FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Streamlining Extraction and Analysis of Android RAM Images

Simon Broenner

Lehrgebiet Datennetze, IT-Sicherheit und IT-Forensik

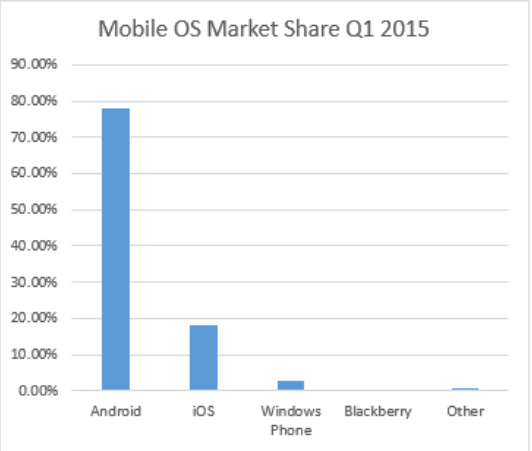
IT Forensik
FH Aachen

Gliederung

- Einleitung
- Stand der Technik
- Herausforderungen bei Android
- Abhilfe durch RAM-Extrahierung
- Analyse der Speicherabbilder
- VOLIX II als Analysetool
- Zusammenfassung

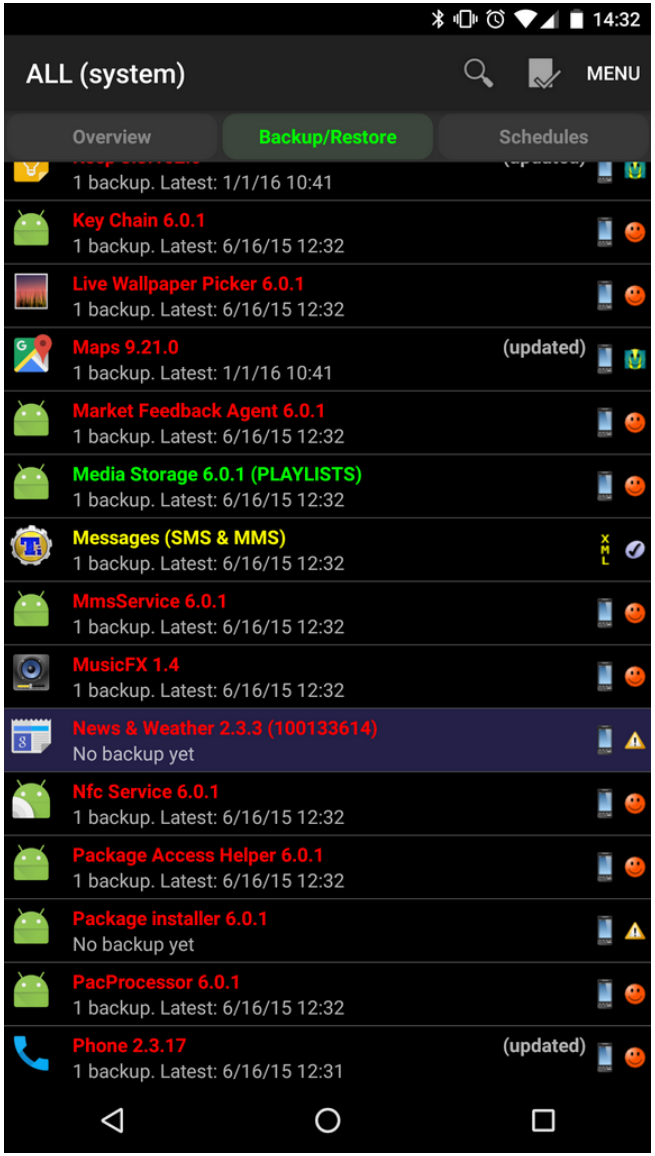
Android im Allgemeinen

- Das aktuell am weitesten verbreitete Betriebssystem
  - Nutzung als zusätzliches Gerät oder Ersatz
  - Günstig:
    - Neue Zielgruppe ohne eigenen Internetzugang
    - Alternativen bisher z.B. Internet Cafes
    - Besonders in Entwicklungsländern
  - Auf Linux-Basis
    - Java-Apps
      - Dalvik (JIT)
      - Android Runtime (ART)
    - Nativer Code mit NDK
- Forensische Untersuchbarkeit aufgrund Verbreitung extrem wichtig



Stand der Technik

- Nichtflüchtiger Speicher:
  - Oft einfach durch Lesezugriff auf nahezu gesamtes Dateisystem
  - Lediglich Schreibschutz auf Systempartition
  - Unverschlüsselte Backups
    - Nandroid-Archive
    - Titanium Backup o.Ä.



Stand der Technik

- Nichtflüchtiger Speicher:
  - Direktes Auslesen durch ADB Pull
  - Direktzugriff per USB
  - Apps wie “AFLogical”
  - Desktop-Tools wie “Andriller”

```
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
root@generic:/ # sur
/system/bin/sh: sur: not found
127|root@generic:/ # ls
acct
cache
charger
config
d
data
default.prop
dev
etc
file_contexts
fstab.goldfish
init
init.environ.rc
init.goldfish.rc
init.rc
init.trace.rc
init.usb.rc
init.zygote32.rc
mnt
proc
property_contexts
root
sbin
sdcard
seapp_contexts
selinux_version
sepolicy
service_contexts
storage
sys
system
ueventd.goldfish.rc
ueventd.rc
vendor
root@generic:/ #
```

Forensische Herausforderungen

Fragmentierung & Systemvielfalt

- Herstellergebunden Updates der Geräte
  - Starke Modifizierungen gegenüber “Stock” Android
  - Separate Kernelversionen
  - Anpassung an Hardware
- Langsame Veröffentlichung der modifizierten Kernel-Quellcodes

=> Oft neue Herangehensweise für jedes Gerät

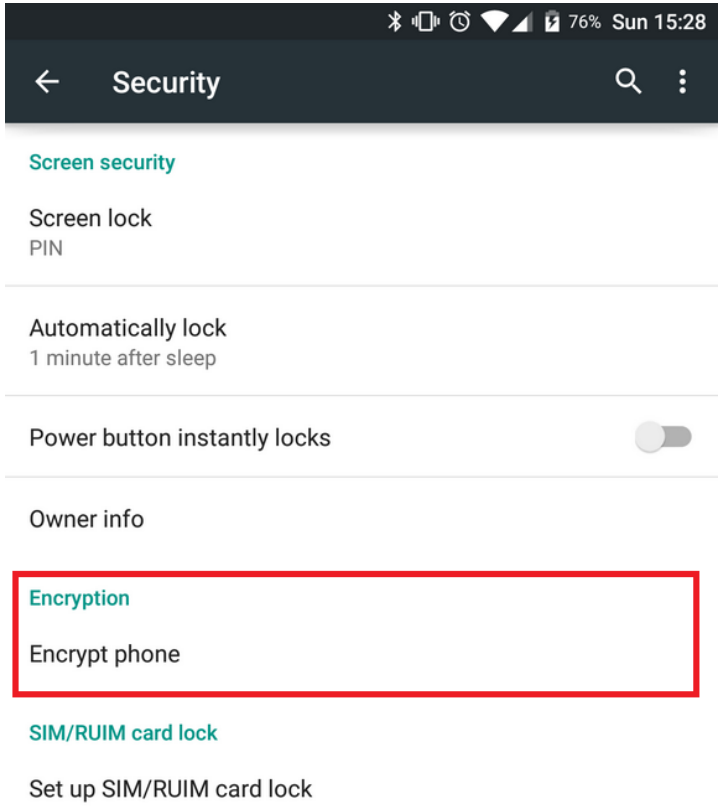
Forensische Herausforderungen

Eingebaute Verschlüsselung:

- Aktivierung in Systemeinstellungen
- Lediglich ein Klick notwendig

Manuelle Verschlüsselung:

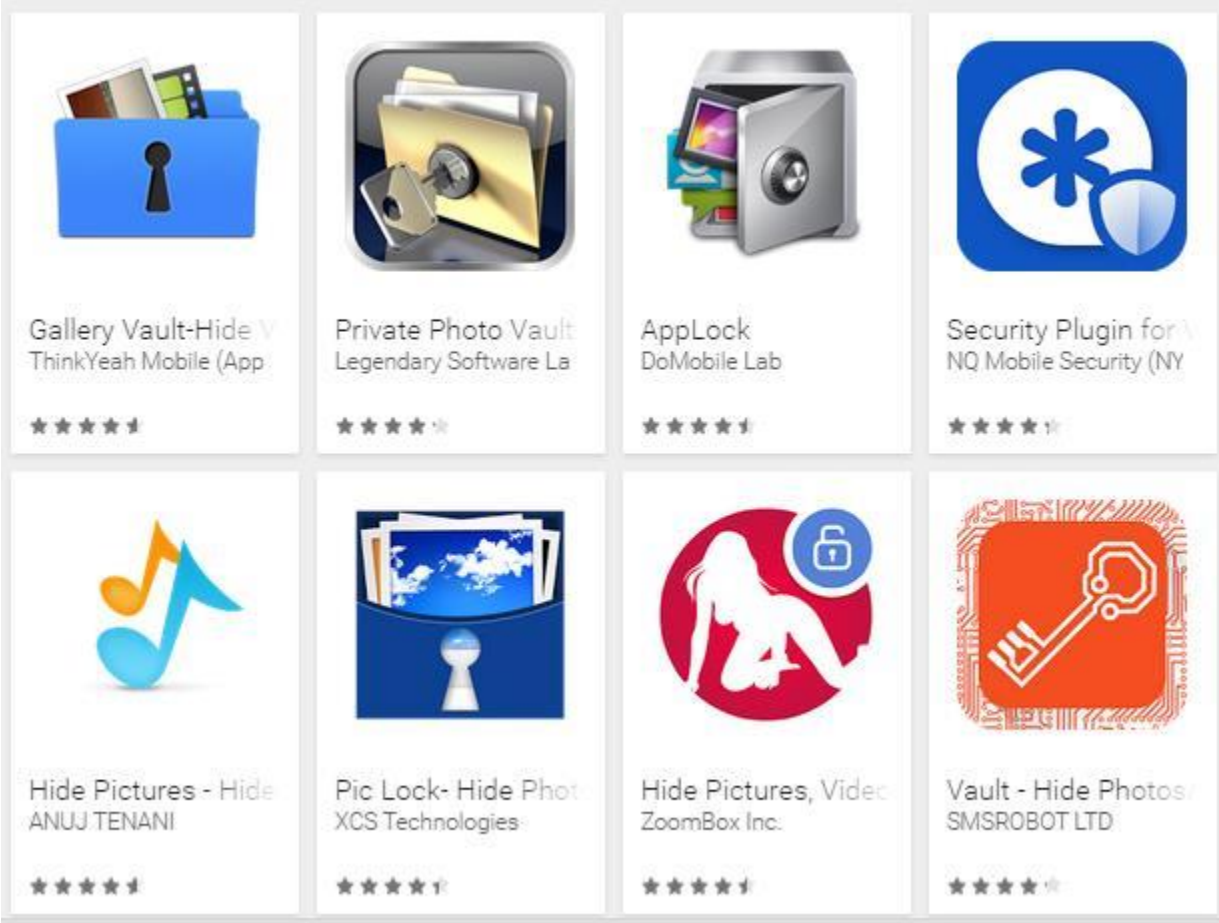
- z.B. LUKS



Forensische Herausforderungen

Locker/Vault Apps:

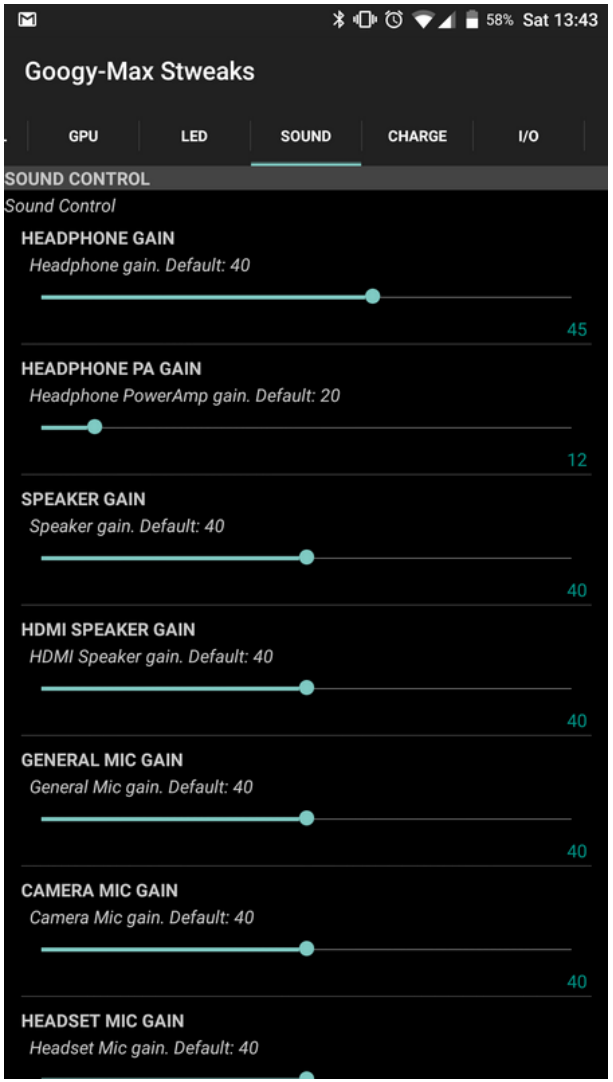
- Direkt im App-Store erhältlich
- Ohne technisches Wissen nutzbar



Forensische Herausforderungen

Malware:

- Android Sandbox schützt System vor Malware, verhindert aber auch effektives Scannen
- Bei Android besonders gefährlich wegen GPS, Kamera, Mikrofon etc.
  - Benutzer hat wenig Kontrolle über Zugriff



Abhilfe durch Auslesen des RAM

Verschlüsselung:

- Informationen zu:
  - Verschlüsselungstyp
  - Parameter
- Zugriff auf Rohdaten
- Temp. Dateien

Proof of concept z.B.:

- FROST (Müller, Spreitzenbarth, Freiling)
- LUKS recovery (Elatov)

Abhilfe durch Auslesen des RAM

Locker/Vault Apps:

- Oft sehr einfach gestrickt
  - z.B. versteckte Pfade im RAM im Klartext
  - Security through obscurity

Malware:

- Zugriff auf Daten aus laufenden Prozessen
  - Normalerweise durch Sandbox-System verhindert
  - Für in-System Malware-Scanner nicht möglich

Einsatzszenarien RAM-Extrahierung

Nutzer-Autorisiert:

- Vollzugriff mit Kennwörtern usw.
- Rootzugriff kann meist erlangt werden
- Kein Remote-Wipe Risiko

Beispiel-Vorgehensweise:

1. Rootzugriff erlangen
2. Custom Kernel kompilieren und booten
3. Kernelmodul kompilieren
4. Extrahieren

Einsatzszenarien RAM-Extrahierung

Ohne Autorisierung (z.B. Untersuchung durch Behörden)

- Gerät meist gesperrt
- Gerät hat möglicherweise schon:
  - Root-Zugriff
  - Custom ROM & Custom Recovery
- Gerät vlt. verschlüsselt

Zusatzschritte:

1. Internetzugriff deaktivieren
2. Mobilfunk deaktivieren
3. Sperrbildschirm aufheben

Extrahierung von Speicherabbildern

Ansatz:

- Kernel Modul zum Auslesen, z.B. *fmem, crash, LiME*
- Andere Zugriffsmöglichkeiten gesperrt
  - Z.B. */dev/mem* Zugriff bei früheren Kernels
- Injektion des Moduls in den laufenden Kernel mit *insmod*

Extrahierung von Speicherabbildern

Voraussetzungen:

- Root-Zugriff
- Kernel-Kompatibilität
  - Versionsnummern
  - Kerneloptionen

```
CONFIG_MODULES=y
CONFIG_MODULES_UNLOAD=y
CONFIG_MODULES_FORCE_UNLOAD=y
```

```
root@hammerhead:/sdcard # insmod lime.ko "path=/sdcard/limetest.dump format=lime"
nsmod lime.ko "path=/sdcard/limetest.dump format=lime"                          <
insmod: init_module 'lime.ko' failed (Function not implemented)
255|root@hammerhead:/sdcard #
```

Extrahierung von Speicherabbildern

- *Fmem* unter ARM problematisch da:
  - *page_is_RAM* zur Unterscheidung von RAM und Hardware Address-Space nicht implementiert
  - *dd* zum Auslesen genutzt:
    - bei Android in 32Bit implementiert (Overflow bei großen Speicheroffsets)
    - Forensische Datenintegrität nicht erhalten (Arbeitsspeicher wird während Extrahierung modifiziert)
- *Crash* nutzt auch *dd* zum Auslesen, gleiche Problematik

Extrahierung von Speicherabbildern

Verwendetes Kernelmodul: Linux Memory Extractor (LiME)

- Erprobt unter Android
- Kompatibel mit Volatility Framework zur weiteren Analyse
- Entwickelt von Joseph T. Sylve (University of New Orleans)

Extrahierung: Testgeräte

- Google AVD Emulator mit Android 5.1.1
- Galaxy Nexus (Maguro) mit Android 4.3
- Galaxy Nexus (Maguro) mit CyanogenMod 11
- Nexus 5 (Hammerhead) mit Android 5.1.1

LiME LKM Kompilierung

Voraussetzungen:

- Build-Environment passend konfiguriert inkl. Toolchain
- Kernel-Quellcode vorhanden
- LKM-Quellcode vorhanden

Kernel-Kompilierung

Voraussetzungen:

- Passendes Build-Environment (wie LKM)
- Möglichst erprobte Kernel Config
- Kernel-Quellcode
- Bei neueren Geräten ohne *zImage*-Unterstützung:
  - *mkbootimg* Utilities
  - *boot.img* aus laufendem Gerät zur Modifizierung

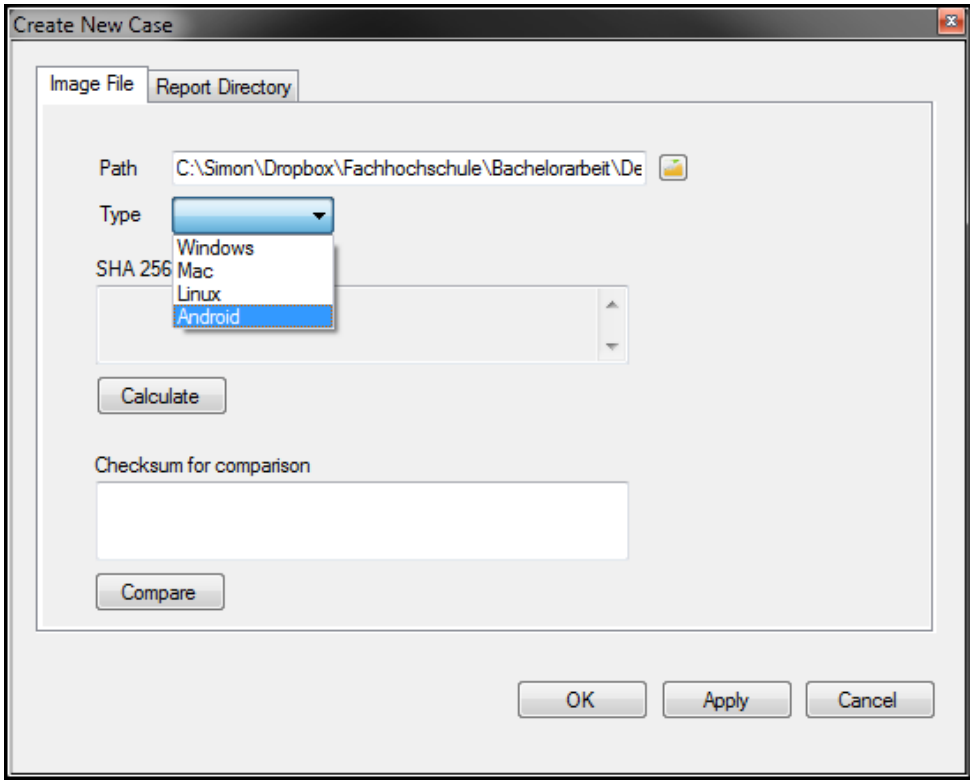
Extrahieren der Daten

1. Custom Kernel auf Gerät starten
2. LiME Modul lokal auf Gerät speichern
3. *Insmod* ausführen
4. Dump sichern

```
C:\Simon\Dropbox\Apps\android-sdk\platform-tools>adb shell
shell@hammerhead:/ $ su
su
root@hammerhead:/ # insmod /sdcard/lime.ko "path=/sdcard/hammerhead.dump format=
lime"
d /sdcard/lime.ko "path=/sdcard/hammerhead.dump format=lime"                    <
1|root@hammerhead:/ # ls -al /sdcard/hammerhead.dump
ls -al /sdcard/hammerhead.dump
-rw-rw---- root     sdcard_r 1982857280 2015-10-02 01:18 hammerhead.dump
root@hammerhead:/ #
```

VOLIX II: Neuerungen

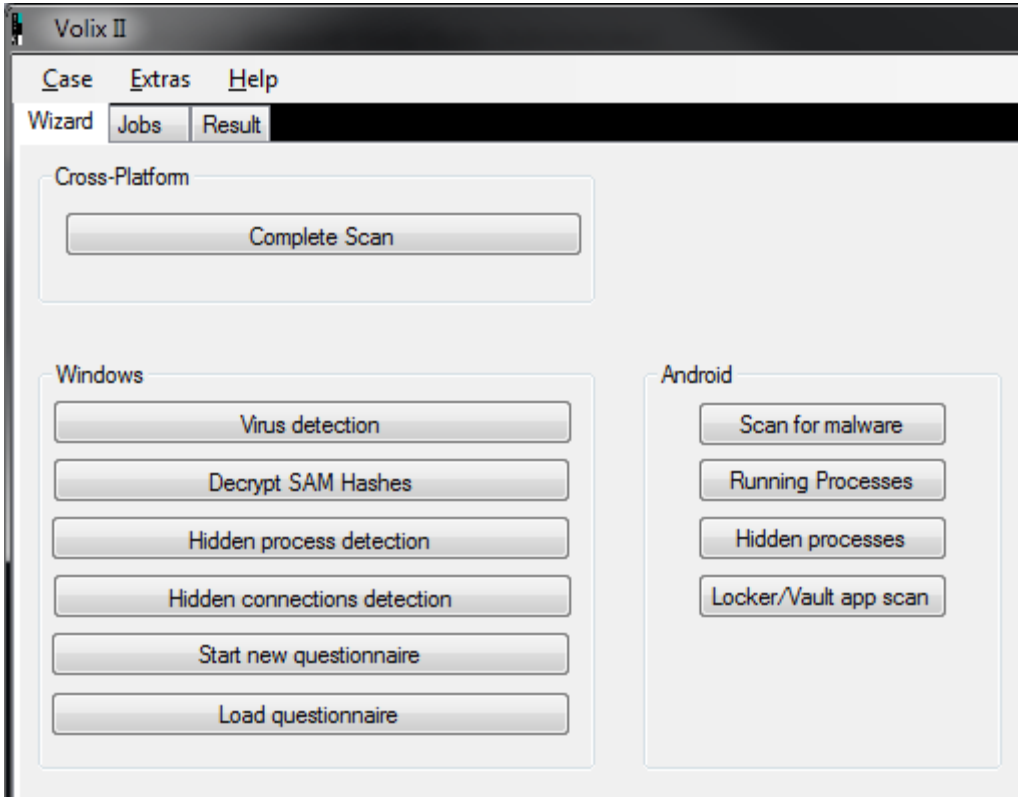
- Unterstützung für Volatility-Profile
  - Notwendig um überhaupt Non-Windows Speicher zu analysieren (ausgenommen manche OSX Verisonen)
  - Erweiterung der Linux-Befehle um *jobType* Property



VOLIX II: Neuerungen

Android Analyse-Routinen:

- Malware-Scan
- Running Processes
- Hidden Processes
- Locker/Vault app scan
- Complete Scan



Volatility Profile für Android

Voraussetzungen:

- Kernelstruktur-Beschreibungen (vTypes)
  - Im .dwarf Format mittels *Dwarfdump*
  - Generierbar durch Kompilierung eines C-Moduls gegen Teile des Kernel-Quellcodes
- Debug-Symbole (System.map)

```
volatility-standalone.exe --plugins=profileFolder --profile=profileName pluginName

python vol.py --profile=profileName pluginName
```

Zusammenfassung, Ausblick

- Extrahierung und Analyse extrem zeitaufwändig
  - Fragmentierung
  - Evtl. Sperren zu umgehen
  - Schlecht zu automatisieren
- Abhilfe durch Analyse-Repository
  - Volatility Profile
  - Lime-Module
  - Custom-Kernels
- Erweiterung von VOLIX II

Weitere Infos

Bachelor-Arbeit:

Dropbox-Link