FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Volix II, RAM Forensik mit dem Volatility Framework

René Wölker
Cand B.Sc. Scientific Programming
Woelker@fh-aachen.de

IT Forensik
FH Aachen

Agenda

1. Analyse von RAM-Dumps
2. Volatility Framework
3. Volix II Ausblick

Analyse von RAM-Dumps

1. Analyse von RAM-Dumps
2. Volatility Framework
3. Volix II

Analyse von RAM-Dumps

Abbildung der Rohdaten des Speichers in eine Datei.

- Entkopplung der Daten von dem Betriebssystem
- Minimierung der Änderung des zu untersuchendem System
- Reproduzierbarkeit der Ergebnisse aufgrund der Beständigkeit

Analyse von RAM-Dumps

Herkunft von RAM-Dumps:

- Live Response Analyse
  - DMA Exploit des Firewire-Anschlusses
- Hibernate Files
- Crash-Dumps

Analyse von RAM-Dumps

Der Arbeitsspeicher ist für sämtliche Daten und Anweisungen die Eingangshalle zu der CPU.

Beträchtliches Reservoir an Informationen:

- Registry Keys
- Netzverbindungen
- Prozesse und damit verbundene DLL‘s
- Threads und Handles
- Aufrufe der Konsole
- Usw.

Analyse von RAM-Dumps

- Vollständige Informationen über den aktuellen Zustand des zu untersuchenden Systems zur Zeit des Erstellen des Images
- Informationen über die Vergangenheit, sofern diese noch nicht überschrieben wurden
- Sowie Informationen über ausstehende Tasks

Analyse von RAM-Dumps

Diese Informationen sind so konzipiert:

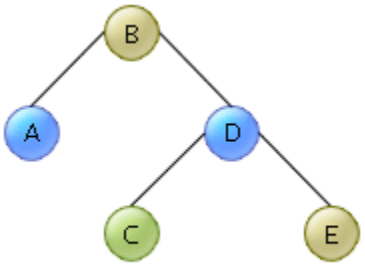
- Schnell
- Günstig
- Performant
- Und nicht forensisch leicht zu analysieren

⟹ Es gilt, diese Vielzahl an Informationen zu finden, in dem richtigen Kontext einzuordnen, sowie zu interpretieren.

RAM Dump – VAD Walk

Erste Idee:

- Ausnutzen der Strukturen des Betriebssystems
- Es existieren Softwarebreakpoints (Debugsymbole) mit Einstiegs-informationen
- Die Virtual Address Descriptors (VAD's) sind Strukturen um die Pages zu verwalten und Speicher zu (de-)allokieren
  - Alle VAD's sind als AVL-Baum angeordnet



(Quelle: http://me-lrt.de/binarbaum-avl-tree-baum-algorithmus-datenstruktur)

RAM Dump – VAD Walk (Fortsetzung)

- Jedes Prozess Objekt ist mit einem oder mehreren VAD's verbunden

- Die Prozesse verweisen auf:
  - Threads
  - Handles
    - Weitere Objekte

- Extraktion analog zu den Funktionen des OS

⟶ Gleiche eingeschränkte Sicht wie das OS

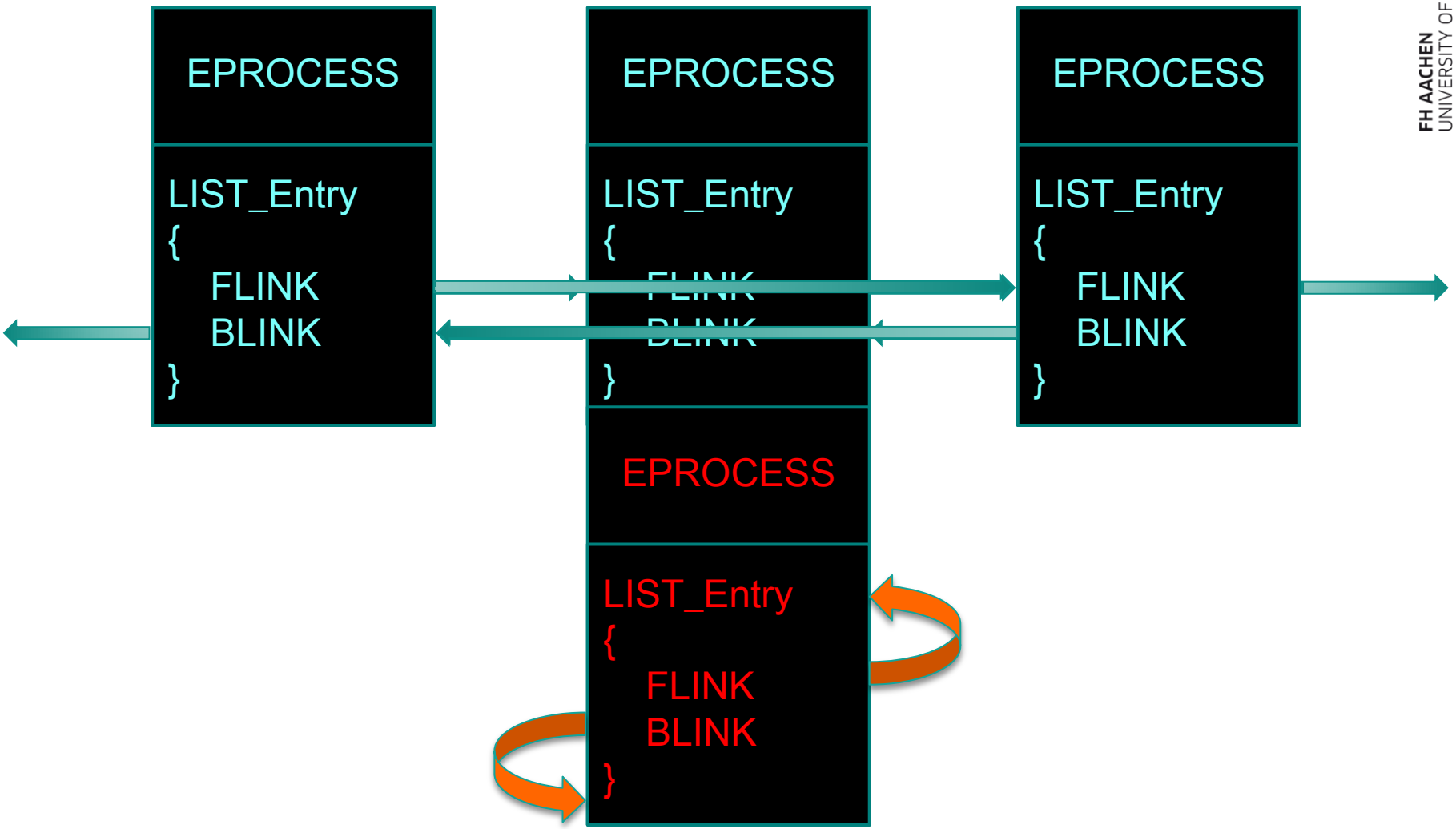
RAM Dump - DKOM

direct Kernel object manipulation

- Rootkits wie der FU Rootkit von Jamie Butler manipulieren die Verweise der Kernel Objekte.

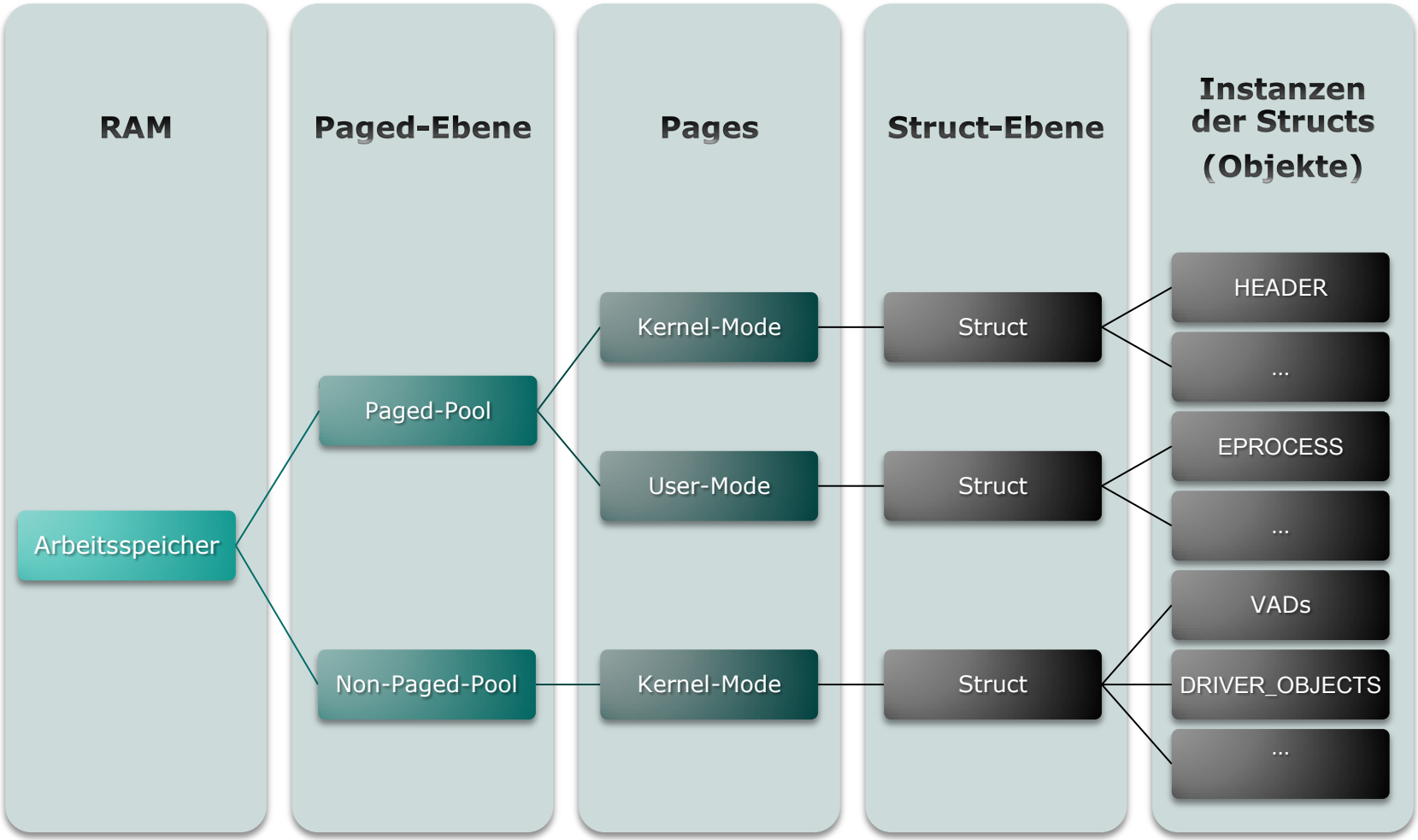
- Manipulierte EPROCESS Objekte werden von dem OS nicht mehr gesehen.

RAM Dump – DKOM



(Quelle: Eigene Darstellung in Anlehnung an Blunden, Bill. 2009. The Rootkit Arsenal. S. 423)

Ram Dump - Aufbau des Speichers



(Quelle: Eigene Darstellung)

RAM Dump – Signaturen Scan

- Alle Daten im Speicher sind opake Objekte
- Jedes dieser Kernel Objekte ist die Instanziierung von einem wohldefiniertem Struct
- Jedes dieser Structs hat einen bestimmten Header
- Zu jedem Header lässt sich eine Signatur bauen, die die Objekte eindeutig identifiziert

RAM Dump – Signaturen Scan

VadRoot	`val == 0 || (val & 0x80000000 == 0x80000000 && val % 0x8 == 0)`

(Quelle: Brendan Dolan-Gavitt et all. Robust Signatures for Kernel Data Structures)

RAM Dump – Signaturen Scan

⟶ Extrahieren der verschiedenen Objekte

- Vollständige Auflistung der Objekte
  - Inklusive der versteckten Objekte
  - Zusätzlich gelöschte Objekte
    - Das OS löscht die Pointer auf das Objekt und markiert den Platz als verfügbar, löscht diese aber nicht

Volatility Framework

1. Analyse von RAM-Dumps
2. Volatility Framework
3. Volix II Ausblick

Volatility Framework

Das Volatility Framework von Volatile Systems:

- in Python verfasst
- open-source
- Commandline-Tool

zur Analyse von Speicherauszügen.

Volatility Framework

Für die meisten Aufgaben existieren zwei Plug-Ins:

- Das eine Plug-In verfolgt die Referenzen zu diesem Objekt
  - Analog zu der Sicht des Betriebssystems
- Das andere Plug-In scannt nach den Signaturen

Volatility Framework

Vereinfachtes Anwendungsbeispiel

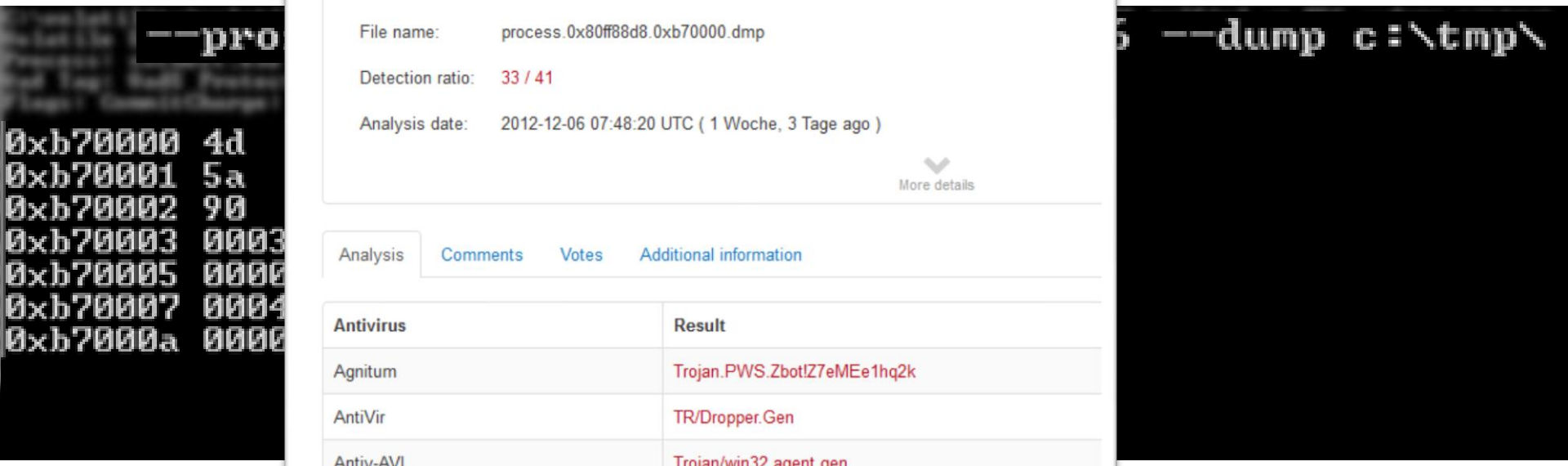
```
volatility-2.2.standalone.exe -f ramdump.vmem imageinfo

                      WinXPSP2x86
```

```
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
0x02214[illegible] 172.16.176.143:1054      193.104.41.75:80         856
0x06015[illegible] 0.0.0.0:1056             193.104.41.75:80         856
```

(Quelle: eigene Darstellung)

Volatility Framework



virustotal

SHA256:	8e3be5dc65aa35d68fd2aba1d3d9bf0f40d5118fe22eb2e6c97c8463bd1f1ba1
File name:	process.0x80ff88d8.0xb70000.dmp
Detection ratio:	33 / 41
Analysis date:	2012-12-06 07:48:20 UTC (1 Woche, 3 Tage ago)

More details

Analysis | Comments | Votes | Additional information

Antivirus	Result
Agnitum	Trojan.PWS.ZbotIZ7eMEe1hq2k
AntiVir	TR/Dropper.Gen
Antiy-AVL	Trojan/win32.agent.gen
Avast	Win32:Zbot-BCW [Trj]
AVG	Win32/Heri
BitDefender	Gen:Variant.Graftor.22830
ByteHero	-

(Quelle: eigene Darstellung)

Volatility Framework

Punkte der Software-Ergonomie :

- Aufgabenangemessenheit ✓
- Selbstbeschreibungsfähigkeit ✗
- Steuerbarkeit ✗
- Erwartungskonformität ✗
- Fehlertoleranz ✗
- Individualisierbarkeit ✗
- Lernförderlichkeit ✗
- Eignung für das Kommunikationsziel ✗
- Eignung für Wahrnehmung und Verständnis ✗
- Eignung für die Exploration ✗
- Eignung für die Benutzungsmotivation ✗

11 bis 17 sowie 110 der Normenreihe EN ISO 9241 „Ergonomie der Mensch-System-Interaktion“, sowie DIN EN ISO 14915 Softwareergonomie für Multimedia-Benutzungsschnittstellen

Volix II Ausblick

1. Analyse von RAM-Dumps
2. Volatility Framework
3. Volix II Ausblick

Volix Déjà-vu

- Auf dem 2. IT-Forensik Workshop wurde bereits 2012 diese Thematik vorgestellt.
- Die Arbeit vom letzten Jahr war erfolgreich.
  - Es gab einige konzeptionelle Erkenntnisse, sowie darauf beruhende Entscheidungen.

⟹ Entscheidung gegen ein Redesign bzw. Refactoring und für ein Rewrite der Software.

Volix II

„Volatility Interface and eXtensions“

Ziele:

- Ein erweiterbares Interface
- Komfortable Software-Ergonomie
  - Hohe Usability
- Für Erweiterungen offen
  - Zukünftige Projekte sollen an dieses anknüpfen können

Volix II – Ziele Details

Ziele (sichtbar für den Endanwender):

- Multilingual 
  - Beliebig um Sprachpakete ergänzbar 
- Benutzer individuelle Speicherung der Einstellungen 
- Anordnung der Analyse in Fällen 
- Automatisches Reporting 
- Automatisierung von Problemlösungen (Wizards) 
  - Malware, versteckte Verbindungen, Prozesse etc. mit einem Klick finden. 

Volix II –Ziele Details (Fortsetzung)

Weitere Ziele (sichtbar für den Endanwender):

- Nur sinnvolle Schaltflächen aktivieren ✓
  - Eventgesteuerte Benutzeroberfläche ✓
  - Wiedererkennung zu anderen GUI's ✓

(Quelle: eigene Darstellung)

Volix II – nicht sichtbare Ziele

Hinter den Kulissen:

- Multithreaded ✓
- Threadsicher ✓
- Saubere Trennung der Klassen, Ausnutzen vernünftiger Vererbungs-Hierarchien ✓
  - Polymorphes Einbinden der Klassen um redundanten Code zu vermeiden und zukünftige Klassen des gleichen Types dynamisch einzubinden ✓
- Verfolgen der OO-Prinzipien ✓

Volix II - ToDo

ToDo:

- Verarbeitung der einzelnen Plug-Ins
- Visualisierung der Rückgabe
- Verarbeitung der Zwischenablage
  - Viele Aktionen über Drag & Drop
- Metriken optimieren
- Unit Tests

Volix II

Implementierung in C#:

- Ressourcen einbinden für individuelle Oberfläche.
- Event gesteuert
- Dynamisches Erstellen von Controls, so dass die Sicht der Anwendung angepasst wird

→ Hohe Software-Ergonomie, sowie gute Möglichkeiten das Projekt weiter zu endwickeln.

EndofFile

Viele Dank für Ihre Aufmerksamkeit!

Woelker@fh-aachen.de