eigene private SSH-Key beim Tracen von `ssh . . .`) oder gar die einzelnen Zeichen der Eingabe des eigenen Passworts oder ähnlich sensible Daten. Der Output von `strace`-Sessions durch `root` hat natürlich ein noch viel höheres Potential, unabsichtlich Informationen preiszugeben.

2.5 Dead-Locks

Wenn man „sich selbst“ straced, dann kann man wunderbare Dead-Locks erzeugen, z.B. wenn man einen Teil der Output-Kette für die Ausgabe von `strace` tracen will: die eigene `ssh`-Verbindung oder das `xterm`, in dem `strace` gerade läuft, oder gar der X-Server, auf dem die `strace`-Ausgabe durchblubbert. Wohl dem, der dann noch einen zweiten Zugang zum System hat und das `strace` beenden kann!

3 Ausblick

Neben `strace` gibt es weitere ähnliche Trace-Tools unter Linux, die ebenfalls hilfreich sein können:

`ltrace`: statt „S“ystem-Calls werden hier für die meisten „üblichen“ Shared-Libraries alle „L“ibrary-Calls ge„l“traced. Man erhält noch *viel* mehr Output (was dann auch die Ausführung sehr stark bremst), aber das explizite Tracen einzelner libc-Calls wie z.B. `getenv()` oder auch String-Funktionen wie `strlen()` oder `strcpy()` etc. können Informationen über den Programmablauf geben. Viele Optionen sind gleich wie bei `strace`, aber `ltrace` kann ausschliesslich ELF-Binaries tracen und keine Shell-Skripte o.ä., was in einigen Fällen die Anwendung erschweren kann.

`LTTng` könnte die Zukunft sowohl für Kernel- aus auch für User-Mode-Tracing sein. Leider fehlt mir hierzu bislang noch jegliche Praxis (`strace` sei Dank?!).

4 Conclusion

Nun einfach viel Spaß, Erfolg und neue UNIX-Erkenntnisse beim `stracen`!

Literatur

[1] RTFM: `man strace ltrace ptrace`

[2] http://linux.die.net/man/1/strace Man-Page von `strace`

[3] http://linux.die.net/man/2/ptrace Man-Page von `ptrace()`

[4] http://linux.die.net/man/1/ltrace Man-Page von `ltrace`

[5] http://lttng.org/ Das LTTng Projekt

Nutzung des offenen OCRA-Algorithmus zur Transaktionsabsicherung mit LinOTP beim Online-Banking

Cornelius Kölbel
cornelius.koelbel@lsexperts.de
http://www.lsexperts.de

Mit dem im RFC 6287 spezifizierten OCRA-Algorithmus[1] steht ein Challenge-Response-Verfahren offen und frei zur Verfügung. Der Algorithmus wurde federführend von der *Initiative for Open Authentication* spezifiziert und erlaubt sowohl das Authentisieren eines Benutzers als auch das Signieren einer Nachricht. Das modulare Authentisierungsbackend LinOTP nutzt diesen Algorithmus um ein robustes TAN-Verfahren für Banken zu implementieren. Dabei sind die entstehenden TANs kryptografisch unverwechselbar an die Transaktionen gebunden.

1 LinOTP

Die Notwendigkeit der Zweifaktorauthentisierung und die flexiblen Möglichkeiten dies mit LinOTP[2] umzusetzen, wurden bereits grundlegend in [2] dargestellt.

LinOTP ist ein modulares Authentisierungsbackend, das sehr unterschiedliche Tokentypen unterstützen und auch zahlreiche Benutzerdatenbanken wie LDAP, SQL oder Flatfile gleichzeitig anbinden kann. LinOTP setzt auf Opensource-Komponenten auf und steht zu weiten Teilen selber unter freien Lizenzen. Offene und gut dokumentierte Schnittstellen ermöglichen die leichte Integration in heterogene Infrastrukturen und die Einbindung in bestehende Workflows. LinOTP verwendet bevorzugt Einmalpasswörter[3] zur Authentisierung. Diese können auf unterschiedliche Art und Weise erzeugt und ebenso unterschiedlich von LinOTP verifiziert werden.

2 TAN-Verfahren

Im Online-Banking-Umfeld kommen neben dem leider wenig verbreiteten HBCI[4]-Verfahren, bei dem der Bankkunde seine Transaktionen mit einer Smartcard und einem entsprechenden Lesegerät freigeben kann, seit langer Zeit TAN[5]-Verfahren in Verbindung mit browserbasierten Banking-Portalen zum Einsatz[6].

[1] OATH Challenge-Response Algorithm
[2] http://linotp.org, http://www.lsexperts.de/linotp.html
[3] OTP = One time password
[4] Homebanking Computer Interface
[5] TAN = Transaktionsnummer
[6] Genau genommen kann auch HBCI mit TANs verwendet werden.

In Deutschland sind die Verfahren iTAN, mTAN und chipTAN und Abwandlungen davon am verbreitetsten. Die TAN soll sicherstellen, dass der Bankkunde sich zum Einen bei jeder Transaktion, die er im Online-Portal durchführt, zusätzlich legitimieren muss, aber auch im Idealfall die Transaktionsdaten gegen Manipulationen schützen.

Die iTAN[7], die als eine ausgedruckte Liste von i.d.R. 100 TANs ausgeliefert wird, kann nur in begrenzter Form Schutz vor Manipulation der Transaktionsdaten bieten. Daher wurde diese bei vielen Banken zu Gunsten von mTAN oder chipTAN abgelöst.

Beim mTAN[8]-Verfahren wird dem Bankkunden die Transaktionsnummer mit den Überweisungsdaten per SMS auf sein Mobiltelefon gesendet. Die Nutzung des zweiten Kanals SMS und die zusätzlich mitgeschickten Überweisungsinformationen ermöglichen es dem Kunden, zu überprüfen, ob diese TAN auch genau für diese Transaktion verwendet werden soll. D.h. dies gewährleistet den zusätzlichen Schutz gegen Manipulation der Transaktionsdaten im Online-Banking-Portal. Jedoch zeigen immer wiederkehrende Meldungen [5] und [6], dass die Nutzung von mTAN mit erheblichen Risiken verbunden ist, die nicht nur in der Angreifbarkeit des Endgeräts, sondern auch in der nicht ausreichenden Sicherung des Transportweges (des GSM-Netzes) und der Handhabung von Mobilfunk-Anschlüssen[7] liegen.

Das chipTAN-Verfahren nutzt mit der EC-Karte und einem zusätzlichen Lesegerät zwei unabhängige Faktoren, die bei den TAN-Verfahren heute die bisher höchste Sicherheit versprechen, jedoch für den Kunden vergleichsweise aufwändig und fehleranfällig sein können. So hängt das Einlesen eines Flickercodes stark von Lichtverhältnissen und Kontrasten ab und kann leicht zur Nicht-Durchführbarkeit führen, so dass die Transaktionsdaten per Hand in das Lesegerät eingegeben werden müssen. Die Nutzungsbedingungen der Banken weisen gar auf die Gefahr von durch den Flickercode verursachten epileptischen Anfällen hin.

3 OCRA: OATH Challenge-Response Algorithm

Die *Initiative for Open Authentication*[9], die bereits die OTP-Algorithmen HOTP[3] und TOTP[4] spezifizierte, definiert mit RFC6287 ein Verfahren, das mit Hilfe eines geheimen, symmetrischen Schlüssels ein Challenge-Response-Verfahren abbildet, bei dem anders als bei dem assymetrischen RSA-Algorithmus die Signatur bzw. der Response auf eine Challenge eine menschenleserliche Länge hat. Dies ist wichtig, da in den gedachten Anwendungsfälle des OCRA-Algorithmus es möglich sein muss, den Response per Hand eingegeben zu können.

Das prinzipielle Challenge-Response-Verfahren ist in Abbildung 1 dargestellt.

[7] indizierte TAN
[8] mobile TAN
[9] http://www.openauthentication.org

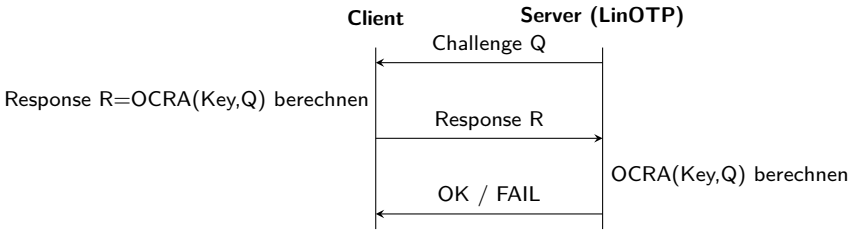


Abbildung 1: Einfache OCRA-Kommunikation

OCRA definiert verschiedene Methoden, um Challenges und die zugehörigen Responses zu berechnen. Dies spiegelt sich in der sogenannten OCRA-Suite wieder.

$$OCRA-1:HOTP-SHA256-6:C-QA64 \tag{1}$$

Durch Doppelpunkte werden dreit Teile voneinander abgetrennt.

OCRA-1 beschreibt die Versionsnummer der OCRA-Definition. Zur Zeit ist nur die Version 1 vorhanden. Spätere Erweiterungen würden sich in einer höheren Versionsnummer widerspiegeln.

HOTP-SHA256 Dies ist die verwendete Kryptofunktion. Dabei wird der in [3] definierte HOTP-Algorithmus mit dem Hashalgorithmus SHA1, SHA256 oder SHA512 verwendet. Zu beachten ist, dass hier immer nur HOTP und nicht TOTP definiert wird!

Die nachgestellte 6 zeigt an, auf wieviele Stellen die Antwort gekürzt wird. Dies ist die Länge des resultierenden OTP-Wertes, der den Response darstellt. Übliche Längen sind hier 4, 6, 8 oder 10.

Der dritte und letzte Teil gibt Auskunft über zusätzliche Parameter:

C zeigt an, dass wie im klassischen HOTP-Algorithmus ein Counter verwendet wird.

Q gibt das Format der Challenge an. So sind bspw.:

- QA64 Eine alphanumerische 64 Zeichen lange Challenge,
- QN10 Eine 10 Zeichen lange numerische Challenge, die besser zur manuellen Eingabe der Challengedaten geeignet ist oder
- QH04 Eine hexadezimale 4 Zeichen lange Challenge.

Weitere Eingabeparameter im dritten Teil sind:

- P In die Berechnung soll ein Passwort mit einfließen.
- S In die Berechnung soll ein Session-Wert mit einfließen.

- T In die Berechnung soll die Zeit (ähnlich TOTP) mit einfließen. Das Zeitintervall wird zusätzlich hinter dem T mit angegeben.

So würde bspw. ein OCRA-Suite **`OCRA-1:HOTP-SHA1-6:QN08-T30S`** bedeuten, dass mit Hilfe des HOTP-SHA1 Algorithmus aus einer 8-stelligen numerischen Challenge ein zeitbasierter (in 30 Sekundenschritten) Response aus 6 Ziffern berechnet wird.

3.1 OCRA Berechnung

Bei der Berechnung kommt wiederum der geheime, symmetrische Schlüssel (*Seed*) zum Einsatz. Zur Berechnung werden nun

- die OCRA-Suite als Textstring (*OCRASuite*),
- ein Trennzeichen (*00*),
- der Counter (C) (wenn in der OCRA-Suite definiert),
- die Challenge (Q),
- das Passwort (P) (wenn in der OCRA-Suite definiert),
- die Session-Information (S) (wenn in der OCRA-Suite definiert),
- die Zeit (T) (wenn in der OCRA-Suite definiert)

konkateniert und mit der Kryptofunktion (bspw. HOTP-SHA1-6) verarbeitet.

$$\begin{aligned} DataInput &= OCRASuite + 00 + C + Q + P + S + T \\ OTP &= HOTP_{SHA1,6}(Seed, DataInput) \end{aligned} \tag{2}$$

D.h. für unterschiedliche Challenges wird der OCRA-Algorithmus unterschiedliche OTP-Werte zurückliefern.[10] Die Nutzung von Counter-, Session- oder Zeitinformationen verhindert Replay-Attacken für wiederverwendete gleiche Challenges.

3.2 Anwendung zur Bestätigung der Transaktionsdaten

Die Idee ist nun, dass eine Client-Applikation oder ein Client-Gerät die entsprechende Challenge entgegennehmen kann - entweder automatisch oder manuell eingetippt - und der Benutzer die Challenge akzeptiert. Wenn der Benutzer nun die zugehörige Response berechnet und zurück an den Server sendet, kann der Server überprüfen, ob die Response stimmt und somit sicherstellen, dass der Benutzer die Challenge gesehen hat und akzeptiert hat.

[10] Anmerkung: Das RFC zu OCRA definiert auch eine gegenseitige Authentisierung von Client und Server, die hier aber nicht betrachtet wird.

Im Online-Banking-Umfeld können nun als Challenge entsprechende Transaktionsdaten gewählt werden. D.h. der Kunde erzeugt erst nachdem er die Transaktionsdaten auf dem Client-Gerät verifiziert hat die TAN (d.h. den OTP-Wert). Diese TAN ist aufgrund der Challenge-Daten auch noch kryptographisch an die Challenge, also die Transaktionsdaten gebunden. Werden die Transaktionsdaten bspw. im Banking-Portal von einem Angreifer abgeändert, wird der Kunde dennoch auf Basis der originalen Transaktionsdaten die TAN erzeugen und das Serverbackend wird feststellen, dass die übermittelte TAN nicht zu den Transaktionsdaten aus dem Banking-Portal passt.

Das OCRA-Verfahren kann offline betrieben werden, da die Übermittlung der Challenge bspw. durch Eintippen in ein Gerät erfolgt und die resultierende TAN weiterhin abgelesen werden kann. Offline bietet dieses Verfahren auch einen größeren Schutz gegen Manipulation des Client-Gerätes.

4 OCRA und LinOTP

LinOTP behandelt alle Tokentypen als Python-Klassen, so dass sich das Hinzufügen eines neuen Tokentypes recht einfach gestaltet. Der OCRA-Token wurde im Python-Modul **`linotp.lib.tokenclass`** als Klasse **`OCRATokenClass`** hinzugefügt und die Standard-Methoden wie **`checkOTP`**, **`getClassType`**, **`resync`** und **`update`** for OCRA implementiert. Das *OCRA Validation Server Profile*[8] sieht ergänzend zum RFC[1]

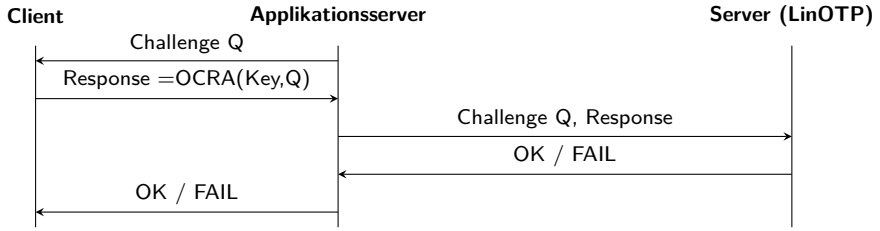


Abbildung 2: Challenge wird von der Applikation erzeugt und zusammen mit dem Response des Clients an LinOTP übergeben.

vor, dass die Challenge nicht unbedingt von dem Validierungsserver erzeugt werden muss, sondern auch von einem Applikationsserver erzeugt werden kann und dann zusammen mit dem Response des Clients an den Validierungsserver übergeben wird. Dies ist in Abbildung 2 dargestellt. LinOTP kann dazu zur Validierung sowohl OTP/Passwort und die challenge entgegennehmen.

```
https://linotpserver/validate/check?user=maria&challenge
    =1234567890&pass=123456
```

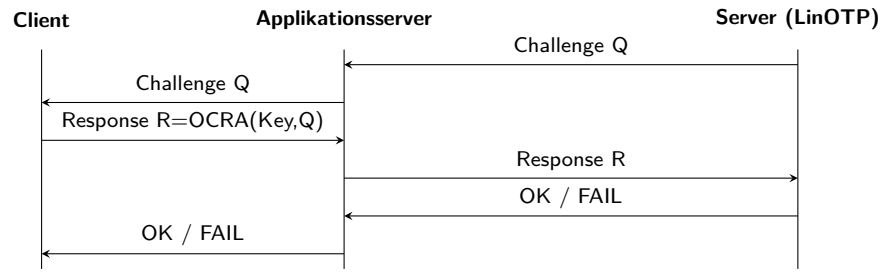


Abbildung 3: OCRA-Kommunikation mit Client, Applikationsserver und LinOTP

Die Challenge kann aber ebenso wie im RFC definiert vom Validierungsserver, also im LinOTP, erzeugt werden. Dies ist in Abbildung 3 dargestellt.

Anders als bei sonstigen OTP-Token wird damit eine Authentisierung nicht in einem einzigen Schritt vom Benutzer angestoßen. Stattdessen geschieht die Authentisierung in zwei Schritten durch Anfordern der Challenge und Beantworten der Challenge im zweiten Schritt. (vgl. Abbildung 3)

/ocra/request fordert von LinOTP eine neue Challenge an. Für den Einsatz bei Banken können dabei die Transaktionsdaten (wie Kontonummer, Überweisungsart, Bankleitzahl usw.) als Freitext übergeben werden.

```
https://linotpserver/ocra/request?user=maria&data=Das ist
    eine Testüberweisung
```

Der Response kann nun wie gehabt über das Interface **/validate/check**[2] gegen den Benutzer validiert werden.

Im Bankenumfeld ist es aber bisweilen nötig, mehrere Transaktionen parallel offen zu halten. Damit LinOTP mehrere ausgesendete Challenges asynchron wieder zuordnen kann, speichert LinOTP intern die Challengeinformationen mit Transaktions-IDs. Das Interface der **OCRATokenClass** ist also um entsprechende Funktionen erweitert um diese Transaktions-IDs verarbeiten zu können. Ebenso reicht das ursprüngliche Authentisierungs-Interface **/validate/check** nicht mehr aus. Ein neues Interface muss den Response mit der Transaktions-ID entgegennehmen. Ein entsprechender Request mit den Bank-Transaktionsinformationen „Das ist eine Testüberweisung“ resultiert in folgender Antwort:

Listing 1: LinOTP output for an OCRA challenge

```
1 "detail": {
2   "data": "lseqr://req?me=Das+ist+eine+Test%C3%BCberwei\
3 ␣␣sung&u=&ch=DasisteineTestberweisung0000itP3bZ9x6b5leo4lLr\
4 ␣␣M3iXXFoiScZgHo32bm&tr=183029118058&si=24fe745bf61f799491c\
```

```
5  ␣␣d20d056a4e6cc30e0021ef3a47aa37d8b1e2b24e2a9e5",
6    "transactionid": "183029118058",
7    "message": "Das␣ist␣eine␣Testüberweisung",
8    "challenge": "DasisteineTestberweisung0000itP3bZ9x6b5leo4\
9  ␣␣lLrM3iXXFoiScZgHo32bm"
10 }, [...]
```

Dabei werden die Transaktions-ID (*transactionid*) und die Challenge-Daten (*challenge*) zurückgeliefert. Abhängig davon, welche OCRA-Suite der OCRA-Token des Benutzers *maria* benutzt, werden die Challenge-Daten unterschiedlich erzeugt. In diesem Fall ist es eine Challenge **QA64**.

Das neue Interface **/ocra/check_t** kann nun auf Basis der Transaktions-ID den Response zur entsprechenden Challenge auch zu einem späteren Zeitpunkt validieren und dazu die *transactionid* und den OTP-Wert bzw. die TAN übergeben:

```
https://linotpserver/ocra/check_t?transactionid
    =183029118058&pass=429510
```

5 QR-TAN zur Vereinfachung der Challenge

In dem vorgestellten Beispiel haben wir gesehen, dass, um eine komplette Bindung an die Transaktionsdaten zu gewährleisten, möglichst alle Transaktionsdaten in die Challenge mit eingehen sollten. Dies lässt sich mit bestehenden Hardware-OCRA-Token, bei denen eine 6-10 stellige Challenge eingetippt wird, allerdings nicht umsetzen.

Daher wurde eine Smartphone-App entwickelt, die die Challenge-Daten per QR-Code einscannen kann und damit das lästige Eintippen der Challenge entfällt. Gleichzeitig lässt sich der QR-Code erheblich schneller und zuverlässiger scannen als der Flickercode beim chipTAN-Verfahren.

Hierzu kann aus dem Wert **data** in Zeile 2 aus Listing 1 ein QR-Code erzeugt werden, der dann mit der QR-TAN App eingescannt wird.
Der eingescannte String enthält verschiedene Daten: **me**, **ch**, **tr**, **si**, **u**. Die Integrität des eingescannten Strings wird anhand der Signatur (**si**) sichergestellt. Die Transaktionsdaten werden in der App nochmals im Klarttext (**me**) angezeigt und erst wenn der Bankkunde die angezeigten Transaktionsdaten als richtig akzeptiert hat wird schließlich der zur Challenge (**ch**) passende Response berechnet und angezeigt.

Ein QR-Code für die QR-TAN App

Optional kann der App im Parameter **u** eine URL mitgegeben werden, so dass die App die TAN zusammen mit der Transaktions-ID (**tr**) selber automatisch an LinOTP senden kann und damit sogar das Eintippen der TAN entfallen kann.

6 Ausblick

Auch weiterhin zeichnet sich LinOTP durch seine Offenheit, Flexibilität und Modularität aus. Somit ließ sich die Erweiterung des OCRA-Algorithmus und des QR-TAN-Verfahrens in einem kurzen Entwicklungszyklus in LinOTP integrieren. Die Flexibilität in der Ansteuerung der Authentisierungsgeräte stellt sicher, dass auch in Zukunft neue Verfahren schnell integriert werden können.

Schon heute können Banken und auch beliebige andere Unternehmen mit LinOTP auf flexible Authentisierungs- bzw. TAN-Verfahren mit einem einheitlichen Backend zurückgreifen, seien es mTAN (mit LinOTP's SMS OTP-Token), klassische OTP-Token, Hardware-OCRA-Token oder QR-TAN Token auf Smartphones.

Literatur

[1] M'Raihi, Rydell, Bajaj, Machani, Naccache: *RFC6287: OATH Challenge-Response Algorithm*. http://tools.ietf.org/html/rfc6287. 2011.

[2] Kölbel: *Starke Zweifaktorauthentisierung mit LinOTP - modular, skalierbar, frei*. Tagungsband, Chemnitzer Linux-Tage 2012.

[3] M'Raihi, Bellare, Hoornaert, Naccache, Ranen: *RFC4226: HOTP Algorithm*. https://www.ietf.org/rfc/rfc4226.txt. 2005.

[4] M'Raihi, Machani, Pei, Rydell: *RFC6238: TOTP Algorithm*. https://www.ietf.org/rfc/rfc6238.txt. 2011.

[5] heise: *Millionenschaden durch mTAN-Betrug*. http://www.heise.de/security/meldung/Millionenschaden-durch-mTAN-Betrug-1763105.html. 6.12.2012.

[6] Polizei Berlin: *Präventionshinweis für Onlinebanking im mTAN-Verfahren*. http://www.berlin.de/polizei/presse-fahndung/archiv/377949/index.html. 13.11.2012.

[7] SC Magazine: *$45k stolen in phone porting scam*. http://www.scmagazine.com.au/News/282310,45k-stolen-in-phone-porting-scam.aspx/0. 6.12.2011.

[8] OATH: *OCRA Validation Server Profile — Version 0.8*. http://www.openauthentication.org/webfm_send/48. 11.2.2011.

Petabyte-Storage leicht gemacht: Ceph

Martin Gerhard Loschwitz
martin.loschwitz@hastexo.com
http://www.hastexo.com

Dieser Beitrag beschäftigt sich mit der Storage-Lösung Ceph, die den Aufbau eines nahtlos in die Breite skalierenden Speichers ermöglicht. Der Text geht zunächst auf die theoretischen Hintergründe von Object Storage ein und erläutert dann die zentralen Funktionsprinzipien von Ceph selbst. Schließlich sind auch die verfügbaren Front-Ends für Ceph Thema.

1 Speicherlösungen im IT-Umfeld

Zwei große Umbrüche lassen sich in der Entwicklung von Storage in den letzten rund 20 Jahren erkennen. Der erste große Übergang war der von einzelnen Storage-Inseln hin zu zentralisierten Speichern, die meist etwas salopp unter den Oberbegriffen SAN und NAS firmieren. Der zweite Wechsel war der von eben jenen zentralisierten Speichern hin zu Lösungen auf Open-Source-Grundlage, die ähnliche Funktionen wie klassische SANs bieten, jedoch in der Anschaffung insgesamt deutlich günstiger sind. Seither scheint die Zeit im Hinblick auf Storage still zu stehen. Am deutlichsten wird das, wenn es um Skalierbarkeit geht.

Zwei Arten von Skalierbarkeit kommen in typischen IT-Setups vor: Auf der einen Seite steht das Skalieren in die Höhe (`Scale-Up`). Dabei erhalten bereits vorhandene Geräte zusätzliche Komponenten, zum Beispiel mehr RAM oder neue Festplatten. Dem gegenüber steht das Skalieren in die Breite (`Scale-Out`) – zusätzliche Geräte erweitern die vorhandenen. Fast immer setzen die Verantwortlichen für IT-Lösungen mittlerweile auf Scale-Out: Webserver-Farmen, denen ein Loadbalancer vorangestellt ist, sind ein typisches Beispiel. Aber auch Datenbanken skalieren immer öfter in die Breite und kommen mit einem Master-Server und vielen Slave-Servern daher, so dass sich die Last durch lesenden Zugriff auf viele verschiedene Systeme verteilt. Stößt das Setup dennoch an die Grenzen seiner Belastbarkeit, lässt es sich problemlos um zusätzliche Server erweitern.

Einzig das Thema Storage kennt bisher nur Scale-Up: Hier liegt stets die gesamte Last auf einer einzelnen Komponente. Und ist ein SAN oder eine der vielen verfügbaren Ersatzkonstruktionen voll und nicht mehr um neue Platten zu erweitern, muss ein zweiter Speicher her, der nicht nur teuer ist, sondern im Handstreich auch den Vorteil des zentral zu administrierenden Storages zunichte macht.

2 Object Storage

Object Storage tritt mit dem erklärten Ziel an, Scale-Out auch beim Thema Storage zu ermöglichen. Es ist nicht der erste Ansatz zur Lösung des Problems, im Gegensatz zu vielen anderen Varianten konnten sich in den letzten Jahren allerdings bereits zwei Object-Storage-Lösungen auch kommerziell etablieren. So betreibt beispielsweise der amerikanische Hosting-Anbieter Rackspace ein Speicherplatz-Angebot auf Grundlage vom Object Store Swift, und die ebenfalls amerikanische Firma Dreamhost bietet eine ähnliche Lösung auf Grundlage von Ceph an, das quasi so etwas wie der Shooting-Star bei Object Storages ist. Wie lösen Object Storages das Versprechen von nahtloser Skalierbarkeit des Speichers ein?

Ein Teil der Antwort auf diese Frage steckt bereits im Namen „Object Storage“ selbst. Anders als konventionelle Speicherlösungen verwalten Ceph & Co. den verfügbaren Speicher nicht block-basiert. Stattdessen wandeln sie jede hochgeladene Information in binäre Objekte um, die sie nach einem festgelegten Schema speichern und verwalten. Weil die physikalischen Grenzen der eigentlichen Festplatten keine Rolle mehr spielen, ist das Storage letztlich beliebig erweiterbar – kommen neue Platten hinzu, bindet die Software diese in ihre Verteilmechanismen ein und nutzt sie ab dem Augenblick, ab dem sie tatsächlich zur Verfügung stehen. Obendrein kümmern sich klassische Object Storages automatisch um das Thema Hochverfügbarkeit, indem sie die abgelegten Objekte automatisch replizieren – fällt eine der genutzten Platten aus, sind die darauf gespeicherten Objekte clusterweit mindestens noch einmal vorhanden, so dass es zu keiner Service-Unterbrechung kommt.

3 Die Ursprünge des Projekts

Die Idee zu Ceph stammt ursprünglich aus der Feder von Sage Weil, der das Projekt als Grundlage seiner Dissertation an der Universität von Kalifornien in Santa Cruz ins Leben rief. Zum besseren Verständnis der Funktionsweise von Ceph hilft die Vorstellung, dass ein Ceph-Setup im Grunde aus zwei Teilen besteht: Dem eigentlichen Object-Store `Ceph` (auch unter der Bezeichnung `RADOS` bekannt), sowie dem Gegenstück auf Seiten des Clients, der den lokalen Zugriff auf die Daten ermöglicht.

An dieser Stelle ist zunächst der Objektspeicher selbst von Interesse: Wie funktioniert er intern, wie organisiert er das Ablegen von Daten und wie kümmert er sich um den reibungslosen Betrieb auch dann, wenn etwas schief geht?

4 Der Object Store: Ceph

Ceph trägt die Verantwortung dafür, jedwede Information, die ihren Weg in den Cluster findet, dort strukturiert abzulegen und entsprechend der Vorgabe des Administrators zu replizieren. Um diese Aufgabe zu meistern, werkeln bei einem typischen

Ceph-Store zwei Diensttypen Hand in Hand: Monitoring-Server (`MONs`) und Object Storage Daemons (`OSDs`).

OSDs bilden in ihrer Gesamtheit den tatsächlich verfügbaren Platz im Object Store. Als OSD kann grundsätzlich jedes Storage-Device dienen, auf dem ein Dateisystem mit Unterstützung für Extended User Attributes vorhanden ist (die Ceph-Entwickler empfehlen zwar Btrfs, doch wird man in der Praxis meist XFS nutzen). Hat ein Server also 14 Festplatten, die Ceph zur Verfügung stehen sollen, so wären diese 14 Platten 14 einzelne OSDs. Die Zahl von OSDs innerhalb einer Ceph-Installation ist weder pro Host noch insgesamt beschränkt – hieraus ergibt sich die Möglichkeit, zusätzliche OSDs nach Belieben in den Speichercluster zu integrieren.

MONs stehen den OSDs zur Seite und spielen eine zentrale Rolle im Hinblick auf den Zugriff durch Clients. Sie verwalten eine Liste der verfügbaren OSDs (`osdmap`) und haben so stets den Überblick über den Zustand des Clusters. Will ein Client den Object Store nutzen, so verbindet er sich zunächst mit den MONs und erhält von den MONs die OSD-Karte, mit deren Hilfe er sich selbst ausrechnen kann, auf welchem jeweiligen OSD die gewünschte Information zu finden ist. Obendrein sind die MONs auch für die Durchsetzung des Quorums im Cluster mittels eines Paxos-Algorithmus zuständig: Fällt bei einem MON die Verbindung zu den übrigen MONs aus, so dass er sich mit weniger als 50% der insgesamt vorhandenen MONs über den Zustand des Clusters einigen kann, so erklärt sich dieser MON für funktionsuntüchtig und verweigert den Dienst. So verhindert Ceph Split-Brain-Szenarien, in denen zwei getrennte Teile des Clusters unkoordinierte Veränderungen der Daten vornehmen und damit für zwangsweisen Datenverlust sorgen.

5 Der CRUSH-Algorithmus

Wie zuvor bereits erwähnt passiert in einer Ceph-Installation viel auf den Clients, die auf den Object Store zugreifen. Zum Verständnis der Vorgänge auf den Clients ist es notwendig, den CRUSH-Algorithmus in seinen Grundsätzen zu verstehen. Denn dieser trägt die Verantwortung für die passende Ablage der Daten im Speicher. Grundsätzlich gilt: Ist die in den Speicher geladene Datei ursprünglich 2 Gigabyte groß, so landet sie nicht als 2 Gigabyte großes Binärobjekt auf einem der OSDs. In der Standardkonfiguration unterteilt der Client stattdessen die Datei in viele binäre Objekte, die jeweils 4 Megabyte groß sind. Diese Objekte weist der Client dann einzelnen *Placement Groups* (kurz *PGs*) zu. Wichtig ist hier die Feststellung, dass eine dieser PGs nicht deckungsgleich mit einer Datei ist; viel mehr werden die zu einer Datei gehörenden Objekte mutwillig auf verschiedene PGs verteilt.

Dann greift der CRUSH-Algorithmus – CRUSH ist dabei die Abkürzung für *Controlled Replication Under Scalable Hashing*. Mit Hilfe des CRUSH-Algorithmus kann der Client sich für jede Placement Group ausrechnen, auf welchem OSD die Daten landen müssen. Im Anschluss nutzen die OSDs selbst den gleichen Algorithmus, um untereinander die benötigten Replicas der Placement Groups anzulegen. Das Lesen

von einem Ceph-Speicher funktioniert übrigens nach genau dem gleichen Schema; mit dem Namen einer Datei findet der Client heraus, zu welcher PG die binären Objekte der Datei gehören und errechnet sich anschließend per CRUSH, auf welchen OSDs die Objekte liegen.

Praktisch ist, dass ein Client gleichzeitig mehrere PGs sowohl lesen als auch schreiben kann. Das sorgt für hohen Durchsatz: Gehören die Binärdaten einer Datei zu Dutzenden verschiedenen PGs, die über mehrere OSDs verteilt sind, so lädt der Client gleichzeitig von mehreren OSDs jeweils ein binäres Objekt herunter. Gleiches gilt für die Schreibfunktion in die umgekehrte Richtung. So lassen sich selbst mit SATA-Standard-Platten hohe Durchsatzraten erzielen.

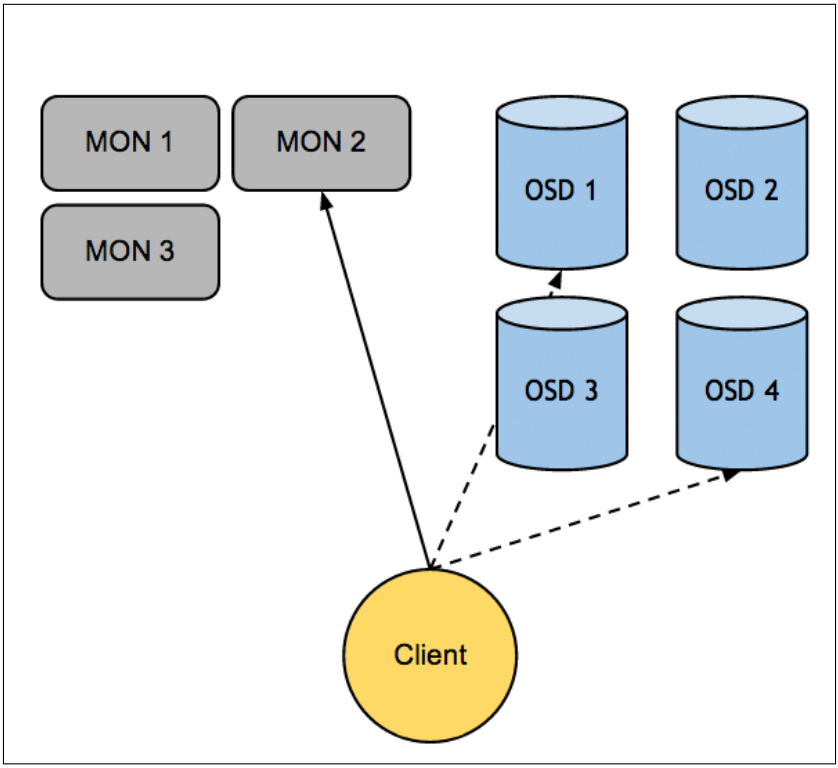


Abbildung 1: Schematische Darstellung von Ceph.

Erwähnung müssen schließlich auch die `Pools` finden: Sie sind im Grunde Namensschilder, die sich einzelnen Objekten anheften lassen und erleichtern so die Administration des Speichers. Obendrein findet die Aufteilung in Placement Groups pro Pool statt und wenn der Admin die Zahl der Replicas verändern möchte, so tut er das ebenfalls auf Basis der Einteilung in Pools.

Sehr angenehm bei Ceph ist die Tatsache, dass die Konfiguration des Object Stores weder kompliziert noch zeitintensiv ist. Anders als andere Lösungen folgt Ceph dem Prinzip *Keep it simple, stupid!* und kommt mit einer Konfigurationsdatei für alle Kernkomponenten des Setups aus. Das heißt freilich nicht, dass es nicht jede Menge Schrauben und Rädchen zum Drehen gäbe – wer aber einen einfachen Einstieg sucht, findet in der folgenden *ceph.conf* eine vollständige Konfigurationsdatei für einen Drei-Knoten-Cluster (Ceph v0.56 alias `bobtail`).

```
[global]
        auth cluster required = cephx
        auth service required = cephx
        auth client required = cephx

[mon]

[mon.daisy]
        host = daisy
        mon addr = 192.168.122.114:6789

[mon.eric]
        host = eric
        mon addr = 192.168.122.115:6789

[mon.frank]
        host = frank
        mon addr = 192.168.122.116:6789

[mds]

[mds.daisy]
        host = daisy

[osd]
        osd journal size = 256 ; journal size, in megabytes

[osd.0]
        host = daisy

[osd.1]
        host = eric

[osd.2]
        host = frank
```

7 Client-Zugriff auf den Ceph Object Store

Ceph kümmert sich wie beschrieben um die sichere Ablage und Replikation der hochgeladenen Daten – unbeantwortet ist bis jetzt allerdings die Frage, welche Möglichkeiten für den Zugriff auf den Speicher verfügbar sind. Hier punktet Ceph mit einer großen Vielfalt, die insbesondere den direkten Konkurrenten OpenStack Swift alt aussehen lässt: Ceph lässt sich mittels eines Dateisystems genauso ansprechen wie als Block Device. Auch eine ReSTful-Schnittstelle steht zur Verfügung, und obendrein geht der Zugriff auch direkt über eine eigene C-API.

8 Das Dateisystem CephFS

Das Dateisystem hört auf den Namen `CephFS` und ist seit Linux 2.6.32 fester Bestandteil des Linux-Kernels. Es lebt direkt im Kernel und umgeht so die Performance-Nachteile, die in FUSE implementierte Lösungen wie GlusterFS haben. CephFS nimmt für sich in Anspruch, ein vollständig POSIX-kompatibles Dateisystem zu sein; wer CephFS nutzt, benötigt deshalb zusätzlich zu den schon beschriebenen `OSDs` und `MONs` auch einen Metadatenserver (`MDS`), der die POSIX-Metadaten an die CephFS-Clients ausliefert. Gespeichert werden die Metadaten übrigens tatsächlich in den Extended User Attributes der einzelnen Objekte, der MDS fungiert lediglich als Server und Cache – anders als beispielsweise bei Lustre führt der Ausfall eines MDS bei Ceph also nicht zu stundenlangen Downtimes.

9 Blockbasierter Zugriff per RADOS Block Device

Ebenfalls zur Riege der Front-Ends für Ceph gehört das *RADOS Block Device*, kurz `RBD`. Auch RBD ist ein Kernel-Treiber, der seit 2.6.37 ebenfalls Bestandteil des Linux-Kernels ist. Per RBD haben Benutzer die Möglichkeit, auf den Object Store zuzugreifen wie auf ein tatsächliches Block Device. Benötigt ein Programm also ein Block-Device als Speicherort für seine eigenen Daten, so lässt sich per RBD dieses Block-Device virtuell im Object Store ablegen. Gut: Anders als bei „normalen“ Block-Devices ist bei RBD die Redundanz inhärent vorhanden. Fällt der ursprüngliche Server mit RBD-Mount aus, lässt sich das gleiche RBD-Device auf einem anderen Host aktivieren und dort nutzen.

10 ReST für Ceph: radosgw

Der Trend geht hin zu Speicherlösungen, die sich per *ReSTful API* ansprechen lassen. Amazons S3 und Dropbox sind gute Beispiele dafür. Tatsächlich bietet der HTTP-basierte Zugriff über ReST Vorteile. Das gilt besonders dann, wenn die angebotene

API mit schon existierenden APIs kompatibel ist und sich so die vorhandenen Clients nutzen lassen. Für Amazons S3 existiert beispielsweise ein Füllhorn voll von bunten und überaus nützlichen Clients, die auch unerfahrene PC-Benutzer nicht vor unlösbare Aufgaben stellen. Grund genug für Ceph, ebenfalls eine ReSTful-kompatible Schnittstelle anzubieten: `radosgw`, das RADOS-Gateway. Dieses bietet ReSTful-APIs an, die sowohl mit Amazons S3-API wie auch mit der OpenStack Swift-API kompatibel sind. In Kombination mit einem Webserver, der FastCGI unterstützt, wird aus dem radosgw eine umfängliche Lösung für Online-Speicherangebote, die vollständig per HTTP-Protokoll nutzbar ist.

11 Zum Selberbauen: librados

Wer mit keiner der vorhandenen Client-Lösungen etwas anfangen kann, freut sich über `librados`: Hierbei handelt es sich um eine vollständige Client-Library für den Zugriff auf Ceph, die in C geschrieben ist und über Bindings für diverse Skriptsprachen verfügt. Über die librados ist es direkt aus einem Programm heraus möglich, per API-Call auf Inhalte des Object Stores zuzugreifen. Qemu ist ein Beispiel für die Nutzung von librados: Ist Qemu mit Unterstützung für librados kompiliert, lassen sich RBD-Images direkt als Festplatten für virtuelle Systeme definieren, so dass der Umweg über den `RBD`-Kernel-Treiber in solchen Setups gar nicht nötig ist.

12 Fazit: Die Zukunft von Storage

Leider reicht der Platz an dieser Stelle für eine ausführlichere Beschreibung von Ceph und seinen Funktionen nicht aus – wer sich für die Lösung interessiert, findet viele nützliche Tipps und Hinweise in der Ceph-Dokumentation. Fest steht jedenfalls: Object Stores und insbesondere Ceph sind im Begriff, den etablierten Storage-Markt ordentlich durcheinander zu wirbeln. Weil keine der zentralen Komponenten in Ceph ein Single Point of Failure ist und von MONs und OSDs letztlich beliebig viele Instanzen vorhanden sein dürfen, bietet Ceph echte Scale-Out-Funktionalität und somit ein Feature, das gerade im Hinblick auf den wachsenden Speicherbedarf des Cloud Computing kaum zu überschätzen ist. Nicht zuletzt spielt freilich auch das liebe Geld eine wichtige Rolle: Ceph-Setups mit Servern von der Stange bieten in aller Regel deutlich mehr (inhärent redundanten) Platz als klassische SAN-Storages und sind dennoch deutlich günstiger – während bei SANs und den typischen Ersatzkonstruktionen viel Geld für schnelle Hardware draufgeht, begnügt sich Ceph mit Standardkomponenten und sticht seine herkömmlichen Kollegen oft trotzdem durch Parallelisierung aus.

Wer sein Storage fit für die Zukunft machen möchte, kommt um Ceph kaum herum.

Literatur

[1] Weil, Sage et al.: *RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters*. University of California, Santa Cruz, 2007
http://ceph.com/papers/weil-rados-pdsw07.pdf

[2] Weil, Sage et al.: *CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data*. University of California, Santa Cruz, 2006
http://ceph.com/papers/weil-crush-sc06.pdf

[3] Loschwitz, Martin: *In die Breite Gegangen*. Admin Magazin 03/2012
http://www.admin-magazin.de/Das-Heft/2012/03/Der-RADOS-Objectstore-und-Ceph-Teil-1

[4] Loschwitz, Martin: *Storage-Magie*. Admin Magazin 05/2012
http://www.admin-magazin.de/Das-Heft/2012/05/Der-RADOS-Objectstore-und-Ceph-Teil-2

[5] Weil, Sage et al.: *The official Ceph documentation*.
http://ceph.com/docs/master/

[6] hastexo: *Hints & Kinks for Ceph*.
http://www.hastexo.com/resources/hints-and-kinks

Sensordaten per SNMP verteilen

Axel Wachtler (axel@uracoli.de) **Ralf Findeisen (rfindeis@gmail.com)**

Im Beitrag wird gezeigt, wie man bekannte Monitoring-Tools (z.B. Nagios oder Icinga) zusammen mit SNMP zur Anbindung eines drahtlosen Sensornetzwerkes verwenden kann. Dazu werden als Konzentrator ein Carambola-Board, das unter OpenWrt läuft und verschiedene 802.15.4-Funkmodule eingesetzt. Die Funkmodule kommunizieren über das einfache P2P-Protokoll aus dem *µracoli*-Projekt. Das Nagios-Monitoring-System ist auf einem Linux-Server installiert.

1 Einleitung

Mit drahtlosen Sensornetzwerken (engl. WSN: wireless sensor network), die auf Grundlage des Standards IEEE 802.15.4 [15] arbeiten, können kleine stromsparende Funkmessknoten aufgebaut werden, die sich vorteilhaft u.a. in der Gebäudeautomatisierung einsetzen lassen. Gerade auch im privaten Bereich ist es hinsichtlich von Energieeinsparung und der Überwachung des Wohnraumklimas nützlich, wenn Messwerte wie Temperatur oder Luftfeuchte sowie der Status der Fensterkontakte dezentral erfasst und zentral abgefragt werden können. Die Steuerung von Aktoren wie Heizung oder Klimaanlage ist aufgrund der bidirektionalen Kommunikation zwar möglich, bei einer selbstgebauten Aktorensteuerung sollte aber grosse Sorgfalt walten.

1.1 System-Überblick

Die Architektur des Sensor-Systems ist in Bild 1 dargestellt. Die Sensorknoten senden ihre Messwerte zu einem Gatewayknoten, der über eine serielle Schnittstelle an den Konzentrator angebunden ist. Der Konzentrator ist ein OpenWrt-Router, auf dem ein SNMP-Dämon läuft, der per WLAN oder Ethernet im Netzwerk erreichbar ist. Auf dem Monitoring-Server läuft Nagios oder ein anderes Monitoring-System, das die Daten der einzelnen Sensorknoten über SNMP-Anfragen ausliest oder Kommandos an die Sensorknoten sendet.

1.2 Anforderungen

Ein Messwert-Monitoring-System soll robust funktionieren, deshalb wurde auf langjährig erprobte Komponenten wie Net-SNMP und Nagios gesetzt. Die Verwendung von SNMP [16] macht das System generisch nutz- und erweiterbar. SNMP ist schmalbandig und verwendet das UDP-Protokoll. Einfache Implementierungen

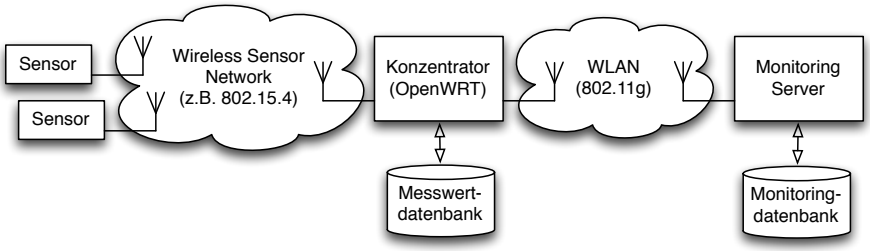


Bild 1: Architekturüberblick

von SNMP-V1 sind bereits für 8-Bit-Mikrocontroller möglich [1]. Durch die Verwendung eines OpenWrt-Routers steht allerdings mehr Rechenleistung und Speicher zur Verfügung, so dass die sichere Protokollversion SNMP-V3 verwendet werden kann.

Im Beitrag wird zu Demonstrationszwecken das sehr einfache aber kryptographisch ungesicherte *µracoli*-P2P-Protokoll verwendet. Dies stellt eine erhebliche Sicherheitsschwachstelle dar, da die ausgetauschten Nachrichten im WSN nicht authentisiert oder verschlüsselt sind. Die vorgestellte Methode kann jedoch auf jedem beliebigen WSN-Protokoll-Stack übertragen werden, da der WSN-Dämon die Daten von der seriellen Schnittstelle des WSN-Gateway-Knotens parst und weiterverarbeitet.

2 Das Sensornetzwerk und die Sensoren

Drahtlose Sensor Netzwerke verwenden je nach eingesetztem Protokollstack unterschiedliche Topologien. Im einfachsten Fall handelt es sich um eine Punkt-zu-Punkt-Verbindung oder um ein Sternnetz. In dieser Konfiguration übermitteln die Sensorknoten ihre Messwerte direkt zum Konzentrator. Beispiele für diese Netzwerktopologie sind der IEEE-802.15.4-MAC-Layer oder das *µracoli*-P2P-Protokoll.

Da die Sensorknoten vorrangig stromsparend aufgebaut sind, ist auch die abgestrahlte Sendeleistung meist nur gering. Beim Einsatz von 2.4 GHz Funktransceivern in Gebäuden ergeben sich zusätzliche Probleme durch die geringe Wellenlänge in Form von Interferenzen und Abschattungen. Zusammen mit anderen Funk-Technologien wie WLAN oder Bluetooth, die um das gleiche Frequenzband konkurrieren, ergibt sich eine eingeschränkte Funkreichweite, und der Gateway-Knoten ist unter Umständen nicht mehr direkt erreichbar. Ist das der Fall, so müssen die Informationen der Sensoren durch Router-Knoten zum Gateway-Knoten weitergeleitet werden. Beispiele für Routing-Protokolle sind ZigBee, IETF 6LowPan, Leightweight-Mesh (Atmel) oder Route Under MAC (Atmel).

Als Sensorknoten finden einige TicTacToe-Platinen aus dem Workshop vom vergangenen Jahr Verwendung [2]. Der darauf befindliche Atmega128RFA1 vereint Micro-

controller, Funkinterface und Temperatursensor in einem Bauteil. Die Firmware des Sensorknotens misst periodisch Temperatur und Spannung und sendet die Daten an den Konzentrator. Als Sendeprotokoll kommt das einfache P2P-WUART-Protokoll zum Einsatz, das in der Payload des Datenrahmens einen Klartextstring überträgt.

Am Konzentrator-Knoten ist das WSN-Gateway über eine serielle Schnittstelle angeschlossen. Auf dem Gateway läuft die *µracoli*-Wireless-UART-Firmware, die die Payload aller empfangenen Rahmen auf der seriellen Schnittstelle ausgibt. Stellt man die Ausgabe des Gateways mit einem Terminal-Programm dar, so erhält man folgende Textzeilen:

```
ADDR=0004, T=18.2, Vcc=2.93
ADDR=0001, T=-3.3, Vcc=2.99
ADDR=0003, T=26.7, Vcc=3.25
...
```

Diese einfache Methode zur Übertragung der Messwerte ist gegenüber einem angepassten Binärprotokoll vielleicht nicht besonders effizient, erlaubt aber die einfache Erweiterung um zusätzliche Messwerte, die vom Sensorknoten erfasst werden können.

Das P2P-Protokoll dient primär zu Demonstrations-, Lern- und Testzwecken. Da es keine kryptographische Sicherung der Messwerte implementiert, sollte es zur Übertragung von kritischen Informationen in einem Produktivsystem nicht verwendet werden. Die Firmware ist im Detail in [13] beschrieben.

3 Der Konzentrator

Der Konzentrator besteht aus einem OpenWrt-Router und dem Gateway-Knoten. Im Beispielprojekt wurde als Routerhardware das Carambola-Board der Firma 8devices verwendet [4]. OpenWrt [8] ist eine Linuxdistribution, die speziell für die Verwendung auf embedded Geräten angepasst ist. OpenWrt umfasst eine Vielzahl an Software-Paketen, die mit dem Paketmanager `opkg` installiert werden können.

3.1 OpenWrt-Software

Wie man die Software-Umgebung zur Erstellung eines Carambola-Images installiert und nutzt, ist in [4] beschrieben. Weitere Erfahrungen zur Inbetriebnahme des Boards sind unter http://uracoli.nongnu.org/carambola.html zu finden.

Wenn die Entwicklungsumgebung läuft, können mit dem Befehl `make menuconfig` die zusätzlich benötigten Software-Komponenten ausgewählt werden.

```
# serielle Treiber für den Gateway-Knoten
kmod-usb-acm, kmod-usb-serial, kmod-usb-serial-ftdi

# Libraries
libsqlite3, librs232, libncurses, libreadline, liblua

# Lua-Module
luaposix, luars232, luasql-sqlite3, lsqlite3

# Tools
lua, sqlite3-cli
```

Im Carambola-Repository sind nicht alle erforderlichen Komponenten enthalten. Durch das leistungsfähige Build-System ist die Erweiterung der fehlenden Komponenten jedoch einfach möglich. Um ein fehlendes Paket zu ergänzen, wird einfach das entsprechende Verzeichnis unter `packages/` anlegt und aus dem OpenWrt-Repository werden das `Makefile` sowie die lokalen Patches und Zusatzfiles kopiert. Danach erscheint automatisch das neue Paket als Eintrag im Konfigurationsmenü.

3.2 Prozessmodell

Zum Schreiben und Lesen der Sensordaten sind zwei Prozesse erforderlich. Auf der WSN-Seite werden die Daten von der seriellen Schnittstelle mit dem WSN-Dämon gelesen. Auf der Netzwerkseite läuft der SNMP-Dämon (siehe Bild 2) und liest die Daten vom WSN-Dämon. Da beide Prozesse vollständig asynchron arbeiten, muss eine Möglichkeit der kollisionsfreien Interprozesskommunikation gefunden werden. Lösungsansätze über einen, mit `flock` abgesicherten File-Zugriff oder der Einsatz von Sockets wurden aus Gründen der Robustheit, des Ressourcenbedarfs und der aufwändigeren Fehlersuche verworfen. Das Problem des kollisionsfreien Zugriffes für mehrere Prozesse wurde letztlich mit einer SQLite3-Datenbank [11] gelöst.

Der WSN-Dämon schreibt die Sensordaten mit UPDATE-Befehlen in die Datenbank. Wenn der SNMP-Dämon ein SNMP-Get-Paket empfängt, liest er die entsprechenden Datensätze mit einem SELECT-Befehl von der Datenbank und leitet sie an den SNMP-Manager weiter.

Da jeder Sensorknoten unterschiedliche physikalische Messwerte liefern kann, wird in der Datenbank je eine Tabelle pro Messwert angelegt.

```
Table tab_LOC (addr, ort)
Table tab_T   (addr, uhrzeit, wert)
Table tab_Vcc (addr, uhrzeit, wert)
...
```

Die Tabelle `tab_LOC` muss manuell gewartet werden, sie beinhaltet die Zuordnung der Adressen der Sensorknoten zu deren Installationsorten. Die Wertetabellen `tab_T` und `tab_Vcc` speichern die Adresse, den Messwert und die Uhrzeit des letzten empfangenen Datensatzes. Anhand des Zeitstempels kann so ein eventueller Ausfall des Knotens detektiert werden.

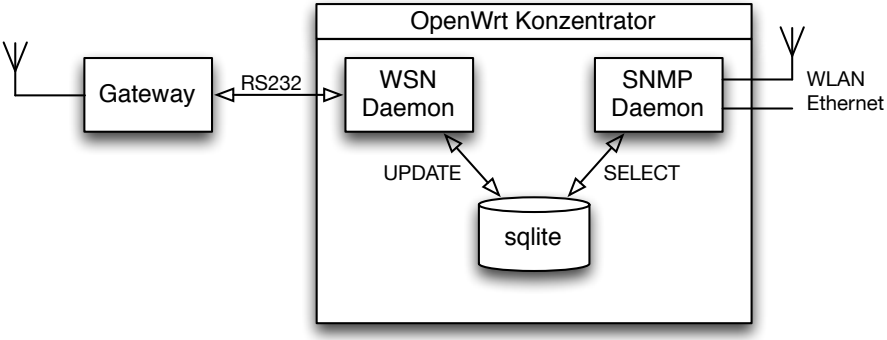


Bild 2: Datenmodell

3.3 WSN-Dämon

Der WSN-Dämon wurde als Lua-Script implementiert. Es liest kontinuierlich die Daten des Gateways von der seriellen Schnittstelle und schreibt sie in die Datenbank. Mit dem Tool `start-stop-daemon` wird das Lua-Script als Prozess auf Systemebene gestartet, angehalten und der aktuelle Status überprüft.

Der WSN-Dämon kann prinzipiell in jeder Programmiersprache geschrieben werden. Da die Speicherressourcen des Carambola-Boards jedoch begrenzt sind, fiel die Wahl auf die Script-Sprache Lua [12].

3.4 SNMP-Dämon

Der SNMP-Dämon ist ein Server-Dienst, der SNMP-Anfragen benatwortet. Die Installation erfolgt mit dem Befehl `opkg install snmpd`. Die Adressierung der Daten erfolgt über die Managment Information Base (MIB). Diese beschreibt die Struktur der Informationen und ist ein hierarchischer Baum von OIDs (Object Indentifier). Jeder OID beschreibt den Inhalt einer Variablen, die les- und/oder schreibbar ist. Für unsere Anwendung liefert die ENTITY-SENSOR-MIB nach RFC3433 [3] passenden Attribute:

```
Objekt-Name             OID
entitySensorMIB         1.3.6.1.2.1.99
entPhySensorTable       1.3.6.1.2.1.99.1.1
entPhySensorEntry       1.3.6.1.2.1.99.1.1.1
...
```

OpenWrt-Systeme werden über das Unified Configuration Interface (UCI) konfiguriert. Für den SNMP-Dämon wird die Datei `/etc/config/snmpd` verwendet. Das Starten des Prozesses erfolgt mit dem Befehl `/etc/init.d/snmpd start`. Dabei wird die eigentliche Konfigurationsdatei `snmpd.conf` erzeugt.

```
Ausschnitt /etc/config/snmpd
# Einbindung der Sensor Table
config pass
    option name MIB-SENSOR
    option prog /bin/wsn_snmpd_pass.sh
    option miboid .1.3.6.1.2.1.99
```

Im Script `/bin/wsn_snmpd_pass.sh` wird der entsprechende ENTITY-SENSOR-MIB Abschnitt implementiert. Das MIB-Attribut wird dem Script als Kommandozeilenparameter übergeben. Als Rückgabewerte liefert es die OID, den Datentyp und den eigentlichen Messwert. Nachdem der Dämon fertig konfiguriert und neu gestartet ist, können mit `snmpwalk` oder `snmpget` die entsprechenden MIB-Attriute über das Netzwerk abgefragt werden. Anzumerken sei noch, dass SNMP keinen Float-Datentyp unterstützt, d.h. man muss sich entweder mit einer Festpunktkonvertierung behelfen oder man verwendet den Datentyp String.

4 Management und Visualisierung auf dem Server

In Wikipedia sind derzeit über fünfzig Netzwerk-Monitoring-Programme aufgezählt [14]. Nagios wurde hier ausgewählt, da es eines der bekantesten Tools in diesem Bereich ist. Das sehr mächtige Programm ist vor allem unter Systemadministratoren zur Überwachung großer IT-Landschaften sehr beliebt. Auf die zu überwachenden Parameter kann mit unterschiedlichen Methoden zugegriffen werden, SNMP ist nur eine von vielen Möglichkeiten.

Nagios setzt die Installation eines Apache-Webservers voraus. Die erforderlichen Pakete werden über den Paketmanager der jeweiligen Distribution installiert. Nach der Installation sind jedoch noch weitere Schritte zur Inbetriebnahme eines funktionierenden Basis-Systems erforderlich [7].

Die Einbindung der Sensor-Messwerte erfolgt durch die Definition von Diensten. Im File `/etc/nagios3/conf.d/konzentrator_nagios2.cfg` wird ein Temperatur-Messwert wie folgt definiert:

```
define service{
    use                     generic-service
    host_name               konzentrator
    service_description     aussen_1
    check_command           check_snmp!1.3.6.1.2.1.99.1.1.1.2
}
```

Da für jeden Sensor und jeden Messwert ein Service definiert werden muss, empfiehlt es sich, dafür ein Generator-Script zu verwenden, das die entsprechenden Definitionsabschnitte anhand der SQlite-Datenbank auf dem Konzentrator-Knoten erzeugt.

Die Funktionalität von Nagios kann durch zahlreiche Plugins erweitert werden. Der zeitliche Verlauf von Messwerten kann u.a. mit dem Plugin PNP4Nagios [9] dargestellt werden.

5 Ausblick

Das in den vorangangenen Abschnitten gezeigte System ist erst ein Anfang. Die Weiterentwicklung dieser Sensorinfrastruktur hat vielfältige Optionen zur Erweiterung und Vervollkommnung.

Denkbar ist beispielsweise die Unterstützung von Mobilgeräten, die Verwendung geeigneter Interfaces zur Verwaltungssoftware oder das Schreiben eines leichtgewichtigen web-basierenden Clients, welcher dann als Applikation auf dem Mobilgerät die Messwerte darstellt und Aktionen anbieten kann. Die heute verfügbaren leistungsfähigen Bibliotheken wie D3 [5] erlauben eine ansprechende Darstellung und daher die Einbindung in publikumssichtbare Systeme, deren Darstellungsanforderungen über pure Nützlichkeit hinausgehen.

Das gezeigte Sensornetzwerk ist auch nicht die einzige Möglichkeit der Messwerterhebung und Übermittlung. Neben verwandten Systemen, wie beispielsweise autarken Sensoren von Seeedstudio, die auch die Verwendung anderer Radiointerfaces erlauben würden [10], kann man auch in Erwägung ziehen, den Nagios-Server um Skripte für die Ansteuerung eines Heizungssteuersystems, wie z.B. MAX! Cube [6] zu erweitern, um so nicht nur die Heizung anhand von Profilen und als Reaktion auf offene Fenster zu steuern, sondern auch auf Ausseneinflüsse einzugehen. Natürlich sind dabei die weiter oben erwähnten Sicherheitsfragen, speziell beim Einsatz von Aktoren zu beachten.

Insgesamt ist die Messung und Automatisierung mit Hilfe günstiger und leicht zu verwendender Sensoren und Überwachungsrechner ein Gebiet mit unserer Meinung nach grossem Potential und wir möchten die Community einladen, mit uns in der Entwicklung fortzuschreiten.

Die Webseite zum Vortrag befindet sich unter http://uracoli.nongnu.org/clt2013.

6 Literatur

[1] *Agentuino - A lightweight SNMP Agent for Arduino Platforms*. URL: http://bit.ly/Wo0QmA.

[2] A. Wachtler et. al. "Tic-Tac-Toe Reloaded". In: *Chemnitzer Linuxtage 2012*. 2012. URL: http://uracoli.nongnu.org/clt2012.

[3] A. Bierman, D. Romascanu und K.C. Norseth. *Entity Sensor Management Information Base*. RFC 3433 (Proposed Standard). Internet Engineering Task Force, 2002. URL: http://www.ietf.org/rfc/rfc3433.txt.

[4] *Carambola - a flexible 802.11n module*. URL: http://www.8devices.com/product/3/carambola.

[5] *D3.js JavaScript Bibliothek*. URL: http://d3js.org.

[6] *ELV Max! Protocol Description*. URL: http://bit.ly/W9kQsf.

[7] *Nagios: Fixing 'error: Could not stat() command file' (on Debian)*. URL: http://bit.ly/vjvkcG.

[8] *OpenWrt*. URL: http://www.openwrt.org.

[9] *PNP4Nagios*. URL: www.pnp4nagios.org.

[10] *Seeedstudio Stalker Waterproof Solar Kit*. URL: http://bit.ly/UVKHDS.

[11] *SQLite Home Page*. URL: http://www.sqlite.org.

[12] *The Programming Language Lua*. URL: http://www.lua.org.

[13] Daniel Thiele und Axel Wachtler. "Funkübertragung mit IEEE 802.15.4 Transceivern - nicht nur für Profis". In: *Embedded Projects Journal* 14 (2012), 22 ff. URL: http://bit.ly/Sf1nJk.

[14] Wikipedia. *Comparison of network monitoring systems*. 2013. URL: http://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems.

[15] Wikipedia. *IEEE 802.15.4*. 2013. URL: http://de.wikipedia.org/wiki/802.15.4.

[16] Wikipedia. *Simple Network Management Protocol*. 2013. URL: http://de.wikipedia.org/wiki/SNMP.

Vertraulich kommunizieren mit OTR

Jens Kubieziel
jens@kubieziel.de
http://www.kubieziel.de/

Wenn wir uns in der Offline-Welt treffen und uns unterhalten, so gehen wir stillschweigend davon aus, dass die Gespräche unter uns bleiben. Bei der elektronischen Kommuninikation werden die Daten an verschiedener Stelle gespeichert und sind lange Zeit abrufbar. Mit Off-the-record Messaging soll dieses Dilemma behoben werden.

1 Einleitung

Alice, Bob und Charlie[1] treffen sich wie jeden Freitag in ihrer Stammkneipe und diskutieren die neuen Entwicklungen in der Welt von GNU/Linux. Änderungen am Kernel, der beste Editor von allen und Diskussionen zur Distribution stehen immer wieder auf der Tagesordnung. Dabei kommt es schon vor, dass die beste Distribution mal Ubuntu, mal openSUSE und dann wieder centOS ist. Niemand hält sich untereinander den Meinungswechsel vor und zum Teil vergessen die drei, wer wann was bevorzugt hat.

Wechseln wir nun in die Online-Welt. Es ist egal, ob das obige Treffen im per IRC, E-Mail oder Videochat veranstaltet wird. Viele Protokolle sind Jahrzehnte nach deren Erfindung immer noch ungeschützt. Das heißt, Eve und Mallory können mitlesen oder sogar Nachrichten ändern, ohne das dies jemand bemerkt.

Auf diese Weise wird noch Jahre später genau nachvollziehbar, wer wann welche Distribution oder welchen Editor bevorzugte. Oder eben welche politischen Einstellungen vor Jahren vertreten wurden.

1.1 Eigenschaften von OTR

Off-the-Record messaging (OTR) versucht, das oben beschriebene Problem zu lösen. Nachrichten, werden verschlüsselt zu einer Gegenstelle übertragen. Diese muss sich vor Beginn der Kommunikation authentifizieren. Jedoch besitzen die übertragenen Nachrichten keine Unterschrift. Daher kann im nachhinein nicht bewiesen werden, wer welche Nachricht versandt hat. Schließlich bereitet der Verlust des geheimen Schlüssels keine Probleme. Ein Angreifer kann mit diesem Wissen keine alten Nachrichten entschlüsseln. Insgesamt versucht OTR damit ein Gespräch unter vier Augen, wie wir es aus der Offline-Welt kennen, nachzubauen.

1.2 Wichtige Begriffe

Im folgenden sollen einige wichtige Begriffe kurz vorgestellt werden:

Definition 1.1 (Verschlüsselung) *Nachrichten oder allgemein Daten werden als verschlüsselt bezeichnet, wenn kein anderer diese lesen kann.*

Formal betrachtet, ist die Verschlüsselung eine Funktion, die einen Klartext und einen Schlüssel als Eingabe erhält und daraus einen Geheimtext erzeugt.

Definition 1.2 (Authentifizierung) *Bei einer Authentifizierung verfiziert der Server bestimmte Eigenschaften, die meist ein Benutzer behauptet hat.*

Üblicherweise authentisiert sich ein Benutzer bei einem Server, indem er einen Nutzername und das dazugehörige Passwort eingibt. Der Server prüft diese Angaben und authentifiziert damit den Benutzer. Beide Begriffe werden üblicherweise nicht stark unterschieden.

Definition 1.3 (Bestreitbarkeit) *Eine Person kann eine Aktion im Nachhinein glaubhaft verneinen.*

Im Rahmen der IT-Sicherheit ist die *Verbindlichkeit* bzw. *Nichtabstreitbarkeit* (aus dem Englischen: non-repudiation) das Thema. Also es wird versucht, dass Aktionen eben nicht betritten werden können. Hierzu werden meist vertrauenswürdige Dritte (trusted-third parties) oder Public-Key-Infrastrukturen verwendet. Im Rahmen von OTR wird nun der Versuch unternommen, Bestreitbarkeit herzustellen.

Definition 1.4 (Perfect Forward Secrecy (PFS)) *Perfect Forward Secrecy wird im Deutschen als* Folgenlosigkeit *bezeichnet und gewährleistet, dass niemand von einem aufgedeckten kryptografischen Schlüssel auf folgende oder vorhergehende Schlüssel schließen kann.*

Um PFS zu gewährleisten, müssen die Schlüssel unabhängig voneinander erzeugt werden. Dies ist beispielsweise ein Problem bei kleinen Routern, wo wenig Zufallsquellen bereitstehen. Die Datei `/proc/sys/kernel/random/entropy_avail` zeigt bei den Geräten direkt nach dem Boot oft sehr geringe Werte. Selbst Werte von 0 sind zu beobachten. Dies sorgt dann dafür, dass diese Rechner beispielsweise immer den gleichen SSH-Schlüssel erzeugen. In dem Fall wäre PFS nicht gewährleistet.

2 Funktionsweise von OTR

OTR kombiniert einige kryptografische Algorithmen, um die Ziele Verschlüsselung, Authentifizierung, Bestreitbarkeit und PFS zu erreichen.

2.1 Diffie-Hellman-Schlüsselaustausch

Der Diffie-Hellman-Schlüsselaustausch[2] dient der Aushandlung eines gemeinsamen geheimen Schlüssels zwischen zwei Parteien. Für die genauen Details wird ein wenig Mathematik, genauer Gruppentheorie benötigt. Unten soll der Ablauf anhand eines Beispiels kurz dargestellt werden.

1. Zu Beginn einigen sich Alice und Bob auf zwei Zahlen, eine Primzahl p und der Erzeuger einer Gruppe g.
2. Alice und Bob wählen zwei zufällige Zahlen zwischen 1 und $p-2$. Wir bezeichnen diese Zahlen als a und b.
3. Alice berechnet $k_a = g^a \mod p$ und Bob $k_b = g^b \mod p$. Beide Werte werden jeweils an den Gegenüber übertragen.
4. Alice berechnet nun $k = (k_b)^a = g^{ab} \mod p$ und Bob $k = (k_a)^b = g^{ab} \mod p$.

Die Zahlen p, g, k_a und k_b werden über einen öffentlichen Kanal übertragen. Theoretisch könnte ein Angreifer diese Zahlen mitlesen und versuchen, k zu berechnen. Dies wird als schweres Problem (Diskreter Logarithmus) angesehen. Bisher sind keine effizienten Algorithmen zur Lösung bekannt.

OTR verwendet die in RFC 3526 definierten Werte[3] für p und g.

2.2 Digital Signature Algorithm (DSA)

Der DSA-Standard stammt aus dem Jahr 1991 und dient der digitalen Unterschrift elektronischer Dokumente. Er wurde von der amerikanischen Standardisierungsbehörde NIST (National Institute of Standards and Technology) für den Digital Signature Standard (DSS) empfohlen.[4]

2.3 Advanced Encryption Standard (AES)

AES entstammt ebenfalls den Standardisierungsbemühungen der NIST. Der bis Ende 1990er Jahre empfohlene Algorithmus DES zeigte deutliche Schwächen. Daher wurde ein internationaler Wettbewerb ausgerufen. Infolgedessen wurde Rijndael[5] als neuer Standard festgelegt.

Der AES-Algorithmus verarbeitet Blöcke von jeweils 128, 192 oder 256 Bit Größe. Die Schlüssellänge entspricht ebenfalls 128, 192 oder 256 Bit. Der Eingabeblock wird byteweise in eine Tabelle mit vier Zeilen übertragen. In mehreren Durchläufen erfolgt eine Substitution, eine Verschiebung und Vermischung der Spalten sowie eine Verknüpfung mit einem pro Durchlauf neu errechneten Schlüssel.

Die Verschlüsselung von Daten mit AES gilt derzeit als sicher. Nach dem derzeitigen Stand der Forschung[6] kann die Verschlüsselung mit einer Komplexität von ca. 2^{100} angegriffen werden. Dieser Wert liegt außerhalb praktisch relevanter Werte. Jedoch ist zu erwarten, dass in der Zukunft deutlich bessere Angriffe gefunden werden. Daher besteht diese Sicherheit nicht für alle Zeit.

2.4 Secure Hash Algorithm (SHA)

Der Secure Hash Algorithm ist eine Familie so genannter kryptografischer Hashfunktionen.

Definition 2.1 (Hashfunktion) *Eine Hashfunktion bildet Eingaben beliebiger Länge auf eine Zeichenfolge fester Länge ab.*

Eine kryptografische Hashfunktion muss daneben noch kollisionsresistent sein. Das bedeutet, dass es sehr schwer sein muss, zwei verschiedene Eingaben zu finden, die die selbe Ausgabe erzeugen.

So könnte beispielsweise die Modulo-Operation, also die Division mit Rest, eine Hashfunktion sein. Stellen wir uns vor, wir rechnen modulo 10. Also $16 \mod 10 = 1 \cdot 10 + 6 \equiv 6$ oder $1973 \mod 10 = 197 \cdot 10 + 3 \equiv 3$. Was wäre $100 \mod 10$? Finden Sie eine Kollision, also zwei verschiedene Werte, die dieselbe Ausgabe erzeugen?

Derzeit existieren einige Hasfunktionen. Sehr bekannt sind MD5 und SHA-1. Letztere stammt aus der SHA-Familie. Beide Algorithmen gelten heute als stark geschwächt bzw. erfolgreich angegriffen und sollten durch sichere Alternativen ersetzt werden. Die sind SHA-2 bzw. der erst 2012 neu standardisierte Algorithmus SHA-3.

2.5 Keyed-Hash Message Authentication Code (HMAC)

Alice könnte aufgrund der oben vorgestellten Hashfunktion auf die Idee kommen, übertragene Dateien zu hashen und den Hashwert zu übertragen. Damit kann Bob feststellen, dass er wirklich die gleiche Datei wie Alice erhielt. Doch was ist, wenn Mallory diese Datei austauscht, einen Hashwert erzeugt und das gemeinsam an Bob sendet? In dem Fall hat Bob keine Prüfungsmöglichkeit, ob die Datei von Alice kam. Message Authentication Codes (MAC) versuchen, dieses Szenario zu lösen.

HMACs sind spezielle MACs, die auf kryptografischen Hashfunktionen basieren. Für eine genaue Beschreibung sei auf RFC 2104[7] verwiesen.

2.6 Das OTR-Protokoll

OTR verwendet die oben kurz vorgestellten kryptografischen Algorithmen, um die gesetzten Ziele zu erreichen.

Zu Anfang erzeugen Alice und Bob langlebige Schlüssel nach dem DSA-Verfahren. Dieser Schlüssel wird durch einen so genannten Fingerprint (Fingerabdruck) repräsentiert. Dieser kann beispielsweise wie folgt aussehen:

```
jens@anonymitaet-im-inter.net:
                    0D35970A 35841C31 B20551A4 83744AB9 CA189C14
```

Möchten Alice und Bob eine Kommunikation aufnehmen, so steht am Anfang die gegenseitige Authentifizierung. Dazu übertragen beide Seiten jeweils den Fingerprint über einen sicheren Kanal. Stimmt der angezeigte Wert mit dem übertragenen überein, so können sich beide Seiten sicher sein, den richtigen Kommunikationspartner vor sich zu haben.

Bei aktueller OTR-fähiger Software kann auch ein gemeinsames Geheimnis zur Authentifizierung dienen. Das heißt, Alice stellt Bob eine Frage, deren Antwort nur Bob kennt (»Wie ist der Titel des Buches in der ersten Reihe ganz links?«). Antwortet Bob wie von Alice vorgegeben, so gilt er als authentifiziert.

Diese gegenseitige Authentifizierung erfolgt nur einmalig. Später greifen die Programme auf die gespeicherten Angaben zurück. Die oben beschriebene Authentifizierung erfolgt innerhalb einer Diffie-Hellman-Sitzung. Das heißt, bevor sich beide Seiten »kennen«, handeln sie einen gemeinsamen Schlüssel zu geheimen Kommunikation aus. Dieser Schlüssel wird nach erfolgreicher Etablierung sofort verworfen. Stattdessen wird, basierend auf den gewonnenen Erkenntnissen, eine neuer Schlüssel verhandelt.

Dieser Schlüssel zusammen mit dem zuletzt von Bob übertragenen Schlüssel bildet die Grundlage für eine neue Nachricht. Angenommen Alice übertrug zuletzt g^{a_i} und Bob g^{b_i}. Dann können beide Seiten $g = (g^{a_i})^{b_i} = (g^{b_i})^{a_i}$ berechnen. Die ersten 120 Bit des SHA-1-Hashes von g bilden den Schlüssel für AES. Der Wert wird nochmal gehasht und ist der Schlüssel für den HMAC. Schließlich verschlüsselt Alice ihre Nachricht und sendet diese zusammen mit dem HMAC-Wert sowie anderen Angaben an Bob.

Er kann im selben Verfahren wie Alice g sowie die AES- und HMAC-Schlüssel berechnen. Dies ermöglicht ihm die Prüfung der Authentizität sowie die Entschlüsselung der Nachricht.

Das Protokoll wurde oben in einer verkürzten Variante dargestellt. Der genaue Ablauf mit allen Parametern findet sich in der aktuellen Protokollbeschreibung[8].

3 Benutzung von OTR

3.1 Verfügbare Software

OTR ist von vornherein an keine bestimmte Software gebunden. Verschiedene Chatprogramme haben Unterstützung für das Protokoll eingebaut. Empathy ist einer der Programme ohne OTR. Trotz entsprechender Wünsche der Nutzer[1] haben sich die Entwickler gegen OTR entschieden.[2] Folgende (unvollständige) Liste von Programmen funktioniert mit der Verschlüsselung:

- Adium
- Kopete
- Pidgin
- Psi
- XChat
- irssi
- mcabber
- weechat
- IM+ und Gibberbot für Android

3.2 OTR mit Pidgin

Pidgin ist ein Chatprogramm, welches weite Verbreitung innerhalb der verschiedenen Linux-Distributionen gefunden hat. Im folgenden werden exemplarisch die Schritte zur Kommunikation mit OTR erklärt.

3.2.1 OTR-Unterstützung für Pidgin

Standardmäßig kommt Pidgin ohne Unterstützung für OTR daher. Dies muss zunächst installiert und später in der Software aktiviert werden. In der Paketverwaltung der meisten Distributionen ist die Software unter dem Namen »pidgin-otr« zu finden.

Nach der Installation hat die Liste der verfügbaren Plugins (Abbildung 1) den neuen Eintrag »Off-the-Record Messaging«. Wenn dieser aktiviert wird, kann das Plugin anschließend konfiguriert werden.

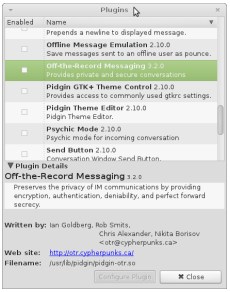


Abbildung 1: Auflistung der Plugins von Pidgin

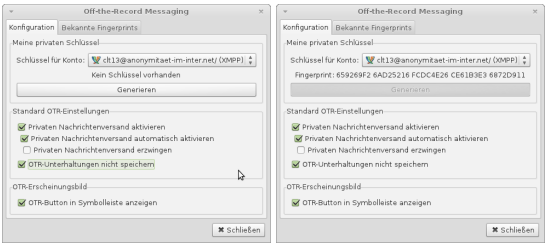


Abbildung 2: Erzeugung eines langlebigen Schlüssels

3.2.2 Schlüssel erzeugen

Der wichtigste Konfigurationseintrag dient der Erzeugung eines neuen Schlüssels. In der Abbildung 2 findet sich links die Ansicht direkt nach der Installation. Ein Klick auf »Generieren« erzeugt den in Unterabschnitt 2.6 beschriebenen langlebigen Schlüssel. Der Fingerprint (rechtes Bild in Abbildung 2) identifiziert den Schlüssel eindeutig.

Weiterhin empfiehlt es sich, den Eintrag »OTR-Unterhalten nicht speichern« zu aktivieren. Denn im Hinblick auf die Schutzziele von OTR kann sich eine lokale, unverschlüsselte Speicherung der Kommunikationsinhalte negativ auswirken. Der vermeintliche Whistleblower Bardley Manning wurde u. a. durch einen Fehler in Adium[3] überführt. Adium speicherte seine Kommunikation im Klartext.

Nun ist OTR innerhalb von Pidgin aktiv. Im Chatmenü findet sich rechts oben der Eintrag »OTR«.

[1] https://bugs.launchpad.net/ubuntu/+source/libtelepathy/+bug/296867
[2] https://live.gnome.org/Empathy/FAQ#Will_Empathy_have_OTR_.28.22Off_The_Record.22.29_support.3F
[3] http://trac.adium.im/ticket/15722

3.2.3 Vertrauliches Gespräch beginnen

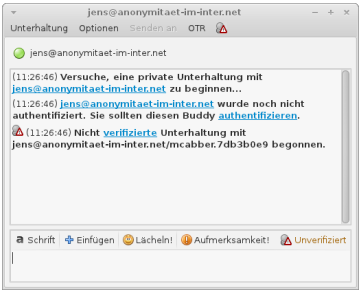


Abbildung 3: Unauthorisierte Kommunikation

Pidgin versucht nun standardmäßig mit den Kommunikationspartnern per OTR in Kontakt zu treten. In der Abbildung 3 wurde die Jabber-ID jens@anonymitaet-im-inter.net angesprochen. Dieser Account nutzt ebenfalls OTR. Allerdings wurde der von clt13@anonymitaet-im-inter.net noch nicht authentifiziert. Daher druckt Pidgin die Warnung, dass die Unterhaltung nicht verifiziert ist. Zu Anfang empfiehlt es sich, den Gegenüber zu verifizieren. Die Menüpunkte »OTR« rechts oben sowie »Unverifiziert« rechts unten bieten dies an.

Abbildung 4: Verifikation des Gegenübers

Der Screenshot zeigt die Authentifizierung mittels einer Frage und der dazugehörigen Antwort. Mit dem Menü lässt sich ebenfalls der Fingerprint prüfen. Ein Klick auf »Authentifizieren« bestätigt den Schlüssel. Fortan wird diesem vertraut. Bei jedem Chat mit dieser Person stellt Pidgin eine OTR-Verbindung her. Es bedarf hierfür keines Eingriffes seitens des Benutzers.

Der letztgenannte Punkt macht OTR für den Endanwender interessant. Denn bis auf die erste Authentifizierung sind keine Interaktionen notwendig. Dennoch können die Benutzer sicher kommunizieren. Insgesamt integriert sich OTR sehr gut in alle Chatprogramme. Es funktioniert mit allen Protokollen, wie ICQ, XMPP und anderen. Damit ist es ein wichtiger Baustein in der sicheren Kommunikation und sollte von jedem eingesetzt werden.

Literatur

[1] Ron L. Rivest, Adi Shamir, Leonard Adleman: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978.

[2] Whitfield Diffie, Martin Hellman: New Directions in Cryptography. IEEE Transactions on Information Theory. 22, Nr. 6, 1976, S. 644–654. http://www-ee.stanford.edu/~hellman/publications/24.pdf

[3] T. Kivinen, M. Kojo: RFC 3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). 2003. https://tools.ietf.org/html/rfc3526

[4] NIST: FIPS PUB 186-3: Digital Signature Standard (DSS). 2009. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[5] Joan Daemen, Vincent Rijmen: The Design of Rijndael. AES: The Advanced Encryption Standard. Springer. Berlin u. a. 2002.

[6] Alex Biryukov Dmitry Khovratovich: Related-key Cryptanalysis of the Full AES-192 and AES-256. Cryptology ePrint Archive, Report 2009/317. 2009. http://eprint.iacr.org/2009/317

[7] H. Krawczyk, M. Bellare, R. Canetti: RFC 2104: HMAC: Keyed-Hashing for Message Authentication. 1997. http://tools.ietf.org/html/rfc2104

[8] Ian Goldberg: Off-the-Record Messaging Protocol version 3. http://www.cypherpunks.ca/otr/Protocol-v3-4.0.0.html

WeLoupe Next Generation Web-Scanner

Philipp Seidel
phibo@dinotools.org
http://www.dinotools.org

Ein Web-Scanner stellt Funktionen zur Analyse von Web-Anwendungen, Web-Servern und Web-Diensten zur Verfügung. Die dabei gewonnen Informationen helfen Administratoren, Sicherheitsexperten und Wissenschaftlern verwundbare Dienste aufzuspüren und abzusichern. Gleichzeitig erlaubt er festzustellen, welche Informationen über die eingesetzte Software und ihre Version ein Angreifer leicht in Erfahrung bringen kann.

In diesem Beitrag wird das WeLoupe-Framework vorgestellt. Mit seiner Hilfe können Web-Seiten abgerufen, eingebettete Scripte ausgeführt, die empfangenen Daten analysiert und anschließend ein Report erstellt werden. Das Hauptaugenmerk liegt dabei darauf, eine einfache und flexible Möglichkeit zur Erweiterung des Frameworks und für das Testen neuer Techniken bereitzustellen.

1 Einführung und Motivation

In den Anfangstagen des World Wide Web gab es nur statische Web-Seiten, die dann im Laufe der Zeit auf dynamische Technologien umgestellt wurden. Zu der Zeit war die Anzahl der verschiedenen Anwendungen noch sehr überschaubar. Dies hat sich seitdem jedoch sehr stark verändert, so dass es heute eine große Anzahl von verschiedensten Web-Anwendungen, -Servern und -Diensten gibt und täglich kommen weitere hinzu. Doch egal welche Software betrachtet wird, kann gesagt werden, dass sie nie zu 100% sicher sein wird. Aus diesem Grund müssen Administratoren in bestimmten Abständen Sicherheitsaktualisierungen einspielen, um so die Anwendungen möglichst gut gegen eventuelle Angreifer zu schützen. Meistens ist dies aufgrund der Vielfalt und der Komplexität heutiger IT-Landschaften jedoch sehr aufwendig. Erschwerend kommt hinzu, dass die Version der einzelnen Komponenten für eine Sicherheitsprüfung nur sehr schwer ermittelt werden kann.

Ein Web-Scanner hilft dabei eine Web-Anwendung, einen Web-Server oder einen Web-Service genauer zu identifizieren. Durch die genauere Analyse ist es möglich, die eingesetzte Version und weitere Informationen wie zum Beispiel installierte Erweiterungen in Erfahrung zu bringen. Mit den so erhaltenen Informationen kann gezielt nach möglichen Sicherheitsrisiken gesucht, eine Lösung zum Beheben gefunden und damit das System besser gegen Angriffe geschützt werden.

2 Verwandte Arbeiten

In der Vergangenheit wurden viele verschiedene Web-Scanner entwickelt. Daher werden in diesem Abschnitt zwei vorgestellt, die dem Ansatz von WeLoupe sehr ähnlich sind.

Der erste Scanner ist WhatWeb[1]. Seine Hauptaufgabe ist es, die Frage zu beantworten „Was ist das für eine Webseite?“. Mit über 1600 Plugins zur Identifikation erkennt er Content Management Systeme (CMS), JavaScript Bibliotheken, Web Server und vieles mehr. Gleichzeitig kann über einen Parameter eingestellt werden, wie aggressiv er beim Scannen vorgehen soll.

Einen etwas anderen Ansatz verfolgt der zweite Scanner mit dem Namen BlindElephant (dt. Blinder Elefant)[2]. Sein Ziel ist es, Web-Anwendungen und ihre Erweiterungen mit Hilfe von Fingerabdrücken der statischen Dateien zu erkennen. Während des Scanvorgangs wird versucht anhand von wenigen statischen Dateien festzustellen, um welche Anwendungsversion es sich genau handelt.

Im Gegensatz zu den beiden vorgestellten Scannern bietet das WeLoupe-Framework eine flexiblere Möglichkeit verschiedene Scann- und Analysetechniken anzuwenden. Gleichzeitig kann es auch sehr leicht um weitere Funktionen erweitert werden.

3 Aufbau und Umsetzung

Das WeLoupe-Framework ist in Python[3], einer universellen Programmiersprache mit einer klaren und übersichtlichen Syntax, umgesetzt. Es besteht aus einem Kern, Plugins und verschiedenen Erweiterungen. Diese Teile werden im Weiteren genauer besprochen.

3.1 Der Kern

Der Kern von WeLoupe besteht aus einer Start-, einer Konfigurations- und einer Download-Routine. Während des Starts wird die Konfiguration eingelesen und die verfügbaren Plugins und Erweiterungen geladen. Weiterhin verarbeitet er alle Download Anfragen und verwaltet den Kontext einer Web-Seite, was zum Beispiel Session-Cookies und zusätzliche Ressourcen wie Bilder etc. einschließt.

Nachdem eine Web-Seite erfolgreich abgerufen wurde, wird der Inhalt ausgewertet. Wobei dieser Prozess sich je nach dem Typ der heruntergeladen Ressource unterscheidet. Zum jetzigen Zeitpunkt verarbeitet WeLoupe zusätzliche Bilder, Stylesheets, Links und weitere Ressourcen, die durch die Web-Seite oder ein Stylesheet eingebunden sind.

[1] http://www.morningstarsecurity.com/research/whatweb
[2] http://blindelephant.sourceforge.net
[3] http://www.python.org

Nachdem der Kern erfolgreich gestartet wurde, werden alle angegeben Kommandos ausgeführt und es werden die angegeben URLs verarbeitet.

3.2 Plugins und Regeln

Der eigentliche Scanvorgang wird mit Hilfe von Plugins und darin enthaltenen Regeln durchgeführt. Dabei besteht ein Plugin nur aus einer eindeutigen ID und weiteren Informationen, die beschreiben, was mit ihm erkannt werden kann. Dies könnte zum Beispiel der Name der Web-Anwendung oder des Web-Servers sein. Die anschließende Identifikation wird dann durch die mit dem Plugin verknüpften Regeln durchgeführt. Dabei stehen unterschiedliche Typen wie die „PageContent"- oder die „PageHTTPHeader"-Regel zur Verfügung. Die erste sucht nach einem definierten Muster in einer Web-Seite, während die zweite es erlaubt die übertragenen HTTP-Header genauer zu untersuchen und in ihnen ein Muster zu finden. Wurde von einer Regel ein passendes Muster gefunden, wird das Ergebnis zurückgegeben.

Zum jetzigen Zeitpunkt gibt es zwei unterschiedliche Möglichkeiten Plugins und die damit verbundenen Regeln zu definieren. Mit der ersten können sie mit Hilfe von JSON definiert werden. Dies ist die bevorzugte Variante und ist im Fall von internen API-Änderungen sehr flexibel. Weiterhin ist es denkbar, dass diese Regeldefinitionen auch von anderen Anwendungen weiterverwendet werden könnten. In Listing 1 ist eine einfache JSON-Definition dargestellt. Die zweite und auf eine andere Weise sehr flexible Variante ist die direkte Verwendung von Python. Jedoch sollte dies nur für sehr spezielle Funktionen wie zum Beispiel der Zugriff auf eine Datenbank verwendet werden. Beide Möglichkeiten können jedoch parallel genutzt werden, da der Regelsatz eines Python Plugins immer durch die entsprechenden Regeln einer JSON-Definition erweitert werden.

3.3 Erweiterungen

Erweiterungen ergänzen den Funktionsumfang des Kerns. Dabei können sie neue Befehle, neue Plugins und neue Regeln definieren. Befehle lassen sich wiederum in drei verschiedene Typen unterteilen: einfache Befehle, Verarbeitungsbefehle und Ausgabebefehle. In diesem Abschnitt werden einige Beispiele für bereits vorhandene Erweiterungen vorgestellt.

Fetch - Die „fetch"-Erweiterung stellt den gleichnamigen Befehl „fetch" breit. Wird er ausgeführt, liest er die angegeben URLs und löst den Downloadvorgang aus.

Scan - Ähnlich wie die „fetch"-Erweiterung stellt auch die „Scan"-Erweiterung einen gleichnamigen Befehl namens „scan" bereit. Dieser steuert den Identifikationsprozess, welcher mit Hilfe der vorhanden Plugins durchgeführt wird. Liefert ein Plugin ein oder mehrere Ergebnisse, werden diese verarbeitet und an die interne Ergebnisliste der Web-Seite angehängt.

Listing 1: Beispiel Plugin-Definition mit einer Regel

```
{
  "id": "app_wordpress",
  "information": {
    "description": "Wordpress␣is␣an␣open␣source␣blogging␣and
        ␣publishing␣platform",
    "name": "Wordpress",
    "type": "app",
    "implies": ["lang_php"],
    "urls": [
      {
        "label": "Website:␣Wordpress",
        "url": "http://wordpress.org/"
      },
      {
        "label": "Wikipedia:␣Wordpress",
        "url": "http://en.wikipedia.org/wiki/WordPress"
      }
    ]
  },
  "rules": [
    {
      "id": "page-meta-generator",
      "type": "page.xpath",
      "label": "Found␣matching␣'Generator'",
      "options": {
        "regex": "(WordPress)[␣]?(?P<version>[0-9.]+)",
        "xpath": "*/meta[re:test(@name,␣'generator',␣'i')]/
            @content"
      }
    }
  ]
}
```

Dump - Die „Dump“-Erweiterung stellt den Befehl „dump-page“ zur Verfügung. Mit ihm können alle während eines Scanvorgangs heruntergeladenen Daten in Dateien gespeichert werden. Zusätzlich werden auch noch alle HTTP-Header abgespeichert und eine Index Datei erstellt. Mit diesen erstellten Informationen ist es möglich den Scanvorgang offline erneut durchzuführen. Für das „abspielen“ einer Web-Seite steht das Tool replay_proxy zur Verfügung.

Stdout - Die „Stdout“-Erweiterung stellt den Ausgabebefehl „stdout“ bereit. Dieser formatiert die beim Scanvorgang erhaltenen Informationen so, dass sie auf der Kommandozeile ausgegeben werden können.

PyV8 - Diese Erweiterung unterschiedet sich von allen anderen zum jetzigen Zeitpunkt verfügbaren Erweiterungen, denn sie stellt einen Verarbeitungsbefehl zur Verfügung. Dabei wird die V8 JavaScript Engine[4] und der pyv8 Python Wrapper[5] verwendet, um den in eine Web-Seite eingebetteten JavaScript Code auszuführen. Weiterhin stellt sie zusätzliche Regeln zur Verfügung mit denen JavaScript im Kontext einer Webseite ausgeführt werden kann und welche es ermöglichen weitere Informationen zur besseren Analyse abzufragen.

Mongo und DB - Die Erweiterungen „Mongo“ und „DB“ stellen Befehle zur Verfügung, um eine Fingerprint-Datenbank anzulegen. Mit Hilfe dieser Datenbank können die Fingerabdrücke der statischen Dateien überprüft und eventuell einer Web-Anwendung und einer Version zugeordnet werden. Dabei benutzt Mongo die MongoDB[6] und DB eine SQL-Datenbank wie MySQL[7].

4 Verwendung

WeLoupe stellt ein umfangreiches Kommandozeilen-Tool bereit. Die folgenden Beispiele sollen zeigen, wie die Komponenten einer Web-Seite identifiziert werden können. In Listing 2 werden die Befehle fetch, scan und stdout verwendet. Dabei werden Befehle immer mit einem führenden @-Zeichen gekennzeichnet. Ausgehend von dieser sehr einfachen Verarbeitung wurden in Listing 3 noch die Befehle pyv8 und dump-page hinzugefügt. Bei der Angabe der Befehle muss jedoch die Reihenfolge beachtet werden, denn sie werden nach ihrem Vorkommen von links nach rechts abgearbeitet. Gleichzeitig werden die Ergebnisse und Ressourcen eines Befehls immer an den nächsten weitergereicht, wodurch sich Verarbeitungsketten erstellen lassen. Jedoch muss auch hier unbedingt die Reihenfolge der angegeben Befehle beachtet werden. So können sich die Ergebnisse der beiden in Listing 4 dargestellten Befehle in ihrem Ergebnis durchaus unterscheiden.

[4] http://code.google.com/p/v8
[5] http://code.google.com/p/pyv8
[6] http://www.mongodb.org
[7] http://www.mysql.com

Listing 2: Einfacher Scanvorgang

```
$ weloupe @fetch @scan @stdout http://example.org/
```

Listing 3: Erweiterte Verarbeitung

```
$ weloupe @fetch @pyv8 @scan @dump-page @stdout http://
    example.org/
```

Listing 4: Einfluss der Befehlsreihenfolge auf das Ergebnis

```
$ weloupe @fetch @pyv8 @scan @stdout http://example.org/
$ weloupe @fetch @scan @pyv8 @stdout http://example.org/
```

5 Ergebnisse

Zum jetzigen Zeitpunkt sind nur wenige Plugins und Regeln verfügbar. Um die Funktionalität von WeLoupe zu demonstrieren, wurden nur Web-Anwendungen untersucht, die auch durch ein Plugin erkannt werden. In Listing 5 sind die Ergebnisse eines Scans einer Wordpress[8] 3.5 Installation mit den in Listing 2 gegeben Befehlen durchgeführt worden. Das Listing 6 zeigt einen Auszug der Ergebnisse, die mit Hilfe einer auf MongoDB basierenden Fingerprint-Datenbank ermittelt wurden.

6 Zusammenfassung

Es wurden die Struktur, die Komponenten und die Abhängigkeiten des WeLoupe-Frameworks vorgestellt. Weiterhin wurden die Verwendung und die dabei erhaltenen Ergebnisse beispielhaft dargestellt.

Wie die Ergebnisse zeigen, ist WeLoupe bereits sehr mächtig, jedoch gibt es noch Raum für Verbesserungen. Bei der Identifikation nehmen dabei die Plugins und die damit verbundenen Regeln einen wichtigen Stellenwert ein. Nach der Veröffentlichung von Version 0.2 gibt es bereits Pläne für eine Verbesserung der Scanergebnisse, die Zusammenarbeit mit anderen Scannern und für den Großeinsatz, um ein detailliertes Lagebild über die aktuelle Situation im Internet zu erhalten.

Abschließend kann gesagt werden, dass der Web-Scanner eine Hilfe für Administratoren, Sicherheitsexperten und Wissenschaftler bei der Analyse von Web-Anwendungen, Web-Servern und Web-Diensten sein kann. Es existieren bereits Pläne WeLoupe in naher Zukunft mit anderen Überwachungstechniken und Sicherheitsscannern zu kombinieren.

[8] http://wordpress.org

Listing 5: Scanergebnisse

```
$ weloupe @fetch @scan @stdout http://example.org/wordpress/
PHP
===

  * PHP is a widely spread scripting language. It's suited for web
      development and can be embedded into HTML.

Results
-------

  * Found matching HTTP-Header: X-Powered-By
    - version: 5.3.2-1ubuntu4.15

Wordpress
=========

  * Wordpress is an open source blogging and publishing platform

Results
-------

  * Found matching 'Generator'
    - version: 3.5

  * Found <link> tag with URL containing wp-content

  * Themes found in resource URLs
    - name: twentytwelve

Resource results
----------------

  * http://example.org/wordpress/wp-content/themes/twentytwelve/js/navigation
      .js?ver=1.0
    - Resource URL containing wp-content or wp-include
    - Theme resource
      + name: twentytwelve

  * http://example.org/wordpress/wp-content/themes/twentytwelve/style.css?ver
      =3.5
    - Resource URL containing wp-content or wp-include
    - Theme resource
      + name: twentytwelve


Apache
======

  * Apache is an open source HTTP Server.

Results
-------

  * Found in Server-Header
    - version: 2.2.14
    - os: Ubuntu
```

Listing 6: Scanergebnisse mit Hilfe einer Fingerprint-Datenbank

```
$ weloupe @fetch @scan @stdout http://example.org/wordpress/
[... Gleiche Ergebnisse wie im vorangegangen Listing ...]

HashIdent(MongoDB)
==================

  * Compare file hashes with known hashes from the file db.

Results
-------

  * Webapps identified
    - app_wordpress: 3.5

Resource results
----------------

  * http://example.org/wordpress/wp-content/themes/twentytwelve/js/navigation
      .js?ver=1.0
    - Identified Webapps
      + app_wordpress: 3.5-RC1, 3.5, 3.5-RC3, 3.5-RC2, 3.5-RC4

  * http://example.org/wordpress/wp-content/themes/twentytwelve/style.css?ver
      =3.5
    - Identified Webapps
      + app_wordpress: 3.5
```

WeLoupe steht als Open Source zum Download unter http://git.dinotools.org zur Verfügung. Gleichzeitig existiert ein Repository für Plugins und Regeln, welches zur Mitarbeit bereitsteht und unter https://github.com/DinoTools/weloupe.rules zu finden ist.

Zuverlässigkeit von GNU/Linux-basierten eingebetteten Systemen

Wolfram Luithardt, Daniel Gachet und Jean-Roland Schuler

Hochschule für Technik und Architektur Freiburg, Boulevard de Perolles 80,

CH-1705 Fribourg, Schweiz

1. Einleitung

Wer kennt sie nicht, die vielen spektakulären Beispiele, in denen der Ausfall von Software- oder Hardwarekomponenten zum Ausfall ganzer Systeme führt. Am 4.6.1996 explodierte z.B. die Ariane 5 auf ihrem Jungfernflug. Schaden 1,7 Mia DM. Grund für den Absturz war der Überlauf einer Variablen der horizontalen Beschleunigung der Rakete. Dieser Teil der Software wurde vom Vorgängermodell, der Ariane 4, übernommen, bei der wesentlich geringere Beschleunigungsraten auftraten. Nach dem Überlauf der Variablen schalteten sich Korrekturtriebwerke ein, die im Endeffekte solche Kräfte auf die Rakete ausübten, dass sie auseinanderzubrechen drohte, wodurch sie sich selbständig sprengte. Obwohl dieser Fall sehr alt erscheinen mag, ist es doch ein gutes Beispiel, da der Fehler vollkommen aufgeklärt und die Ursache publiziert wurde. Dies ist nicht immer der Fall: Im April 2009 legte ein Softwarefehler im Server des Home Location Register das gesamte Mobilfunknetz der Telekom für 5 Stunden lahm. Vielleicht weniger katastrophal aber zumindest sehr ärgerlich.

Während es in der Hardware elektronischer Systeme zu einer natürlich Alterung der Komponenten durch verschiedene physikalische Prozesse (Elektromigration, Diffusion, etc.) kommen kann, was zu spontanen Ausfällen führen kann, sollten Softwarekomponenten eigentlich deterministisch sein, d.h. Fehler sollten unter gleichen Ausgangsbedingungen reproduzierbar auftreten. Spontanes Auftreten von Ausfällen in der Software kann nur durch ein komplexes Zusammenspiel von verschiedenen, oft zeitabhängigen Effekten beschrieben werden. Solche Effekte sind immer auch ein Zeichen für Programmierfehler, welche selbst durch ausführliche Tests oft nur schwer gefunden werden können. In diesem Beitrag wollen wir etwas Licht in die komplexe Zuverlässigkeitstheorie bringen und an einigen Beispielen zeigen, wie sich Fehler in Programme schleichen können, die dann trotz aller Tests plötzlich 'zuschlagen'. In Kapitel 2 betrachten wir kurz die klassische Zuverlässigkeitstheorie im Vergleich von Hard- zu Softwaresystemen. In Kapitel 3 zeigen wir einige Beispiele für zuverlässigkeitsrelevante Softwarefehler. In Kapitel 4 diskutieren wir einige Werkzeuge, mit denen die Zuverlässigkeit analysiert werden kann. In Kapitel 5 geben wir dann noch einige Hinweise, wie Software in Bezug auf die Zuverlässigkeit verbessert werden kann.

Das Erstellen von zuverlässiger Software beinhaltet immer mehrere Stufen:

a.) Zuverlässigkeitsoptimiertes Design

b.) Vermeidung von Softwarefehlern in der Implementierungsphase

c.) Analyse und Tests im Hinblick auf Zuverlässigkeit (Hardware, Software und des Gesamtsystems). Für alle 3 Teile existieren zahlreiche hierarchische, modularisierte Validations- und Testmethoden [1,2].

d.) Reduktion der Auswirkung von Softwarefehlern

Dabei sind alle 4 Stufen unabdingbar damit ein System wirklich stabil und zuverlässig läuft. Die Schritte b.) und c.) sind dabei iterativ; a.) und d.) müssen bereits bei der Konzeption des Systems berücksichtigt werden.

Die Materie ist natürlich sehr komplex und könnte ganze Bücher füllen. In diesem kurzen Beitrag können wir deshalb nur einen kurzen Einblick in verschiedene Teilbereiche geben.

2. Vergleich von Hard- und Sofwarezuverlässigkeit

In mechanischen und elektronischen Systemen wird die Zuverlässigkeit meist über die sogenannte Ausfallrate λ(t) beschrieben. Diese Grösse besitzt die Einheit 1/Zeiteinheit, häufig 1/h. Da diese Grössen natürlich relativ klein sind, wird häufig die Einheit 1 FIT verwendet, wobei 1 FIT = 10^{-9}/h ist. Typische Werte für eingebettete Systeme liegen bei 1'000 - 10'000 FIT.

Grundlage für λ(t) ist die Annahme, dass bei einer grossen Anzahl von Systemen diese nacheinander in statistisch verteilter Wahrscheinlichkeit ausfallen. Zu einem Zeitpunkt t=0, bei dem die Systeme in Betrieb gehen, ist die Anzahl der ausgefallenen Systeme gleich 0, zum Zeitpunkt t= ∞ sind alle Systeme ausgefallen. Diese für eine grosse Anzahl von Systemen definierte Grösse kann leicht auch als Ausfallwahrscheinlichkeit für jedes einzelne System interpretiert werden. λ(t) gibt dann die Wahrscheinlichkeit an, mit der ein System, das bis zum Zeitpunkt t nicht ausgefallen ist, im Zeitraum zwischen t und t+Δt ausfallen wird. Erfahrungsgemäss folgt die Ausfallrate λ(t) der sogenannten Badewannenkurve, wie sie in Abbildung 1 dargestellt ist. Direkt nach Inbetriebnahme der Systeme ist die Ausfallrate relativ hoch, man spricht in diesem Fall auch von Frühausfällen. Nach einer gewissen Zeit sinkt λ auf einen niedrigeren und über eine relativ lange Zeit konstante Ausfallrate um danach wieder stark anzusteigen. Interessant ist, dass auch die Ausfallraten in biologische Systeme (bei Menschen spricht man von Sterberaten) häufig einer ähnlichen Form folgt. Im Bereich des normalen Lebenszyklus ist λ(t) = λ zeitlich konstant, was bedeutet, dass die Systeme nicht altern. In diesem Fall wird der Kehrwert von λ als MTBF (Mean Time between Failure) bezeichnet. MTBF beschreibt die Zeit, bis zu der

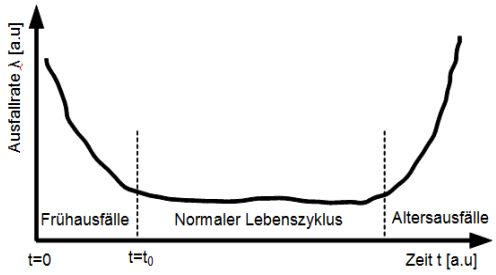


Abb.1: Ausfallrate elektronsicher Systeme

63% der Geräte bereits ausgefallen sind. Wenn die Ausfallrate zeitlich nicht konstant ist, spricht man häufig von MTTF (Mean Time To Failure). In wichtigen Systemen, in denen vermieden werden soll, dass die Ausfallrate für Frühausfälle sehr gross ist, wird häufig ein sogenanntes Burn-In durchgeführt. Dadurch wird durch eine beschleunigte Alterung (meist bei höherer Temperatur) in kurzer Zeit die Ausfallrate auf $\lambda(t_0)$ getrieben. Für weitere Informationen zu diesem sehr komplexen Thema sei z.B. auf [3] verwiesen.

Da Software keine statistisch beschreibbare Ausfallrate besitzt, sondern eine statische (Fehler sollten deterministisch auftreten!) fällt hier die Definition schwerer. Dennoch können im Lebenszyklus der Software Fehlerraten (also auftretende Fehler/Zeiteinheit) beschrieben werden. So ist sicherlich am Anfang die Fehlerrate hoch, bei einer kontinuierlichen Verbesserung oder einem Erlernen des Umgangs mit den Fehlern sinkt diese. Nur nach der Implementierung von neuen Features wird die Fehlerrate wieder ansteigen. Dies ist in Abbildung 2 dargestellt. Nach langer Laufzeit wird die Fehlerrate auf einen kleinen und konstanten Wert sinken. Wenn eine Software lange Zeit nicht in Betrieb war und auf einem neuen System wieder in Betrieb genommen werden soll, so entsteht meist der gestrichelte Anstieg. Jeder, der schon mal versucht hat eine lange nicht gepflegte Software erneut in Betrieb zu nehmen, wird dies bestätigen ;-)

Abb. 2: Fehlerrate bei Software

3. Beispiele für zuverlässigkeitsrelevante Softwarefehler

Prinzipiell stellt jeder Programmierfehler ein potentielles Zuverlässigkeitsproblem dar. Viele Fehler werden jedoch bei Unit- oder Funktionstests entdeckt und können sofort behoben werden. Viel schwieriger wird es, wenn Programmierfehler erst nach einer langen Laufzeit oder nur bei einer bestimmten Konstellation von internen oder externen Bedingungen auftreten. In diesem Fall sind diese Fehler durch Tests nur schwer auffindbar. Es ist also wichtig, diese Fehler erst gar nicht zu machen. Wir haben im folgenden 4 typische Kategorien zusammengestellt, die alle zu Fehlern führen können, die nur selten auftreten.

a.) Funktionsrückgabewerte nicht richtig verarbeitet

In Snippet 1 ist ein typisches Beispiel einer unsauberen Programmierung dargestellt. Der Aufruf von read kann mehrere Jahre lang gutgehen und damit läuft diese Sequenz fehlerfrei. Bis eines schönen Tages das read einen Fehler von -1 zurückgibt. Gründe dafür können in einem Fehler des Zugriffs auf die Ressource lie-

```
int fd, ret, i = 12345;   // some variable
char buffer[100];
fd = open(....);
ret = read (fd, buffer, sizeof(buffer)-1);
buffer[ret] = '\0';
```

Snippet 1: Gefährliches Lesen

gen, oder noch einfacher dadurch, dass das Blocking read durch ein Signal unterbrochen, sprich entblocked wird. In diesem Fall kommt der Systemaufruf read mit einem Rückgabewert von -1 zurück und die globale Variable errno wird auf EINTR gesetzt. Damit wird also buffer[-1] auf '\0' gesetzt und damit die Variable i verändert. In vielen Fällen wird das Überschreiben eines Bytes der 32-bit-Variablen i mit binärem 0 sogar überhaupt keine Auswirkung haben, bei anderen Werten von i wird diese Variable jedoch massiv verändert. Solche Effekte sind extrem schwer zu finden, da sie sich meistens an ganz anderen Stellen auswirken, als wo sie aufgetreten sind. Dies zusammen mit dem nichtdeterministischen Auftreten des Fehlers sind Merkmale von massiven Zuverlässigkeitsproblemen der Software.

```
char buffer[100]; char* b=buffer;
int fd, ret, number=sizeof(buffer)-1;
fd = open (....);
while (number !=0 && (ret = read (fd, b, number)) != 0){
    if(ret == -1) {
        if (errno == EINTR) continue;
        perror("read");
        break;
    }
    number -= ret;
    b += ret;
}
*b = '\0';
```

Snippet 2: Besseres Lesen von einem File

Wie müsste die obige Sequenz richtig programmiert werden? Ein einfaches Testen des Rückgabewertes ret reicht nicht, da dadurch ein entblocktes read (z.B. durch ein Signal) seine eigentliche Aufgabe nicht richtig erfüllen würde. Eine 'richtige' oder zumindest 'bessere' Implementierung ist in Snippet 2 dargestellt. Prinzipiell sollte der Rückgabewert jeder Funktion auf Konsistenz getestet werden. Ebenso muss beim Design des Programms genau überlegt werden, was mit nicht richtigen Rückgabewerten gemacht wird. Ausführliche Tests der Rückgabewerte verlängert zwar den Sourcecode massgeblich, ist aber unabdingbar um gute, stabile Software zu erstellen.

b.) Überlauf von Variablen.

Der Überlauf von Variablen, ganz egal, welchen Typs, führt besonders dann zu einer Gefahr, wenn entweder unbekannte Grössen (z.B. Rauschen) mit in eine Berechnung eingehen oder die Spezifikation der Software nach Erstellen noch geändert wird (z.B. durch Änderung von HW-Komponenten (Sensoren)). In Abbildung 3 ist ein einfaches Beispiel graphisch darge-

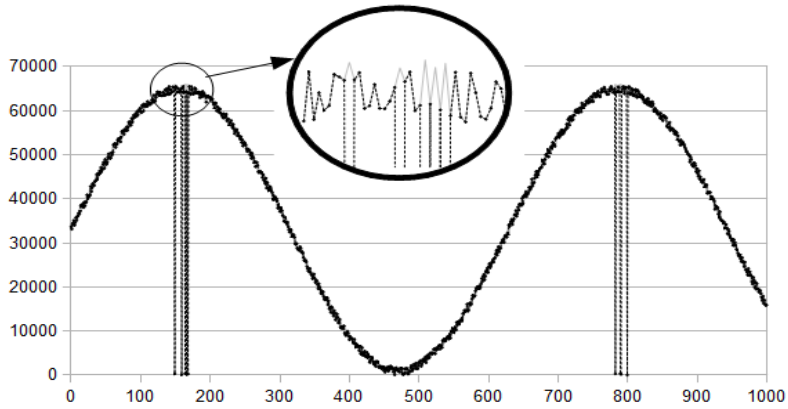


Abbildung 3: Typischer Variablenüberlauf

stellt. Auf einen einfachen Sinus wurde noch ein zufälliges Signal addiert, was manchmal zu einem Überlauf führt. Ariane lässt grüssen!!!

In der Praxis sind solche Probleme natürlich wesentlich perfider und schwerer zu erkennen. Möglichkeiten, diese zu umgehen sind entweder eine genaue Analyse der möglichen auftretbaren Werte oder die Verwendung von Rechenoperationen, die gegen Überläufe geschützt sind. Eine Möglichkeit ist z.B. das makrobasierte Framework **safe_iop**, das sichere Rechenoperationen mit Integer zur Verfügung stellt [5]. So wird z.B. ein c = a * b zu einem safe_mul(&c, a, b). c enthält danach das Resultat; der Rückgabewert des Makros gibt an, ob das Resultat einen Überlauf verursacht hat oder nicht. Natürlich ist es recht aufwendig, eine bestehende Software auf dieses Framework umzuschreiben. Es wäre natürlich viel besser, wenn der Compiler selbst überlaufsichere Operationen anwenden würde, was aber zumindest in C nicht der Fall ist. Für Floatingpoint-Berechnungen kann auch das Signal für Floatingpoint-Fehler SIGFPE durch Installieren eines entsprechenden Signalhandlers abgefangen werden, das dann den Fehler bearbeitet oder zumindest vor ihm warnt.

Eine besondere Art von Variablenüberläufe entsteht häufig beim Arbeiten mit Arrays. C bietet im Gegensatz zu anderen Programmiersprachen nur elementare Möglichkeiten, Indizes die aus dem deklarierten Bereich laufen, zu erkennen und zu vermeiden. Hier ist strikte Programmierdisziplin gefordert, um solche Probleme zu vermeiden. Bei Texten (array von char), die mit einer binären 0 abgeschlossen werden, ist ebenfalls eine strikte Verwendung der Standardbibliothek oder eine grosse Disziplin notwendig. Eine häufig angewandte Methode um zumindest ein deterministisches Verhalten zu erzwingen ist, jeden Text, der neu beschrieben wird, vorher mit memset(the_array, '\0', sizeof(the_array)); vollständig zu löschen. Damit wird vermieden, dass der Inhalt der Variablen, neben dem eigentlichen Fehler, auch noch von der Historie der Variablen abhängt.

c.) Race Conditions

Eine Race-Condition ist eine Eigenschaft beim Ablauf eines Programms bei der das Ergebnis von der genauen Reihenfolge der Instruktionen abhängt. Sie entsteht häufig, wenn 2 pseudo-parallel ablaufende Tasks auf dieselbe Datenbasis zugreifen. Wenn während einer Verarbeitung von Daten der ablaufende Task unterbrochen und auf einen anderen task umgeschaltet wird, dieser 2 Task dieselben Daten verarbeitet, kann das Endergebnis der Verarbeitung vom genauen Zeitpunkt der Unterbrechung abhängen. Race-Conditions sind sehr perfide, da sie eventuell nur sehr selten auftreten und dadurch sehr schwer zu finden sind. Sie sind aber meist ein Zeichen von schlechtem Programmdesign, da niemals 2 verschiedene Tasks ohne Synchronisation auf dieselben Daten zugreifen sollten. Bei Multiprocessing-Systemen, bei dem verschiedene Prozesse jeweils einen unterschiedlichen virtuellen Speicherbereich besitzen ist der Zugriff auf gemeinsame Ressourcen eher selten. Wenn die Prozesse miteinander Daten austauschen müssen, so muss das über eine spezielle Form von Interprozesskommunikation (IPC) implementiert werden. Beim gleichzeitigen Zugriff auf globale Ressourcen wird die Synchronisation meist über Semaphore erreicht, mit denen sogenannte atomic operations implementiert werden können. Mit diesen wird erreicht, dass eine Verarbeitung eines Datums, also meist ein Lesen, ein Modifizieren und ein Schreiben einer Variablen, ununterbrechbar wird. In Multi-Threading Programmen, in denen verschiedene Threads auf dieselbe Datenbasis zugreifen, muss im Prinzip jede Variable vor Race-Conditions geschützt werden. Dafür werden meistens Mutexes verwendet. Beim Design eines Programms sollte also immer abgewogen werden, ob eine Implementierung der Auf-

gabe als Multiprozessingsystem mit einer vernünftigen IPC, oder als Multithreadingsystem mit vielen Mutexes zu bevorzugen ist.

d.) Memory-Leaks

Memory Leaks entstehen, wenn ein Programm Speicher alloziert, diesen aber nach Gebrauch nicht mehr freigibt. Dadurch wird der verfügbare Speicher langsam verbraucht und ein weiterer Versuch, Speicher zu allozieren schlägt fehl. Memory Leaks sind deshalb zuverlässigkeitsrelevant, da die Fehler erst nach langer Betriebszeit der Software auftreten, wenn niemand mehr damit rechnet. Memeory Leaks entstehen häufig, wenn das Design einer Routine. Das folgende Snippet zeigt ein typisches Problem, das zu Memory Leaks führen kann.

```
int foo (void){
   struct Struc *s= malloc (sizeof(struct Struc));
   if (s == NULL) return(-1);
   ....
   if (some_error) return(-2);
   ...
   free(s);
   return(0);
}
```

Snippet 3: Memory Leak

Läuft alles gut in der Funktion wird der Speicher wieder freigegeben, im Fehlerfall (some error) allerdings nicht. Je häufiger also der Fehler auftritt, desto mehr Speicher wird nicht verbraucht. Ein Absturz der der Software, verursacht durch zu wenig verfügbarem Speicher, kann erst nach Jahren auftreten. Solche Fehler können leicht vermieden werden, wenn beim Programmieren darauf geachtet wird, dass jede Funktion nur ein einziges return() besitzt. Somit wird man dazu gezwungen, die Funktion sauber abzuschliessen. Sollte das Deallozieren vergessen werden so wird zumindest bei jedem Aufruf Speicher verbraucht, was dazu führen kann, den Fehler schneller zu finden. Es gibt aber auch viele Tools, mit denen Memory Leaks gefunden werden können. Meist ist ein erster Schritt ein regelmässiger Aufruf von **free**, **top** oder **ps** in der Testphase des Programms. Hiermit kann der benötigte Speicherplatz des Prozesses beobachtet werden. Es gibt aber auch besser Methoden: mit der Libraryfunktion **mtrace()** wird ein Logfile mit den Allokationen und Deallokationen erstellt, das einfach zu analysieren ist. Mit **memwatch** werden falsche Zugriffe auf allozierten Speicherplatz erkannt. Auch mit valgrind (siehe Kapitel 4) können relativ einfach Memory Leaks erkannt werden.

Natürlich gibt es noch viele weitere potentielle Elemente, die zu zuverlässigkeitsrelevanten Fehlern führen können, die aber hier nur kurz genannt werden sollen:

- Die Reihenfolge von Operatoren ist häufig nicht eindeutig. Besonders achtgeben muss man bei der Verwendung von Macros. Vorallem bei der Verwendung von Präfix- und Postfix-Operatoren ist es oft nicht klar, was zuerst abgearbeitet wird.
- Stackoverflow vor allem bei rekursiver Programmierung, aber auch bei Übergabe von grossen Datenmengen in Funktionen.
- Alignment-Effekte: Variablen in 32-Bit- oder 64-Bit- Prozessoren müssen meist an den 4 Byte-Grenzen ausgerichtet sein. Sind sie es nicht, entsteht ein Speicherzugriffsfehler. Dies ist besonders wichtig, wenn Strukturen mit __*attribute*__ ((packed)); compiliert werden.

4.) Werkzeuge zur Vermeidung von Softwarefehlern

Unter GNU/Linux gibt es bereits viele Werkzeuge, die es erlauben, Software auf zuverlässigkeitsrelevante Aspekte zu analysieren und zu testen. In der GNU Compiler Collection gcc verfügt der der Compiler über einige Flags, die bei der Compilation oder des Linkens zuschaltbar sind und auf Softwarefehler hinweist. Mit **-Wall** und **-Wextra** lassen sich z.B. viele Warnmeldungen ausgeben, die auf strukturell fragwürdigen aber leicht verbesserbaren Sourcecode hinweist.

Auch mit Hilfe vieler, in jedem System vorhandener kleiner Programme können potentielle Probleme aufgespürt werden. Mit **free**, **ps** oder **top** kann der Speicherverbrauch des Systems oder der einzelnen Prozesse beobachtet werden.

Systemaufruf können mit **strace** verfolgt werden, was helfen kann, zuverlässigleitsrelevante Probleme aufzuspüren. Das **ltrace**-System kann zum tracen von Library-Funktionen verwendet werden und **ftrace** zum Tracen des des Linuxkernels.

Ein Werkzeug, das speziell für die Analyse von dynamischen Speicherzugriffen und Multitaskingeffekten verwendet werden kann, ist **valgrind**. Es handelt sich dabei um eine Art virtuelle Maschine, die den ausführbaren Programmcode in eine prozessorneutrale Zwischenform, die dann analysiert und ausgeführt wird. Damit können viele der oben beschriebenen Problematiken wie Race Conditions, Memory-Leaks und Variablenüberläufe entdeckt werden. valgrind ist darauf angelegt, dass auch weitere Analysen relativ einfach selbst erstellt werden können. Des Weiteren existieren auch eine ganze Reihe von graphischen Werkzeugen, die die recht kryptischen Ausgaben von valgrind etwas lesbarer darstellt. valgrind kann auch für Profiling-Aufgaben verwendet werden.

5.) Zusammenfassung und Tipps

Neben Analysen und Tests einer erstellten Software können und müssen bereits im Vorfeld, also während der Planung und des Schreibens von Software bestimmte Prinzipien angewandt werden, um das System in Bezug auf seine Zuverlässigkeit zu optimieren.

Architekturelle Massnahmen

Eine saubere Trennung zwischen den unterschiedlichen Aufgaben erleichtert nicht nur die Testbarkeit von Software sondern steigert auch ihre Zuverlässigkeit. Ein System, bei dem jedes Teilsystem an sich fehlerfrei läuft, ist zwar nicht zwingend als Gesamtheit fehlerfrei, jedoch können Schwachstellen in den Interfaces wesentlich einfacher gefunden werden, als wenn nicht genau klar ist, woher das Problem wirklich stammt.

Eine gute, architekturelle Methode zur Vermeidung von Race-Conditions ist die Verwendung von Multiprocessing anstelle von Multithreading. Der Aufwand, eine saubere IPC zu implementieren ist zwar grösser, kann aber eventuel durch einen wesentlichen

zuverlässigeren Austausch von Werten ohne Race-Conditions ausgeglichen werden.

System sollten auch immer so gestaltet werden, dass Sie fehlertolerant arbeiten. Ist in einem Prozess ein Problem aufgetreten, dass z.B. zum Absturz des Prozesses führt, darf die Integrität des Gesamtsystems nicht beeinträchtigt werden. Zwar kann ein abgestürzter Prozesse einfach durch ein respawn-Atrribut in der inittab wieder automatisch aufgestartet werden, Kommunikationspfade und

Defensives Programmieren

Unter defensivem Programmieren wird verstanden, dass ein Programm so erstellt wird, dass es alle Interaktionen mit der Aussenwelt auf Konsitenz testet. Bei User-eingaben ist dies offensichtlich. Viele Funktionen der C-Standardbibliothek bieten die Möglichkeit, Eingaben über reguläre Ausdrücke zu beschreiben und damit zu filtern. Häufig wird die Eingabe selbst über gets(Achtung auf Buffer-Overflows!) durchgeführt und dann das Resultat mittels scanf() und regulären Ausdrücken gefiltert. Ganz Allgemein können mit regulären Ausdrücken extrem gut und transparent (evt. etwas kryptisch) Eingaben gefiltert werden. Für Umwandlungen von ASCII-Eingaben zu Integer- oder Floatvariablen gibt es Funktionen (strtol() oder strtod()) die eine automatische Prüfung durchführen und im Fehlerfall einen Fehlercode zurückliefern. atoi() und seine Freunde tun dies nicht.

Funktionrückgaben sollten immer geprüft werden. Damit wird vermieden, dass sich, durch einen fehlerhaften Funktionsaufruf entstandene, Rückgabewerte weiter verbreiten und an einer anderen Stelle Schaden anrichtet. Solche nichtlokalen Fehler sind wesentlich schwieriger zu finden.

Tests

Mit Tests können zwar Fehler gefunden werden, aber nur in den wenigsten Fällen kann die Fehlerfreiheit garantiert werden. Es gibt ein Vielzahl unterschiedlicher Tests, angefangen von Komponententests bis zu Systemtests. Je nach Komplexität des Programms sollten aber immer mehrere verschiedene Tests von verschiedenen Personen durchgeführt werden. Einen guten Einblick in die Materie gibt [1, 2].

Literatur

[1] U.Vigenschow: Testen von Software und Embedded Systems, dpunkt Verlag (2010).

[2] D. Mölle: Testsieger, Stabile Software durch Design for Testability, iX 11/2012, 86-89.

[3] A. Meyna und B. Pauli, Zuverlässigkeitstechnik, 2. Auflage, Hanser 2010.

[4] M. Grottke: Prognose von Softwarezuverlässigkeit, Softwareversagensfällen und Softwarefehler, in P. Mertens und S. Rässler(Hrsg), Prognoserechnung , 7. Auflage, Springer/Physica, 2012, S. 585-619,

[5] http://code.google.com/p/safe-iop/

3 Zusammenfassungen der weiteren Vorträge

3.1 3D-Visualisierung mit Blender

Erik Schufmann, ROGALL Architekten Ingenieure, info@erikschufmann.de

In diesem Vortrag sollen den Teilnehmern nach einem kurzen Überblick über die Blender Foundation die Möglichkeiten der Open-Source-3D-Suite Blender aufgezeigt werden. Angerissen werden die Bereiche UI, Modellierung, Materialien, Beleuchtung, Rendern, Compositing, Cycles, welche an konkreten Beispielen aus der Berufspraxis anschaulich dargestellt werden. Die Möglichkeiten der Einbindung in proprietäre Software werden ebenfalls angesprochen.

Weitere Informationen: http://www.blender.org

3.2 Aktuelle Entwicklungen beim Linux-Kernel

Thorsten Leemhuis, Heise Zeitschriften Verlag: c't/heise open, thl@ct.de

Der Vortrag gibt einen Überblick über die jüngsten Verbesserungen des Linux-Kernels, denn viele von ihnen sind auch für Allerwelt-PCs oder Server von Belang; mit Distributionen wie Ubuntu 13.04 erreichen sie in Kürze eine breite Anwenderschar.

Der Vortrag geht auch auf einige Neuerungen bei Kernel-naher Software ein – etwa den Open-Source-3D-Grafiktreibern. Angerissen werden auch einige noch in Vorbereitung befindliche Änderungen, der Entwicklungsprozess sowie andere Aspekte rund um den Kernel, die für die kurz- und langfristige Entwicklung von Linux und Linux-Distributionen wichtig sind.

Weitere Informationen: http://www.heise.de/open/kernel-log-3007.html

3.3 Aktuelles vom Supercomputing – Storage Cluster mit ZFS

Peter Großöhme, MEGWARE Computer GmbH, peter.grossoehme@megware.com
Johan van den Boogaart, Nexenta Systems Inc.

Der Vortrag vermittelt die Grundlagen des High Performance Computing und gibt einen kurzen Ausblick auf die neuesten Technologien und Trends in diesem Bereich.

Im Anschluss daran werden das Dateisystem ZFS näher beleuchtet sowie verschiedene Konfigurationen und Einsatzmöglichkeiten zum Betrieb eines hochverfügbaren Storage-Clusters mittels NexentaStor aufgezeigt.

3.4 All about FreeYourAndroid!

Erik Albers, Free Software Foundation Europe

Android is an operating system that is published as Free Software. Unfortunately, the original code by Google tries to tie you to a Google account and the Google services to

use the whole functionality of your device. Other resellers add proprietary apps to the system that you cannot get rid off. Most of them even lock the bootloader to prevent you from installing alternative systems. FSFE's FreeYourAndroid campaign aims at giving back the full control to all users over their devices and the software they are running. This talk will show you how to do so and details about the campaign.

Weitere Informationen: http://freeyourandroid.org

3.5 Anständiges Werkzeug: emacs

Erich Wälde

Gutes und gut erlerntes Werkzeug ist der Schlüssel zum effektiven Arbeiten. Die wichtigsten Werkzeuge im Linuxland sind die Shell und der Editor; hier habe ich mich für «emacs» entschieden.

Im ersten Teil wird der Einstieg gezeigt: Editieren, Nutzung der Hilfe, Beenden. Die Hörer werden ermuntert, emacs einmal in die Finger zu nehmen.

Im zweiten Teil wird vorgeführt, was emacs noch so kann: Texte editieren, beim Programmieren helfen, mit dem Betriebssystem reden, Zeiten, To-dos, Kontakte organisieren etc.

Weitere Informationen: http://www.emacswiki.org/

3.6 Aptitude – bekannt und doch unbekannt

Axel Beckert, ETH Zürich, abe@debian.org
Frank Hofmann, Hofmann EDV – Linux, Layout und Satz, frank.hofmann@efho.de

Apt-Get und Aptitude werden oft als gleichwertige Paketmanager angesehen. Deshalb wird häufig wahllos einer der beiden verwendet. Aber trotz ähnlicher Parameter auf der Kommandozeile unterscheiden sie sich in vielerlei Hinsicht.

Aptitude bietet u.a. eine Text-Modus-Benutzeroberfläche (TUI) und kann Paketaktionen vormerken. Aber auch auf der Kommandozeile erlaubt es wesentlich komplexere Operationen als Apt-Get.

Der Vortrag wird sowohl auf die Unterschiede zwischen Apt-Get und Aptitude auf der Kommandozeile eingehen als auch die Text-Modus-Oberfläche vorstellen.

Weitere Informationen: http://noone.org/talks/aptitude/

3.7 Best Practice OTRS – Serviceverträge mit OTRS planen, verwalten und überwachen

Rico Barth, c.a.p.e. IT GmbH, info@cape-it.de

Betrachtet wird die Projektpraxis bei der Einführung des Servicevertragsmanagements auf Basis von OTRS. Ausgehend von den Anforderungen wird auf die Konfigurationsmöglichkeiten von Servicekatalogen und Servicevertragsparametern (SLA) eingegangen. Vorgestellt werden die damit in Zusammenhang stehenden Servicekriterien und

unterschiedlichen Eskalationsmethoden von OTRS. Die praktischen Herausforderungen bei der Planung des Servicekatalogs und der SLA beleuchtet der Vortrag ebenso. Die SLA-Überwachung mittels Dashboard und Berichten zu sinnvollen Messgrößen rundet den Praxisvortrag ab.

Weitere Informationen: http://www.cape-it.de, http://www.otrs.org, http://www.otterhub.org

3.8 Building Email Server Cluster with Dovecot

Timo Sirainen, Dovecot, tss@iki.fi

Dovecot is a highly flexible IMAP/POP3 server, which allows it to operate in many different environments. This also means it can sometimes be difficult to figure out which features to use. This talk gives several tips about common issues in running Dovecot and describes in detail one way to build a highly available Dovecot cluster.

Weitere Informationen: http://www.dovecot.org

3.9 Community Matters

Elmar Geese

Communities haben die Welt verändert: Freie Software, Open Source, Wikipedia, Hacker, Netzaktivisten … Die schnelle und strategisch durchdringende Verbreitung von Open Source ist eine einzigartige Erfolgsgeschichte. Das Wachstum hat viele Vorteile und wenige Probleme gebracht. Open Source hat den Einstieg von Google, Microsoft, IBM und allen anderen gut verkraftet. Wie geht es weiter? Ist die Community noch die treibende Kraft, oder sind es die Unternehmen? Eine selbstbewusste Bestandsaufnahme zum Erreichten und zur Zukunft.

3.10 Computer und Hacking überall mit Raspberry Pi & Co

Alexander Schreiber, Google Switzerland GmbH, als@thangorodrim.de

Mit Raspberry Pi, Hackberry und anderen gibt es mittlerweile mehrere kompakte, preiswerte ARM-basierte Computer. Auf der einen Seite sind sie leistungsfähig genug, um auch für anspruchsvollere Aufgaben verwendbar zu sein (MediaCenter), zum anderen klein und lautlos, dass sie dezent überall eingesetzt werden können. Das Vorhandensein von GPIO bei z.B. Raspberry Pi macht den einfachen Anschluss eigener Elektronik möglich.

Der Vortrag soll einen Überblick über das Angebot sowie verschiedene Beispielprojekte geben und über die Erfahrungen des Autors berichten.

3.11 Das fast nützliche Perl-5-Quiz

Herbert Breunung, Selbstständig, lichtkind@cpan.org

Für alle, die glauben, von Perl Ahnung zu haben: Zeit für einen Realitätscheck. Kennt ihr auch die Ecken und Ritzen der Scriptsprache, die kaum jemand beachtet? Manchmal enthalten sie wirklich nützliche Funktionen. Auch die Neuerungen der letzten

Versionen enthalten winzige Änderungen, die in üblichen Zusammenfassungen übergangen werden, sich aber im Programmieralltag ernsthaft auswirken können.

Weitere Informationen: http://lichtkind.de/vortrag/

3.12 Datenschutzrecht für Linux-Admins in Betrieben und Behörden

Christian Klostermann, Rechtsanwalt Dr. Klostermann, Zwickau , kanzlei@drklostermann.de

Der Vortrag gibt einen Überblick über das aktuelle Datenschutzrecht, das Linux-Administratoren in ihrer täglichen Arbeit in Betrieben und Behörden zu beachten haben.

Unter dem Motto «Was kann, und was darf!» werden aktuelle Fragen aus dem Datenschutz behandelt, insbesondere die Frage nach der Kontrolle von privatem E-Mail- und Internetverkehr im Betrieb.

Weitere Informationen: http://www.drklostermann.de

3.13 Debian GIS – GIS und OpenStreetMap mit Debian-Bordmitteln

Andreas Tille, Debian GNU, tille@debian.org

Debian GIS ist ein «Pure Blend», das sich die Betreuung von Paketen zum Thema Geoinformationssysteme und OpenStreetMap zum Ziel gesetzt hat. Der Vortrag gibt einen Statusbericht des Projektes und zeigt auf, wo noch Hilfe gebraucht wird.

Weitere Informationen: https://wiki.debian.org/DebianGis

3.14 Der mount-Befehl, die Wahlfreiheit für das Dateisystem

Thomas Winde, Thomas Winde Ausflugsfahrten, ausflug@web.de

Auch wenn auf vielen Rechnern USB-Sticks, CDs und DVDs automatisch in das Dateisystem eingebunden werden, sollte man sich mit dem mount-Befehl beschäftigen und ihn kennen. Damit kann man verstehen, was im Rechner technisch vor sich geht, wie man das Dateisystem aufbaut und viele kleine Zusatzfunktionen nutzt.

3.15 Der NAO – ein humanoider Roboter auf Linux-Basis

Markus Dittmann, TU Chemnitz

Der Aldebaran NAO ist ein humanoider Roboter, der unter anderem beim RoboCup sowie an verschiedensten Forschungseinrichtungen zum Einsatz kommt. Als Basis für die Steuerungssoftware dient dabei ein GNU/Linux-System. Im Vortrag soll ein Überblick über Hardware, Betriebssystem und insbesondere die Möglichkeiten, Programme und Komponenten für den Roboter zu entwickeln, gegeben werden. Eine kurze Demonstration und Programmierbeispiele in C++ oder Python sind vorgesehen.

Weitere Informationen: http://www.aldebaran-robotics.com/en/

3.16 Die Digitalisierung von Büchern

Peter Koppatz, Sudile GbR

Die Digitalisierung von Büchern gehört inzwischen zu den Standardaufgaben vieler Institutionen. In dem Vortrag soll das Projekt zur Digitalisierung der zwölfbändigen Gesamtausgabe des Dichters Friedrich Rückert vorgestellt werden. Es werden die eingesetzten Programme, der Arbeitsablauf und die besonderen Schwierigkeiten bei der Umsetzung dargestellt und zur Diskussion gestellt.

Weitere Informationen: http://rueckert.gesammelte-werke.org/

3.17 DNS im RZ-Betrieb mit PowerDNS und MySQL

Tobias Stein

Tilo Werner, activate communication systems gmbh, tilo+clt2013@moosbee.de

Dieser Vortrag stellt ein im Rechenzentrum durchgeführtes Projekt des letzten Jahres vor, welches zum Ziel hatte, die Benennung der Server zu vereinheitlichen und die Wartung der DNS-Informationen möglichst sicher und einfach zu gestalten.

Es wird ein Einblick in den Aufbau einer DNS-Infrastruktur zum internen Gebrauch im RZ gegeben. Mit praktischen Vorführungen ist dieser Vortrag auch als Live-How-To zu verstehen.

Schwerpunkte sind: DNS mit PowerDNS, MySQL als Backend, MySQL-Replikation mit x509-Zertifikaten (TinyCA), und Poweradmin als Web-Frontend.

3.18 Ein hochverfügbarer iSCSI Storage mit Linux-Tools

Thomas Groß, teegee, info@teegee.de

Durch die Kombination von Technologien wie LVM, DRBD, Pacemaker und iSCSI kann ein hochverfügbares Speichercluster aufgebaut werden. Die LAX-Cluster-Verwaltungswerkzeuge erlauben es, Volumes zu erzeugen, zu verändern, zu sichern und rückzusichern. Das Speichercluster kann beliebig viele iSCSI Targets bereitstellen.

Weitere Informationen: http://cluster.teegee.de/

3.19 Einführung in BigData (Hadoop/Pig/Hive)

Peter Dickten, dcs – die Entwickler, mail@dcs-fuerth.de

Es wird eine grundlegende Einführung in BigData gegeben. Im Rahmen des Vortrags werden vorgestellt: Apache Hadoop, Apache Pig, Apache Hive. Es wird live gezeigt, wie man diese Tools verwendet, um große Datenmengen auszuwerten.

3.20 Einführung in das Dateisystem BTRFS

Sebastian Andres, info@andresedv.de

Das Dateisystem BTRFS wird immer stabiler. Es ist fester Bestandteil des Linux-Kernels. In diesem Vortrag wird auf ein paar Unterschiede zum Ext4-Dateisystem eingegangen und ein existierendes Ext4-Dateisystem nach BTRFS konvertiert. Der Beitrag legt den Fokus auf die Konvertierung und richtet sich ausdrücklich an Einsteiger!

Grundkenntnisse von Partitionierung, Live-Systemen und der Konsole sind von Vorteil.

3.21 Einführung ins Cross Media Publishing

Markus Wirtz, Open Source Press GmbH, info@opensourcepress.de

Die Veröffentlichung von Texten ist ein komplexer Vorgang: In diesem Vortrag geht es um technische und organisatorische Grundlagen, um mit AsciiDoc, DocBook, XSL & Co. aus einer Textvorlage (möglichst) alle gewünschten Ausgabeformate zu generieren.

3.22 Einsatz von Postgres im Bereich der Verkehrstelematik

Mario Krumnow, TU Dresden

In der Verkehrstelematik ist man oft mit einer Vielzahl unterschiedlicher verkehrstechnischer Messgrößen konfrontiert, z.B. Daten von Verkehrssensoren, Lichtsignalanlagen oder Verkehrsbeeinflussungsanlagen. In diesem Vortrag werden verschiedene Möglichkeiten aufgezeigt, an welchen Stellen PostgreSQL und Erweiterungen wie PostGIS genutzt werden können. Angefangen bei der Datenarchivierung, über die Datenanalyse bis hin zur Visualisierung bietet PostgreSQL ideale Voraussetzungen für den effektiven Einsatz in Verkehrsmanagementsystemen.

3.23 Embedded GNU/Linux mit Gnublin

Benedikt Sauter, embedded projects GmbH, sauter@embedded-projects.net

Das Projekt GNUBLIN entstand gemeinsam mit der Hochschule Augsburg, um Studenten eine Ausbildungsplattform für embedded GNU/Linux anbieten zu können. Dank der Open-Source-Hardware, einem Wiki und einem Forum entstand eine Community.

Wo früher mühselig Assembler und C geschrieben wurde, kann man dank der Leistung moderner Mikrocontroller heute Linux einsetzen. Es läuft auf gerade 2x5 cm^2 in der kleinsten Version und bietet allen Komfort eines modernen Servers an. Zu den vielen Anwendungen kommt hinzu, dass man einfach auf externe Hardware über die Board-Schnittstellen zugreifen kann.

Weitere Informationen: http://www.gnublin.org

3.24 Fast alles ist Text. Datenformate unter UNIX/Linux.

Frank Hofmann, Hofmann EDV – Linux, Layout und Satz, frank.hofmann@efho.de

Auf anderen Betriebssystemen hat jedes Programm sein eigenes binäres Datenformat, dass Sie nur mit diesem Werkzeug lesen und schreiben können. Unter UNIX/Linux ist vieles einfach nur Text mit einer bestimmten Struktur. Das erleichtert Ihnen das Arbeiten ungemein. Wir stellen Ihnen die Datenformate vor, mit denen Sie durch den Linux-Alltag kommen.

3.25 Forth überall

Carsten Strotmann, carsten@strotmann.de

Wäre es nicht schön, wenn es für alle Rechner, vom Einplatinenrechner bis zum Großrechner, eine Programmiersprache gäbe? Und wenn diese Sprache nie veraltet, sie «mitwächst» mit den Bedürfnissen des Entwicklers und neuen Paradigmen der Softwareentwicklung? Und wenn diese Sprache im Browser funktioniert, auf dem Mobiltelefon, und sogar als Netzwerk-Protokoll zu benutzen ist? Und wenn diese Sprache als Maschinensprache direkt in Hardware ausgeführt werden kann, und damit massiv parallele CPUs mit 144 Kernen auf einer Fläche von 1x1cm² möglich sind?

Neugierig geworden?
Weitere Informationen: http://forth-ev.de

3.26 Freiheit leben für Anfänger – vier Freiheiten der Freien Software als Meilensteine auf dem Linux-Weg

Jana Wisniowska, linux@janapirat.de

Ausgehend von den vier Freiheiten der Freien Software möchte ich den Weg eines Einsteigers in die Linux-Welt erläutern und jeden dazu ermutigen, diesen Weg zu gehen. In diesem Vortrag möchte ich zeigen, wie Einsteiger sich der unbekannten Linux-Welt nähern können, was dazu notwendig ist und wo sie Hilfe erhalten. Es werden nicht nur die vier Freiheiten erklärt, sondern auch praktische Hinweise gegeben, die einen erfolgreichen Einstieg erleichtern.

3.27 GROUP-E Collaboration Software – best practice aus Aue / Sachsen

Sandy Graupner, Bauer Tiefbauplanung GmbH
Harald Grzybowski, group-e@bytespring.de

GROUP-E ist eine webbasierte Groupware- und Collaboration-Lösung und 100 % Open Source (GPL). Features sind u.a. Projekt- und Dokumentenverwaltung, E-Mail-Messaging, Adressverwaltung, Zeiterfassung, Wiki, Integration in Telefonanlagen und SMS-Versand.

Die Projektzuordnung von Terminen, Aufgaben, Adressen, Dateien, E-Mails und Zeiteinträgen bietet eine effektive Lösung für ein standortunabhängiges Arbeiten in virtuellen Teams.

Gemeinsam mit der «Bauer Tiefbauplanung GmbH» aus Aue in Sachsen präsentiert BYTESPRING ein «best practice» über die Einführung dieser Lösung und die Abbildung betrieblicher Prozesse mit GROUP-E.
Weitere Informationen: http://www.group-e.info

3.28 Grundkonfiguration des Mailclients Mutt

Sebastian Andres, info@andresedv.de

Mutt ist ein kleiner, schlanker und schneller Mailclient. Aber wie wird er denn jetzt konfiguriert und benutzt? In diesem Einsteigervortrag geht es um die Grundkonfigu-

ration: Editor einstellen, Absender festlegen, Signatur einfügen, usw. Vorkenntnisse sind nicht erforderlich!

3.29 Host-based disk-mirroring with MD

Hannes Reinecke, SUSE Labs, hare@suse.de

Disk-mirroring is a standard feature of basically every OS today. However, these solutions cover built-in disks only. For remotely connected disks e.g. via FibreChannel or iSCSI the situation changes.

Normally every I/O error is being treated as an fatal error, requiring manual interaction to replace the disk. With remote disks transient I/O errors might occur, requiring a more careful I/O handling.

In this talk I will give an overview on the challenges when implementing such a solution and present the 'md_monitor' program for handling transient I/O errors on MD mirror setups.

Weitere Informationen: git://github.com/hreinecke/md_monitor

3.30 I2P, Tor, Freenet, GNUnet, Retroshare – Kurzvorstellung der Techniken

Lars Schimmer, TU Graz, CGV

Dieser Vortrag soll kurz die grundsätzlichen Techniken diverser Software vorstellen, die den Datenverkehr der User anonymisieren. Auch die Grenzen der Technik werden diskutiert.

3.31 Ich weiß, was du letzte Woche getan hast! Seitenkanäle in der Gebäudeautomation.

Steffen Wendzel, Hochschule Augsburg / FernUniversität in Hagen

Dieser Vortrag führt in die Thematik der Gebäudeautomation ein und beleuchtet anschließend Seitenkanäle und verdeckte Kanäle in der Gebäudeautomation. Besprochen werden soll dabei unter anderem die Überwachung von Bewohnern und die Datenexfiltration über das Gebäude-Netzwerk.

Dieser Vortrag wird durch den eBusiness-Lotsen Schwaben unterstützt.

3.32 ICINGA – Open Source Monitoring Today and Tomorrow

Bernd Erk, ICINGA – Open Source Monitoring, info@icinga.org

Icinga is an enterprise grade open source monitoring system and Nagios fork with scalable and extensible monitoring, notification and reporting.

This talk will introduce Icinga's technical foundations and share our latest technology preview, Icinga 2 – a Nagios core rewrite. It will explain how Icinga 2's new component

loader design and check execution model will make it more efficient, scalable and easier to configure in large environments.

We'll demonstrate the new web interface with integrated reports and signpost future plans for the project.

Weitere Informationen: http://www.icinga.org

3.33 Integration: Fluch oder Segen – integrierst du noch oder arbeitest du schon?

Georg Schütz, KMUX-Projekt, KaMUX GmbH & Co. KG, georg.schuetz@kamux.de

«Integration» ist ein Zauberwort. Damit werden uns die vielfältigsten Dinge angeboten. Software, die unsere Unternehmensprozesse «integriert abbildet», Cloud-Plattformen, mit denen wir unsere Daten mit anderen «nutzen und integrieren» können, mobile Geräte, die sich überall «integrieren» lassen usw. Einzelne Softwareprodukte werden komplexer, weil immer mehr Funktionen «integriert» werden. Das gilt für proprietäre wie für Freie Software. In meinem Vortrag möchte ich einen Beitrag wider die Komplexität leisten. Einfach ist einfach einfacher und Offenheit besser für die Zusammenarbeit.

3.34 IPv6 – Migration, Betrieb und Deployment

Markus Schade, Hetzner Online AG

Der Vortrag widmet sich den Herausforderungen beim Deployment von IPv6 allgemein und im Speziellen in den Rechenzentren der Hetzner Online AG. Hierbei geht es von der Strukturierung des Präfix über den ersten Testbetrieb und die Verfügbarkeit für Kunden bis zur Umstellung aller Dienste auf Dual-Stack und der standardmäßigen Auslieferung von Systemen mit vorkonfiguriertem IPv6.

Dabei geht der Vortrag auch auf Möglichkeiten zur automatische Installation von Systemen ein, deren PXE-Implementierung keine IPv6-Unterstützung enthält. Ebenso wird ein Ausblick auf zukünftige Veränderungen durch die Einführung von UEFI gegeben.

3.35 IT-Management mit Open Source Tools

Alexander Heidenreich, GK Software AG, clt2013@gk-software.com

Systemadministratoren stehen über kurz oder lang vor der Frage, wie sie ihre ständig wachsende Umgebung nicht nur im Griff behalten, sondern auch für zukünftige Entscheidungen die nötigen Informationen sammeln können. Asset- und Softwaremanagement heißt die Antwort, und wir stellen freie Tools vor, die diese Aufgabe erledigen, ohne dabei kommerziellen Produkten nachzustehen.

3.36 KDE-Grundeinstellungen: Desktophintergrund, Plasmoids und mehr

Monika Eggers, Kubuntu, monika@kubuntu-de.org

Wie ändert man den Desktophintergrund? Was ist ein Plasmoid (Miniprogramm), und wie bekommt man es in den Desktop oder die Kontrollleiste? Welche beliebten Miniprogramme gibt es? Wie kann man ändern, was passiert, wenn man den Mauszeiger in eine Ecke bewegt? Wie kann man die Fensterleiste anpassen? Wie legt man seine Lieblingsprogramme in die Kontrollleiste? Wie ändert man die Knöpfe in der Titelleiste der Fenster? Wie stellt man lustige Fenstereffekte wie Wobbeln, Drehwürfel oder Schnee ein, und wie wird man sie wieder los? Solche Fragen stellt sich der KDE-Neuling, und hier gibt es die Antworten.

3.37 LibreOffice im Unternehmenseinsatz

Jacqueline Rahemipour, LibreOffice, jrahemipour@documentfoundation.org

Die Argumente «Open Source» und «kostenlos» sind nicht alles. Damit LibreOffice den Einzug in Unternehmen schafft, gehört ein wenig mehr dazu: Hohe Kompatibilität und ein auf Unternehmensbedürfnisse abgestimmter Funktionsumfang sind die Grundvoraussetzungen für einen erfolgreichen Einsatz von LibreOffice im professionellen Umfeld. Dabei muss sich die freie Office-Suite fast immer mit dem Marktführer Microsoft Office messen lassen. Dieser Vortrag durchleuchtet die Chancen und Risiken eines Umstiegs, zeigt mögliche Stolperstellen auf und erörtert, wie man diese umgehen kann.

3.38 Linux auf ARM-Servern

André Przywara, Linaro

ARM-Prozessoren sind bisher hauptsächlich aus mobilen oder eingebetteten Geräten bekannt. Seit kurzem gibt es aber auch Server mit ARM-Technik. Im Vortrag werden die Unterschiede zu x86-Servern vorgestellt und der Aufbau eines Systems beschrieben. Dann werden die Unterschiede auf der Software-Seite etwas näher beleuchtet, z.B. beim Booten oder bei der Linux-Kernel-Erstellung und Konfiguration. Abschließend gibt es noch ein paar praktische Erfahrungen und einen kleinen Ausblick in die Zukunft.

3.39 Linux Essentials – Einsteigerzertifizierung des LPI

Michael Gisbers, Linux-Schmie.de, michael@gisbers.de

Die Linux-Essentials-Prüfung des Linux Professional Institutes ist nicht nur eine Erweiterung des Portfolios, sondern ein Einstieg in die Welt der LPI-Prüfungen.

Als Basisprüfung soll sie gerade jungen Menschen und Umsteigern den Eintritt und den ersten Kontakt mit Zertifizierungen erleichtern. In Zusammenarbeit mit dem LPI Inc., vielen Lehrern von Berufsschulen und Gymnasien sowie zahlreichen ehrenamtlichen Helfern wurde ein Anforderungskatalog erstellt, der rein auf das Grundlagenwissen der verschiedenen Distributionen abzielt.

Weitere Informationen: http://tinyuri.de/9l-1s

3.40 Linux im Büro

Frank Richter, Esda Feinstrumpffabrik GmbH, clt2009@esda.com
Daniel Schreiber, Esda Feinstrumpffabrik

Linux auf dem Desktop ist an Firmenarbeitsplätzen noch ein Exot. Die Vortragenden betreuen seit mehr als fünf Jahren über 100 Anwender, die einen Linux-Desktop benutzen. Der Vortrag berichtet davon, wie es geht, wie man das (mit wenig Personal) macht und welche Fallstricke lauern.

Folgende Themen werden angesprochen: zentrale Vorkonfiguration wichtiger Anwendungen, Benutzerverwaltung über mehrere Standorte hinweg und Verwalten von Benutzern, die an mehreren Standorten arbeiten.

3.41 Linux-Dienstleister stellen sich vor

Bei den Chemnitzer Linux-Tagen sind Unternehmen seit Jahren ein fester Bestandteil. Sie entsenden ihre Mitarbeiter, um Vorträge zu halten, sie präsentieren sich bei Linux-Live und unterstützen uns als Sponsoren. Auch unter den Gästen sind viele, die beruflich mit Freier Software arbeiten. Umrahmt von speziellen Business-Vorträgen bieten wir hier eine Stunde lang die Möglichkeit, dass sich Unternehmen in wenigen Worten vorstellen, ihre Wünsche für gemeinsame Projekte äußern und Arbeitsmöglichkeiten mit sowie in verschiedenen Branchen zeigen können.

Weitere Informationen: http://chemnitzer.linux-tage.de/2013/addons/business-forum

3.42 Managing larger projects with TaskJuggler

Chris Schlaeger

This talk will provide an introduction into managing larger projects with the TaskJuggler project management software. Key topics will be resource organization, composition of project portfolios and project and status tracking. The necessary features in TaskJuggler will be explained in detail and some procedures will be demonstrated live.

Weitere Informationen: http://www.taskjuggler.org

3.43 Mercurial: Experimente für Fortgeschrittene

Markus Zapke-Gründemann, inqbus, markus@keimlink.de

Dieser Vortrag zeigt die vielfältigen Einsatzmöglichkeiten von Mercurial abseits der häufig genutzten Funktionen.

Themen des Vortrags sind unter anderem: Cherry Picking, Bookmarks, Patches, Subrepositories, Konvertieren und Zusammenführen von Repositories, Umgang mit großen Dateien sowie das Veröffentlichen von Repositories.

Weitere Informationen: http://mercurial.selenic.com/

3.44 Modellierung einer virtuellen DMZ

Thomas Groß, teegee, info@teegee.de

Eine Demilitarisierte Zone (DMZ) kann auch aus mehreren virtuellen Maschinen (VM) gebildet werden. Dazu ist es notwendig, Schnittstellen explizit an die VM zu übergeben und getrennte virtuelle Netze für DMZ und Intranet anzulegen. Das LAX-Clustermanagement unterstützt dies durch vorgefertigte virtuelle Maschinen, die leicht kombinierbar sind.

Weitere Informationen: http://cluster.teegee.de/

3.45 MySQL HA mit Galera

Erkan Yanar, erkan.yanar@linsenraum.de

Der Vortrag zeigt, wie mit dem Plugin Galera MySQL in ein Multi-Master Cluster verwandelt werden kann. Dieser repliziert die Daten schneller als die eingebaute semisynchrone Replikation von MySQL, nimmt im Gegensatz zur klassischen Replikation von MySQL auf allen Nodes des Clusters auch schreibende Anfragen entgegen und beherrscht sowohl rolling Upgrades als auch Restarts – bisher eine Domäne des MySQL NDB Clusters. In Verbindung mit einem Loadbalancer existiert eine HA-Lösung für MySQL. Einige Schattenseiten sollen im Vortrag ebenfalls erwähnt werden.

Weitere Informationen: http://codership.com

3.46 Nach dem ersten RPM gehts erst richtig los ...

Robert Scheck, Fedora Project, robert@fedoraproject.org

Mit dem ersten erfolgreichen selbst gebauten RPM-Paket ist das Ende der Fahnenstange noch lange nicht erreicht: Makros, Skriptlets, Unterpakete, verschiedene Hardware-Architekturen, unterschiedliche Versionen einer Linux-Distribution oder gar unterschiedliche Linux-Distributionen, Patches, Debug-Info, reproduzierbare Ergebnisse mit einem Buildsystem lassen auf sich warten. All dies wird im Vortrag angesprochen und teilweise auch live gezeigt und durchgespielt.

Der Vortrag richtet sich vor allem an technisch orientierte Besucher, die tiefer in den RPM-Paketbau einsteigen möchten und bereits Erfahrung mit einfachen RPM-Paketen haben.

3.47 Nebenläufige Programme mit Python

Stefan Schwarzer, SSchwarzer.com, sschwarzer@sschwarzer.com

In diesem Vortrag soll es vor allem um Grundlagen nebenläufiger Programme gehen, wie sie mit den Modulen threading und multiprocessing möglich sind.

Neben Einsatzszenarien für nebenläufige Programme werden grundsätzliche Probleme an Hand von Beispielen erläutert, die Verwendung von Locks und Queues erklärt, und es wird auf die Vermeidung von Deadlocks eingegangen. Weitere Aspekte sind Ereignissteuerung und transaktionaler Speicher.

Weitere Informationen: http://www.python.org

3.48 Netzwerksicherheit durch aktive Sicherheitsscans

Alexander Heidenreich, GK Software AG, clt2013@gk-software.com

Das eigene Netzwerk aus Sicht eines Angreifers zu sehen, um Sicherheitslücken aufzuspüren, ist eine essentielle Tätigkeit jedes Systemadministrators. Tools wie NMAP, Arachni und ähnliche unterstützen ihn dabei. Mit OpenVAS steht ein Tool zur Verfügung, das diese Tests automatisiert, übersichtliche Reports generiert und den Administrator bei weiteren Analysen unterstützt.

Weitere Informationen: http://www.openvas.org/

3.49 Neues aus der BSD-Welt

Julian Fagir

Während es zu Linux diverse News-Seiten und Berichte gibt, hört man von den großen BSDs (DragonFly BSD, FreeBSD/PC-BSD, NetBSD, OpenBSD) abseits von neuen Releases nur wenig. Dabei gibt es sehr viele Neuerungen, die aber nur wenig mediale Aufmerksamkeit bekommen und oft stillschweigend untergehen, obwohl sie für die Open Source Community durchaus relevant sind.

In diesem Vortrag will ich daher vorstellen, was sich im letzten Jahr bei BSD technisch und in der Community getan hat und wo die Reise hingehen soll.

Weitere Informationen: http://komkon2.de

3.50 Nutzung der aktuellen Version des Notensatzprogramms LilyPond

David Kastrup, LilyPond, dak@gnu.org

Im letzten Jahr ist die Entwicklung des Notensatzprogramms LilyPond deutlich vorangekommen. Ein wichtiger Schritt hierbei waren die im August bei einem Entwicklertreffen in Waltrop erfolgten Freigaben der stabilen Version 2.16 und der ersten Versionen des aktuellen Entwicklungszweiges 2.17.

Anhand von 2.16 wird die Eingabe und Bearbeitung von Noten mit Emacs und mit dem dedizierten Editor Frescobaldi vorgestellt. Ein Ausblick auf die kommende stabile Version 2.18 und auf Notenplattformen im Internet, die intern auf LilyPond zurückgreifen, vervollständigt den Vortrag.

Weitere Informationen: http://www.lilypond.org

3.51 Nutzung von PCs und Thin Clients für VDI- und Cloud-Anwendungen

Alexander Stecher, openthinclient gmbh, info@openthinclient.com

Haben Sie sich auch schon überlegt, wie Sie Ihre Administrationskosten in Ihrem Unternehmen senken und gleichzeitig sowohl die Verfügbarkeit als auch den Komfort für die Nutzer bei der täglichen Arbeit erhöhen können? Der erste Schritt ist, Anwendungen oder ganze Arbeitsplätze zu virtualisieren. Im nächsten Schritt werden

bestehende PCs oder ThinClients per PXE mit openthinclient gebootet – einem Betriebssystem auf Linux-Basis. Mit einer einfach zu bedienenden Management-Oberfläche stellen Sie zentral ein, welcher Client auf welche VDI- oder Cloud-Anwendung zugreifen darf.

Weitere Informationen: http://openthinclient.org

3.52 Open choice beim Dokumentenformat: Ist der Anwender überfordert?

Lothar K. Becker, .riess applications, lothar.becker@riess.de

Im Sinne des professionellen wie privaten Anwenders greift der Vortrag die Bedeutung nachhaltiger Wertschöpfung beim Dokumentenformat auf und thematisiert tägliche Herausforderungen für Unternehmen und Verwaltungen. Es wird zunächst der Status Quo mit Interoperabilitätsproblemen aufgezeigt. Danach werden denkbare Ansätze zur Zusammenarbeit, wie sie pragmatisch Interoperabilität fördern und damit die freie Wahl der Nutzung von Office-Suiten ermöglichen, diskutiert. Erfahrungen aus einer Vielzahl von professionellen Migrationen zu freien und proprietären Office-Suiten runden die Thematik ab.

Weitere Informationen: http://www.riess-applications.de

3.53 Open eCard

Andreas Kuckartz, Open eCard Team, a.kuckartz@ping.de

Das Open-eCard-Projekt und die darin entwickelte Open eCard App werden vorgestellt. Damit kann in Web- und Cloud-basierten Anwendungen leicht eine starke Authentisierung mit bekannten Chipkarten wie z.B. dem neuen Personalausweis (nPA) oder der elektronischen Gesundheitskarte realisiert werden.

Neben dem generellen Potential der Open eCard App für die Open-Source-Gemeinschaft (insbesondere als Alternative zur bisherigen «AusweisApp») wird auch auf innovative Aspekte, wie z.B. die mobile Nutzung des nPA, eingegangen.

Weitere Informationen: http://openecard.org

3.54 Open KnowledgeWorker – Wissen ist frei

Lars Fassmann, chemmedia AG, info@chemmedia.de

Unser Ziel ist es, die Erstellung von Lerninhalten so einfach wie möglich zu machen. Dafür haben wir die kostenfreie Plattform Open KnowledgeWorker ins Leben gerufen. Warum wir das tun? Unsere Produkte setzen auf Open-Source-Komponenten und offene Standards. Open KnowledgeWorker ermöglicht den freien Austausch von Wissen und ist speziell auf die Produktion von interaktiven E-Learning-Bausteinen ausgelegt. Jeder kann Bausteine unter Creative Commons beisteuern und Inhalte anderer in eigenen Kursen frei nutzen. Entwickler können sich über Web Services an der Entwicklung beteiligen. Join now!

Weitere Informationen: http://www.openknowledgeworker.com

3.55 Open Source und die Zukunft

Roman Geber, B1 Systems GmbH, geber@b1-systems.de

Open Source hat zweifellos die Welt verändert. Dabei ist die freie Verfügbarkeit über Quellcode nur die Spitze eines Eisbergs, dessen wahre Ausmaße wir noch nicht einmal annähernd abschätzen können. Das Copyleft-Prinzip hat längst seine Kinderstube in der Softwareentwicklung verlassen und greift nun auf Hardware, Wissenschaft, Medizin, Produktion und viele weitere Bereiche über. Kreativen Köpfen aus aller Welt eröffnen sich neue Horizonte, deren Erforschung unsere Zukunft nachhaltig beeinflussen wird. Die Brennpunkte dieser Entwicklung sind Thema dieses Vortrags.

3.56 Open Source für Klein(st)unternehmen – die vergessene Zielgruppe

Klaas Freitag, ownCloud GmbH, freitag@owncloud.com
Stefan Schäfer, FSP Computer & Netzwerke, stefan@invis-server.org

Der Vortrag geht auf die Probleme kleiner und kleinster Unternehmen mit IT ein. Berührungsängste, Kommunikationsprobleme und blanker Ärger sind an der Tagesordnung. Die Referenten versuchen, nicht ohne Humor, Fragen wie «Kann Open Source hier helfen?», «Benötigen Kleinstunternehmer wirklich Enterprise-Lösungen?», «Welche Sprache sprechen eigentlich IT-Dienstleister?» zu klären. Selbstverständlich werden mit den eigenen Projekten «invis-Server« und «Kraft» auch Lösungsansätze für Klein(st)unternehmen vorgestellt.

3.57 Open-Source-Grafiktreiber für eingebettete Systeme

Lucas Stach

Linux ist bei eingebetteten Systemen (inkl. Android) eine feste Größe. Viele der PC-Distributionen werden zukünftig auch für diese Geräte zur Verfügung stehen.

Anders als beim PC ist die Auswahl an verschiedenartiger Grafikhardware im Embedded-Bereich sehr groß, wobei viele Hersteller nur unzureichende Treiberpakete anbieten. Dies ist Herausforderung und Chance zugleich für die Open Source Community.

Der Vortrag wird einen Überblick geben über derzeit verfügbare Hardware, die Unterschiede bei der Entwicklung von freien Treibern im Vergleich zu PC-Systemen und dem derzeitigen Stand der Dinge.

3.58 Open-Source-Projekte einer breiten (Business-)Community verfügbar machen

Silvia Frank, Univention GmbH, frank@univention.de

Der Vortrag wird zeigen, wie Open-Source-Projekte ihre Lösungen einfach in die Linux-Enterprise-Distribution Univention Corporate Server integrieren können, um sie so einer breiten Anwenderschaft bekannt zu machen und zur Verfügung zu stellen.

Weitere Informationen: http://www.univention.de/produkte/ucs/ucs-komponenten/app-center/

3.59 openSUSE on ARM

Alexander Graf, SUSE, agraf@suse.de

There have been a lot of things going on in the openSUSE on ARM world during the last year. We have released a openSUSE 12.2 as semi-official release. We have improved our workflows. We have also improved board support by quite a bit.

This talk will show you what exactly changed, how easy it is to be part of this awesome project and what the next steps are.

Weitere Informationen: https://en.opensuse.org/Portal:ARM

3.60 Performancetuning für iSCSI

Stephan Seitz, Heinlein Support GmbH, s.seitz@heinlein-support.de

Am Beispiel von open-iscsi, multipathd und scst auf aktuellen Linux-Plattformen werden Beispiele und Denkanstöße gegeben, wie man iSCSI-Storages bestmöglich nutzt.

Anwendungsbezogen werden unterschiedliche RAID-Level ebenso wie Vorteile und Probleme in Verbindung mit HA-Szenarien besprochen. Etwas Mathematik zum Optimieren des IP-Stacks rundet den Vortrag ab.

3.61 Picosafe: Open Source USB-Stick für Sicherheitsanwendungen

Michael Hartmann, embedded projects GmbH, hartmann@embedded-projects.net

Picosafe Stick ist ein Embedded-Linux-Gerät im USB-Stick-Format. Alle Daten und Anwendungen auf dem System sind verschlüsselt: Bootloader, Kernel und Dateisystem. Erst durch Eingabe eines persönlichen Geheimnisses startet die Entschlüsselung.

Mit Hilfe einer modernen Hardwareverschlüsselung durch den Prozessor und eines internen sicheren Schlüsselspeichers für den Bootloader bietet Picosafe eine perfekte Basis für Sicherheitsanwendungen.

Auf dem Stick läuft ein Debian 6.0 mit einem Kernel 3.3. Die Entwicklungsumgebung ist frei verfügbar.

Weitere Informationen: http://www.picosafe.de/?page_id=3077

3.62 Plan 9, ein Rasberry Pi und Linux

Carsten Strotmann, carsten@strotmann.de

Das Betriebssystem Plan 9, erschaffen u.a. von den Unix-Erfindern Rob Pike, Ken Thompson, Dave Presotto und Phil Winterbottom unter Mitarbeit von Dennis Ritchie, perfektioniert die Unix-Grundidee: alles ist eine Datei. Mit Plan 9 lassen sich Systeme erstellen, die sich nahtlos über viele Rechner erstrecken. Seit 2012 ist es auch auf dem Minirechner «Rasberry Pi» verfügbar. Der Vortrag gibt eine Übersicht über Plan 9, erklärt, worin sich Plan 9 von Unix/Linux-Systemen unterscheidet und zeigt die Integration von Plan-9-Konzepten in Linux.

Weitere Informationen: http://plan9.bell-labs.com/plan9/

3.63 Portable Build-Systeme

Jörg Schilling, FOKUS Fraunhofer

Der Vortrag beschreibt ausgehend von den Anfängen portabler Build-Umgebungen (Larry Wall: Configure, X11: imake, ...) aktuelle Ansätze wie nmake (David Korn), Makefiles (Jörg Schilling), autoconf (GNU), cmake (kitware) mit ihren Vor- und Nachteilen sowie Gemeinsamkeiten. Am Ende werden die Features besprochen und diskutiert, die heute wünschenswert erscheinen, aber noch nicht implementiert wurden. Der Vortrag gibt auch Tipps für die Entwicklung eigener portabler Software sowie Hinweise zur Verwendung rekursiver Make-Aufrufe über verschiedene Directories eines Baumes.

3.64 POWERLINK + Open Source IP-Core => FPGA

Daniel Krüger, SYS TEC electronic GmbH, info@systec-electronic.com

POWERLINK ist ein Kommunikationssystem zur Echtzeit-Datenübertragung über Ethernet. Mit der Open-Source-Implementierung openPOWERLINK ist neben einer rein softwarebasierten auch eine FPGA-basierte Lösung unter BSD-Lizenz verfügbar. Dieser Vortrag gibt eine kurze Einführung in das Kommunikationsprotokoll. Der Fokus liegt allerdings auf dem Aufbau und der Funktionsweise der Slave-Implementierung in einem Altera Cyclone III FPGA. Dabei werden auch die eingesetzten Entwicklungswerkzeuge vorgestellt. Natürlich wird eine Demonstration nicht fehlen.

Weitere Informationen: http://openpowerlink.sourceforge.net

3.65 Rechtsfragen für Postmaster und Unternehmen

Peer Heinlein, Heinlein Support GmbH, mail@heinlein-support.de

Der rechtssichere Umgang mit E-Mails, Spam- und Virenschutz wirft seit jeher für den Administrator zahlreiche Fragen und Probleme auf, die selten gelöst werden. Doch sobald die private Nutzung der E-Mails durch Mitarbeiter ins Spiel kommt, wird es endgültig kompliziert und unübersichtlich.

Anforderungen an Unternehmen und Administratoren stehen dabei im krassen Widerspruch zu datenschutz- oder telekommunikationsrechtlichen Vorschriften oder der gelebten Praxis in den Unternehmen. Konflikte sind vorprogrammiert – und zwar in jeder Hinsicht.

Weitere Informationen: http://www.heinlein-support.de/vortrag/rechtsfragen-fuer-administratoren-und-unternehmen

3.66 Replikation mit PostgreSQL

Andreas Kretschmer
Andreas Scherbaum, EMC/Greenplum, ads@pgug.de

Die Replikation mit PostgreSQL wurde in den letzten Versionen stetig verbessert. Dieser Vortrag gibt einen Überblick über die aktuelle Implementierung und zeigt anhand praktischer Beispiele, wie schnell sich eine Replikation aufsetzen lässt.

3.67 Sieben Mythen über Open Source – Was wir schon immer mal richtig (durch-) denken wollten ...

Karsten Reincke, Deutsche Telekom AG

Der Vortrag analysiert sieben gern verwendete Open-Source-«Mythen». Speziell geht es um Herausforderungen, die sich aus der geschäftlichen Nutzung von Open-Source-lizenzierter Software ergeben. Was man aus privater Sicht noch gelassener handhaben mag, kann aus Geschäftssicht dramatisch werden: Wenn man nicht tut, was die Lizenzen fordern, verliert man die Nutzungsrechte.

Thema des Vortrags ist daher auch das «Open Source License Compendium», das die Telekom – durchaus als Giving Back – unter einer Creative-Commons-Lizenz veröffentlicht, um sich und anderen die Einhaltung der Lizenzen zu erleichtern.

Weitere Informationen: http://www.oslic.org

3.68 SSH und unzuverlässige Leitungen

Axel Beckert, ETH Zürich, abe@debian.org

SSH ist beliebt und bekannt, kann aber Nerven kosten, wenn das WLAN wackelig ist, das GSM oder SkyDSL laggt, man den Laptop mal kurz schlafen legt oder in ein anderes Netzwerk und damit die IP wechselt.

Der Vortrag stellt AutoSSH und Mosh vor, zwei Tools, die das Problem von ganz unterschiedlichen Seiten aus angehen.

Weitere Informationen: http://noone.org/talks/ssh-tricks/#mosh

3.69 Systemadministration in einer agilen Welt

Ralph Angenendt, ImmobilienScout24, ralph+chemnitz@strg-alt-entf.org

Dauerte früher die Entwicklung einer Applikation mehrere Monate bis Jahre, bevor sie in Betrieb gehen konnte, befinden wir uns heute in einer agilen Welt. Selbständige Entwicklerteams entwickeln Software in kurzen Zyklen und stellen diese Änderungen dann direkt «Live». Daran muss sich die Systemadministration – Applikationsmanagement, Betrieb bzw. System Engineering – messen lassen: Sie muss Strategien entwickeln, um diese immer kürzer werdenden Fristen zu unterstützen.

Dieser Vortrag zeigt, wie Systemadministratoren in dieser Welt «überleben» können, indem sie ebenfalls agil agieren.

3.70 Technische Dokumentation sicher und schick mit zpub

Georg Schütz, KMUX-Projekt, KaMUX GmbH & Co. KG, georg.schuetz@kamux.de

Dokumentationen für technische Produkte, Anwendungen oder Studien-, Bachelor- oder Masterarbeiten sollten einfach, stabil und sicher erzeugt werden können. Dazu gibt es viele Werkzeuge. In diesem Vortrag möchte ich zpub vorstellen, praktische Beispiele zeigen und so den Nutzen einer cleveren Kombination Freier Software demonstrieren.

3.71 Tippen statt klicken – die Welt der Linux-Kommandozeile

Holger Trapp, URZ der TU Chemnitz, hot@hrz.tu-chemnitz.de

Linux bietet neben modernen Desktops eine leistungsfähige Shell-Schnittstelle mit Unix-Werkzeugkasten, bei der man das System durch geschickt kombinierte textuelle Kommandos steuert. Sie wird wegen ihrer Mächtigkeit von erfahrenen Anwendern sowohl interaktiv als auch bei der Automatisierung von Abläufen durch Skripte rege genutzt, oft im Zusammenspiel mit der grafischen Oberfläche. Der Vortrag unternimmt einen Abstecher in die Welt der Befehlsschnittstelle und möchte anhand praktischer Beispiele ein Gefühl für deren Philosophie und sinnvolle Anwendungsgebiete vermitteln.

Weitere Informationen: http://www-user.tu-chemnitz.de/ hot/LT/2013/

3.72 Total Testing

Neil Langmead, Emenda GmbH, neil.langmead@emenda.eu

Emenda – Expertise in Software Testing. Defining and implementing a total testing platform. Twelve levels of testing and analysis for FOSS and COTS software. Integrated static source code analysis, security and architectural discovery. Sensible key performance indicators (KPIs) for software assets. Managing pitfalls of Open Source licencing, compliance and OSS component management.

3.73 Totalschaden: Ein gehackter Server auf dem OP-Tisch

Peer Heinlein, Heinlein Support GmbH, mail@heinlein-support.de

Eigentlich war es «nur» ein gehackter Root-Server wie jeder andere auch – aber gleichzeitig offenbart dieser Patient auf dem OP-Tisch der Intensivstation in seinen letzten Lebensminuten ein geradezu mustergültiges Beispiel über Backdoors, Rootkits und was man damit alles anstellen kann.

Dieser Vortrag zeigt mit einem Protokoll der Schritt-für-Schritt-Untersuchung eines real existierenden Vorfalls, woran man Einbrüche und Rootkits erkennt, wie diese funktionieren und welche Hinweise auf Manipulationen man nicht übersehen sollte.

Weitere Informationen: http://www.heinlein-support.de

3.74 Transparenz und Transformation: auf dem Weg in die offene Gesellschaft

Julia Bonk, Sächsischer Landtag, Fraktion DIE LINKE, julia.bonk@slt.sachsen.de

Am Ausgangspunkt wird ein Problemaufriss stehen. Ein wichtiger Schritt zur Ausschöpfung der Demokratisierungspotentiale besteht darin, die Schwelle zwischen Staat und Bürger/innen zu verändern. Gesetzlich bedeutet das, das Informationsrecht der Bürger/innen höher zu wichten als das Konzept des Amtsgeheimnisses. Hier soll ein Vorschlag aus der sächsischen Landespolitik vorgestellt werden.

3.75 Tupi – wie die Indios laufen lernen

Sirko Kemter, Fedora Project, gnokii@fedoraproject.org

Wenn man mit Linux 2D-Animationen erstellen möchte, gibt es nicht gerade viel Auswahl. Die meisten Open-Source-Projekte sind nur für Windows erhältlich oder wie Pencil und KToon als gescheitert anzusehen. Platzhirsch der 2D-Animationssoftware scheint SynfigStudio zu sein, welcher es aber dem Einsteiger nicht gerade leicht macht. Doch zu diesem Programm gibt es eine Alternative. Der Vortrag soll einen Überblick über die Situation der 2D-Animationssoftware geben, Tupi 2D Magic näher vorstellen und in einer kurzen Vorführung zeigen, wie kinderleicht es ist, mit Tupi Animationen zu erstellen.

Weitere Informationen: http://www.maefloresta.com/portal/

3.76 Tux als Bruchpilot – Vom Höhenflug ins Ministerium zur Bruchlandung im Klassenzimmer

Kurt Gramlich, kurt@skolelinux.de

Skolelinux ist eine Komplettlösung für Schulen und andere Bildungsträger. Das Projekt wurde in Norwegen 2001 gestartet und basiert auf der Linux-Distribution Debian. Hamburg bietet seit 2006 seinen Schulen Skolelinux als Alternative an. Rheinland-Pfalz hat 2008 entschieden, Skolelinux als Linux-Lösung für die Schulen des Landes zu verwenden. Der Vortrag berichtet über drei Jahre Erfahrungen mit der Entwicklung von Skolelinux in Rheinland-Pfalz aus der Sicht des ehrenamtlichen Projektleiters des deutschen Teams.

3.77 Vom Aussterben bedroht: Die Universalmaschine Computer

Matthias Kirschner, Free Software Foundation Europe, mk@fsfe.org

Computer sind universelle Maschinen, die beliebig programmierbar sind und prinzipiell alles können. Vielen IT-Unternehmen ist dies mittlerweile ein Dorn im Auge. Sie wollen willkürlich beschränken, was wir als Gesellschaft mit dieser Maschine machen können. Sie ergreifen technische Maßnahmen, mit denen sie uns Möglichkeiten nehmen und Rechte entziehen, die wir normalerweise haben, wenn wir ein Produkt kaufen. Die Industrie will entscheiden, was wir mit unseren Computern machen können, um ihren Profit zu maximieren. Wollen wir Ihnen diese Macht einräumen?

3.78 Wenn Geeks Langeweile haben

Uwe Berger, bergeruw@gmx.net

Jeder war bestimmt schon mal in dieser «misslichen» Lage: man sitzt gelangweilt zuhause, die Freunde, Freundin oder Frau haben keine Zeit, und im Fernseher läuft nur Müll. Also was tun, um die Zeit sinnvoll totzuschlagen? Innerhalb dieses Vortrags werden ein paar Hardware-Projekte des Autors vorgestellt, die zuerst meist nur zum Zeitvertreib begonnen wurden, teilweise allerdings auch ausarteten und fast immer in einem Know-How-Gewinn endeten. Ziel des Vortrags ist es, Anregungen für eigene Ideen und Experimente zu geben.

3.79 Zarafa als Exchange-Ersatz in der (private|public) Cloud

Michael Kromer, Zarafa Deutschland GmbH, m.kromer@zarafa.com

In diesem Vortrag wird ein Überblick von Zarafa im Zusammenhang mit dem Betrieb als Microsoft-Exchange-Ersatz in der private oder public Cloud vorgenommen. An Beispielen werden implementierte Umgebungen anhand von einem kleinen SMB-Hoster, einem mittelständischen Unternehmen sowie einem multinationalen Konzern aufgezeigt. Die dazugehörige Technologie und eingesetzte Infrastruktur sowie Best Practices im Einsatz werden erläutert. Speziell die neuen Funktionen in Zarafa 7.1 mit erweiterten Cluster-Fähigkeiten mit gleichzeitiger Skalierung stehen hierbei im Fokus.

Weitere Informationen: http://www.zarafa.com

3.80 Zur Funktionsweise von Schadsoftware

Stefan Schumacher, Magdeburger Institut für Sicherheitsforschung,
stefan.schumacher@sicherheitsforschung-magdeburg.de

Schadsoftware ist nicht auf kommerzielle und Closed-Source-Betriebssysteme beschränkt, schließlich wurde der Morris-Wurm 1988 auf 4BSD- und Sun-3-Systemen im Internet verbreitet. Der Vortrag zeigt daher die prinzipielle Funktionsweise von Schadsoftware am Beispiel des Staatstrojaners. Dazu werden Grundlagen der Computer- und Betriebssystemarchitektur vorgestellt. Es wird gezeigt, wie Schadsoftware diese zur Infektion und Replikation ausnutzen kann. Einige Gegenmaßnahmen werden ebenfalls kurz vorgestellt. Der Vortrag richtet sich an Einsteiger, die einige Grundlagen über Betriebssystem-Sicherheit kennenlernen möchten.

Weitere Informationen: http://www.sicherheitsforschung-magdeburg.de/buecher.html

3.81 Zutrittskontrolle im lokalen Netzwerk

Robert Sander, Heinlein Support GmbH, r.sander@heinlein-support.de

Dieser praxisorientierte Vortrag zeigt, wie die Zugangskontrolle in einem lokalen Netzwerk (LAN oder WLAN) mit Hilfe von FreeRADIUS und openLDAP realisiert werden kann. Es wird gezeigt, wie Netzwerk-Teilnehmer anhand von LDAP-Attributen bestimmten VLANs zugewiesen werden. Damit kann ein entsprechendes Sicherheitskonzept auf VLAN-Ebene einfach umgesetzt und z.B. ein Gast-Zugang für unbekannte Teilnehmer vom Firmennetz getrennt werden.

4 Zusammenfassungen der weiteren Workshops

4.1 3D-Visualisierung mit Blender

Stefan Müller
Erik Schufmann, ROGALL Architekten Ingenieure, info@erikschufmann.de

In diesem Workshop sollen den Teilnehmern nach einem Überblick über die Open-Source-3D-Suite Blender folgende Themen praktisch vermittelt werden: Filmausschnitte, Benutzeroberfläche, Modellierung (Object-/ Edit-Mode), Materialien, Beleuchtung, Rendern.

Weitere Informationen: http://www.blender.org

4.2 Datenintegration mit Talend Open Studio

Sandy Lucka, Avantgarde Labs GmbH, slucka@avantgarde-labs.de

Oft sind in Unternehmen Anwendungslandschaften historisch gewachsen, die nun Schritt für Schritt in moderne serviceorientierte Architekturen überführt werden. Talend übernimmt dabei die Rolle einer zentralen Datendrehscheibe, welche die verschiedensten Quell- und Zielsysteme miteinander verknüpft.

Melden Sie sich an und erfahren Sie, wie man Daten ohne Programmierkenntnisse extrahieren, transformieren und in ein Zielsystem laden und das Talend-Ökosystem inklusive Big-Data-Komponenten einsetzen kann.

An einem Datenintegrationsbeispiel werden grundlegende Talend-Funktionalitäten verdeutlicht.

Weitere Informationen: http://avantgarde-labs.de/de/tools/talend-open-studio

4.3 Die interaktive Programmier-Sprache «J»

Martin Neitzel, Gaertner Datensysteme, neitzel@marshlabs.gaertner.de

J ist eine Programmiersprache für parallele Datenverarbeitung, die sich ebenso im interaktiven «Taschenrechner-Modus» wie für größere Programm-Entwicklung bewährt. Die Implementierung ist als Open Source für Linux verfügbar.

Der Workshop beginnt mit einem kurzen Überblick über Geschichte und Charakter der Sprache, besteht dann aber in der Hauptsache aus einer Serie von praktischen Übungen mit einer Untermenge der Sprache. Bitte Laptops mit ssh und WLAN-Zugang mitbringen.

Weitere Informationen: http://gaertner.de/ neitzel/clt/j/

4.4 Django: Schnell performante Web-Applikationen entwickeln

Stephan Hoyer
Andreas Hug
Markus Zapke-Gründemann, inqbus, markus@keimlink.de

Ziel des Workshops ist die Erstellung einer Django-Applikation zur Verwaltung von Lesezeichen für beliebig viele Benutzer.

Wir beginnen mit einer kurzen Django-Einführung. Als erstes muss jeder Teilnehmer Python und Django auf seinem Rechner installieren. Nach dem Erstellen des Projektes werden zunächst die Objekte (Models) zum Verwalten der Daten angelegt, für die im weiteren Verlauf Templates und Views zur Darstellung der Inhalte im Browser erstellt werden. Dabei wird das in Django bereits enthaltene Admin-Backend vorgestellt und zum Anlegen und Bearbeiten der Datensätze genutzt.

Weitere Informationen: https://www.djangoproject.com/

4.5 Firewallkonfiguration mit Shorewall

Marek Walther, Marek-Walther@gmx.de

Shorewall ist ein Konfigurationswerkzeug zum Einrichten des Paketfilters IPtables. Es ermöglicht die Konfiguration von Paketfiltern mit einfachen Konfigurationsmodellen und allgemeinen Filterregeln. Es werden die Grundlagen einfacher Paketfilter und der Einsatz in verschiedenen Netzwerkmodellen erläutert und auf die Problematik von Paketfiltern bei verschiedenen Protokollen eingegangen. Der Einsatz transparenter Web-Proxys unter Shorewall und ein Einblick in den Themenbereich Traffic Shaping und multiple WAN bilden den Abschluss.

4.6 Inkscape – Sexy Jeans

Sirko Kemter, Fedora Project, gnokii@fedoraproject.org

Inkscape ist ein freier Vektorgrafikeditor, und dass man mit ihm mehr als ein paar schnöde Kreise zeichnen kann, zeigt dieser Workshop. Inkscape verwendet den SVG-Standard des W3C. In diesem Standard sind auch Filter definiert, mit denen man durchaus einfach auch photorealistisch zeichnen kann. Neben dem Umgang mit diesen Filtern wird auch die Arbeit mit den sogenannten LPE – Live Path Effects – gezeigt, mit denen man Pfade in Echtzeit verändern kann, denn irgendwie muss man mit Inkscape ja auch nähen können. Am Ende soll eine Jeanstextur und mit dieser eine Tasche entstehen.

Weitere Informationen: http://inkscape.org

4.7 LaTeX–Einsteigerworkshop

Doris Behrendt
Herbert Voß, Freie Universität Berlin, herbert@dante.de

LaTeX ist eine kostenlose Software, mit der man beliebige, meist PDF-Dokumente in Buchdruckqualität erstellen kann.

Teil 1: Installation einer LaTeX-Arbeitsumgebung auf dem mitgebrachten Laptop.

Teil 2: Erste Schritte: Erstellung einfacher Dokumente zum Mitmachen.

Teil 3: Themen nach Wunsch, z.B. Präsentationen, Mathematiksatz, Graphikeinbindung, Umgang mit Literaturdatenbanken, Zitaten, Querverweisen, Layout, Schriften.

4.8 LibreOffice für Umsteiger

Jacqueline Rahemipour, LibreOffice, jrahemipour@documentfoundation.org

Auf nahezu jedem Rechner gehört ein Office-Paket zum Standardumfang. Heutzutage können die meisten PC-Anwender auf Erfahrung z.B. mit Microsoft Office durch Schule, Studium oder Beruf zurückgreifen. Ist die freie Alternative LibreOffice für die typischen Office-Aufgaben gleichermaßen geeignet?

Dieser Workshop zeigt anhand praktischer Beispiele die Einsatzmöglichkeiten von LibreOffice und erläutert auch Unterschiede zu anderen gängigen Office-Suiten. Mitmachen auf dem eigenen Laptop ist ausdrücklich erwünscht!

4.9 OpenFOAM: Numerik mit Freier Software am Beispiel

Steffen Weise, TU Bergakademie Freiberg

OpenFOAM ist ein in C++ geschriebenes freies Softwarepaket zur Lösung von numerischen Problemen, bevorzugt aus dem Bereich der Strömungsmechanik. Nach der Erläuterung des Aufbaus der Software wird gezeigt, wie man Gitter für numerische Berechnungen erzeugt, Randbedingungen und Parameter für Simulationen festlegt und deren Ergebnisse auswertet und mit Paraview visualisiert.

4.10 Python: Programmieren macht Spaß

Thomas Güttler, TBZ-PARIV GmbH, guettli.clt2013@thomas-guettler.de
Stefan Schwarzer, SSchwarzer.com, sschwarzer@sschwarzer.com

Python ist eine objektorientierte Scriptsprache. Die einfache Syntax und umfangreiche Standardbibliothek ermöglichen eine schnelle Entwicklung von robuster Software.

Im Workshop werden in der ersten Hälfte die wichtigsten Grundlagen gezeigt. Anschließend versucht jeder Teilnehmer, ein Liste von Beispielen selbständig umzusetzen. Wir «springen» dann zwischen den Teilnehmern, die Fragen haben.

Weitere Informationen: http://www.thomas-guettler.de/vortraege/python/einfuehrung.html

4.11 Setup eines hochverfügbaren Virtualisierungsclusters

Thomas Groß, teegee, info@teegee.de

Der Workshop demonstriert das Setup eines hochverfügbaren 2-Knoten-Clusters von der Installation der Server bis zum Betrieb virtueller Maschinen. Nach einem Überblick über das LAX Clustermanagement werden virtuelle Maschinen (VM) aus Vorlagen installiert, Hochverfügbarkeit wird eingerichtet und getestet und die Sicherung

sowie Rücksicherung von VMs demonstriert. Zum Schluss wird die Clusterstruktur auf einen virtuellen Desktop abgebildet.

Weitere Informationen: http://cluster.teegee.de/

4.12 ZFS-Einsteigerworkshop

Benedict Reuschling, The FreeBSD Project, bcr@FreeBSD.org

Das von Sun (heute Oracle) entwickelte ZFS ist heute auf vielen Open-Source-Distributionen als Dateisystem verfügbar. Obwohl ZFS einfach zu administrieren ist, kann es trotzdem für Anfänger Fragen zum richtigen Einsatz geben. Im Workshop wird gezeigt, wie man die Features von ZFS nutzen kann. An praktischen Beispielen werden wir zusammen einen Storagepool aufbauen und verschiedene Konfigurationseinstellungen vornehmen, die je nach Einsatzzweck sinnvoll sind. Zusätzlich schauen wir uns an, was ZFS sowohl auf dem Desktop als auch im Serverbereich bereits heute schon bietet.

Personen

Albers, Erik, 111
Andres, Sebastian, 115, 117
Angenendt, Ralph, 128

Barth, Rico, 112
Becker, Lothar K., 124
Beckert, Axel, 112, 128
Behrendt, Doris, 133
Berger, Uwe, 130
Berger, Uwe, 31
Bonk, Julia, 129
Boogaart, Johan van den, 111
Breunung, Herbert, 113

Dickten, Peter, 115
Dittmann, Markus, 114

Eggers, Monika, 120
Erk, Bernd, 118

Fagir, Julian, 123
Fassmann, Lars, 124
Findeisen, Ralph, 77
Frank, Silvia, 125
Freitag, Klaas, 125

Gachet, Daniel, 103
Geber, Roman, 125
Geese, Elmar, 113
Gisbers, Michael, 120
Graf, Alexander, 126
Gramlich, Kurt, 130
Graupner, Sandy, 117
Groß, Thomas, 115, 122, 134
Großöhme, Peter, 111
Grzybowski, Harald, 117
Güttler, Thomas, 134

Hartmann, Michael, 126
Heidenreich, Alexander, 119, 123
Heik, Andreas, 41
Heinlein, Peer, 127, 129
Hofmann, Frank, 112, 116
Hoyer, Stephan, 133
Hug, Andreas, 133

Kastrup, David, 123
Kemter, Sirko, 130, 133
Kirschner, Matthias, 130
Klostermann, Christian, 114
Kölbel, Cornelius, 61
König, Harald, 49
Koppatz, Peter, 115
Kretschmer, Andreas, 127
Kromer, Michael, 131
Krüger, Daniel, 127
Krumnow, Mario, 116
Kubieziel, Jens, 85
Kuckartz, Andreas, 124

Langmead, Neil, 129
Leemhuis, Thorsten, 111
Loschwitz, Martin Gerhard, 69
Lucka, Sandy, 132
Luithardt, Wolfram, 103

Meier, Wilhelm, 13
Müller, Stefan, 132

Neitzel, Martin, 132

Pönisch, Jens, 21
Przywara, André, 120

Rahemipour, Jacqueline, 120, 134
Reincke, Karsten, 128
Reinecke, Hannes, 118
Reuschling, Benedict, 135
Richter, Frank, 121

Sander, Robert, 131
Sauter, Benedikt, 116

Schade, Markus, 119
Schäfer, Stefan, 125
Scheck, Robert, 122
Scherbaum, Andreas, 127
Schilling, Jörg, 127
Schimmer, Lars, 118
Schlaeger, Chris, 121
Schreiber, Alexander, 113
Schreiber, Daniel, 121
Schütz, Georg, 119, 128
Schufmann, Erik, 111, 132
Schuler, Jean-Roland, 103
Schumacher, Stefan, 131
Schwarzer, Stefan, 122, 134
Seidel, Philipp, 95
Seitz, Stephan, 126
Sirainen, Timo, 113
Stach, Lucas, 125
Stecher, Alexander, 123
Stein, Tobias, 115
Strotmann, Carsten, 117, 126

Tille, Andreas, 114
Trapp, Holger, 129

Voß, Herbert, 133

Wachtler, Axel, 77
Wälde, Erich, 112
Walther, Marek, 133
Weise, Steffen, 134
Wendzel, Steffen, 118
Werner, Tilo, 115
Winde, Thomas, 114
Wirtz, Markus, 116
Wisniowska, Jana, 117

Yanar, Erkan, 122

Zapke-Gründemann, Markus, 121, 133