Emulationsbasiertes Entpacken von laufzeitgepackten Schadprogrammen und darüber hinaus

Lutz Böhne (lutz.boehne@redteam-pentesting.de)
RedTeam Pentesting GmbH
http://www.redteam-pentesting.de

9. Februar 2010, Hamburg
17. DFN Workshop "Sicherheit in vernetzten Systemen"

Über mich

- Lutz Böhne
- Absolvent der RWTH Aachen (2008)
- Vortrag basiert auf meiner Diplomarbeit
- Seit 2008 Penetrationstester der RedTeam Pentesting GmbH

RedTeam Pentesting, Daten & Fakten

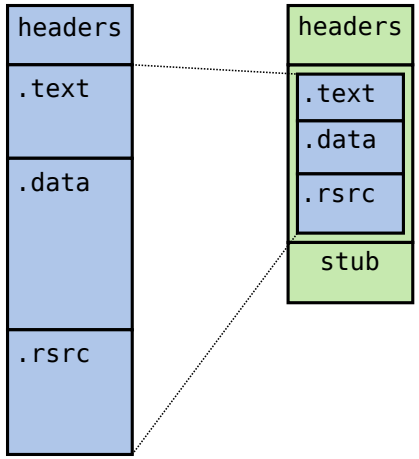
- ★ Gegründet 2004
- ★ Spezialisierung ausschließlich auf Penetrationstests
- ★ Forschungsarbeit im IT-Sicherheitsbereich

Motivation

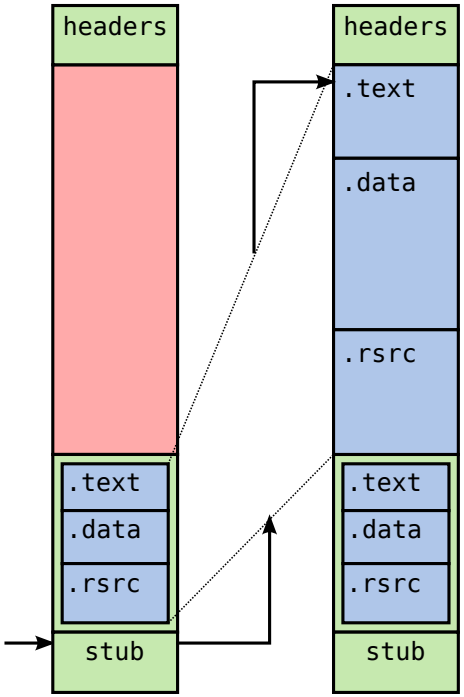
Malware

- ★ Malware stellt wachsende Gefährdung dar.
- ★ Ständig neue Varianten
- ★ Malware oft laufzeitgepackt
- ★ Kaum kostenlose oder freie Analysewerkzeuge verfügbar

Laufzeitpacker



Packen



Entpacken

Schwächen typischer Laufzeitpacker

- ★ CPUs können Code nur in “Klarschrift” ausführen
- ★ Code wird zur Laufzeit in den Arbeitsspeicher entpackt
- ★ Üblicher Ansatz:
  Überwachen der Entpackroutine, Erzeugen eines Speicherabbilds wenn neuer Code ausgeführt wird
- ★ Thema einiger Arbeiten, allerdings werden Werkzeuge oder Source Code selten veröffentlicht.

Implementierung eines automatischen Entpackers

Pandora's Bochs

Pandora's Bochs: automatischer Entpacker, basierend auf *Bochs*

Herausforderungen bei der Entwicklung:

- ★ Unauffälligkeit ggü. Gastsystem
- ★ Kenntnis des Gast-Betriebssystem-Zustands
- ★ OEP-Erkennung
- ★ Terminierung
- ★ Rekonstruktion gültiger PE-Dateien

Instrumentierung

Bochs kann bestimmte Ereignisse instrumentieren, z.B.:

- ★ Modifikation des CR3 (Page Directory Base) Registers
- ★ Speicherzugriffe (insb. Schreibzugriffe)
- ★ Ausführung von Sprungbefehlen

→ Ideal um das Entpacken zu beobachten

Pandora's Bochs wurde auf Basis einer (selbst entwickelten) Python-Schnittstelle zu diesen Mechanismen implementiert.

Instrumentierung

Pandora's Bochs instrumentiert in zwei Stufen:

- Grobe Instrumentierung:
  - beobachte Wechsel des virtuellen Adressraums
  - bestimme jeweils aktuellen Prozess
  - schalte feine Instrumentierung ein oder aus
- Feine Instrumentierung:
  - Protokolliere Schreibzugriffe auf den Arbeitsspeicher
  - Beim Ausführen eines Sprungbefehls:
    $\rightarrow$ liegt das Sprungziel in einem modifizierten Speicherbereich?

Speicherabbilder

Bei einem Sprung in modifizierten Speicherberreich:

- ★ Speichere eines Speicherabbild in Datenbank
- ★ Setze Ausführung des Programms muss fort, falls in mehreren Schritten entpackt wird
  $\rightarrow$ Markiere Speicher um das Sprungziel als nicht modifiziert

OEP-Erkennung

Sprünge in modifizierte Speicherbereiche sind OEP-Kandidaten
Einschränkungen:

- ★ Nur der *erste* Sprung in solch einen Bereich
- ★ Nur Sprungziele innerhalb des ursprünglichen Prozessabbilds
- ★ Letzter Kandidat der Wahrscheinlichste → Wann kann die Beobachtung beendet werden?

Terminierung

Wird noch weiterer Code entpackt?

→ im Allgemeinen unentscheidbar

→ Wann kann das Entpacken beendet werden?

- ★ (Konfigurierbares) Zeitlimit garantiert Terminierung
- ★ Vorher: Bestimme “Innovation” innerhalb des Prozesses
- ★ Halte Emulation an, wenn über einen bestimmten Zeitraum kein beobachteter Prozess Innovation zeigt

Rekonstruktion von gültigen PE-Dateien

Vorgehensweise

- ★ Erzeuge neue Header
- ★ Setze “Entry Point” auf erkannten OEP
- ★ Korrigiere Sektionen
- ★ Rekonstruiere Imports, insbesondere die IAT

API Call Tracing

Mittels API Call Tracing kann das Laufzeitverhalten eines Malware Sample analysiert werdem

- Sprünge werden von Pandora's Bochs bereits instrumentiert
  → Kaum Mehraufwand: stelle fest, ob Ziel API-Funktion ist
- Funktionsprototyp muss bekannt sein, um Stack Layout und Funktionsargumente zu bestimmen.
  → Mit Hilfe von GCC-XML und pygccxml implementiert

Ergebnisse

Kriterien

Pandora's Bochs wurde mit bekannten Programmen und unbekannter Malware getestet. Kriterien:

- ★ Laufzeit
- ★ OEP-Erkennung
- ★ Stimmt entpackter Code mit ursprünglichem Code überein?
- ★ Wurde eine gültige und ausführbare PE-Datei rekonstruiert?

Ergebnisse - Synthetische Samples

- ★ Packen von Notepad (68kB) und Wget (732kB)
- ★ 30 verschiedene Laufzeitpacker in ihren Standardeinstellungen
- ★ in fast allen Fällen konnte versteckter Code extrahiert werden
- ★ OEP in 80% der Fälle korrekt erkannt
- ★ gültige, *ausführbare* PE-Programmdateien konnten für 58% der Samples rekonstruiert werden
- ★ Größtes Hindernis: Packer, die ursprünglichen Code verändern
- ★ Laufzeiten von einigen Minuten bis zu einer Stunde und mehr

Malware Samples

- ★ 409 Samples, die innerhalb eines Monats im Honeynet der RWTH Aachen gesammelt wurden
- ★ 379 durch ClamAV als Malware erkannt, 239 laut PEiD laufzeitgepackt
- ★ 361 konnten ausgeführt werden
- ★ 343 führten Code in modifiziertem Speicher aus
- ★ Durchschnittliche Laufzeit war 7 Minuten und 21 Sekunden
- ★ Qualitative Analyse deutet darauf hin, dass die meisten korrekt entpackt wurden

Fazit

Fazit und Ausblick

- ★ Einfaches Entpacken funktioniert zuverlässig
- ★ API Call Tracing noch nicht vollständig implementiert, dennoch vielversprechende erste Resultate
- ★ Größtes Hindernis für automatisierte Entpacker: Laufzeitpacker, die den ursprünglichen Code verändern
  → Lösbar durch bessere, interaktive Werkzeuge?
- ★ Emulationsgeschwindigkeit nicht zufriedenstellend, teilweise Kompatibilitätsprobleme
  → Andere Emulations-/Virtualisierungslösung?
  → Verzicht auf Python?

Fragen?

- Diplomarbeit: `https://0x0badc0.de/PandorasBochs.pdf`
- Source Code:
  `https://0x0badc0.de/gitweb?p=bochs/.git`
- Folien: `http://www.redteam-pentesting.de`

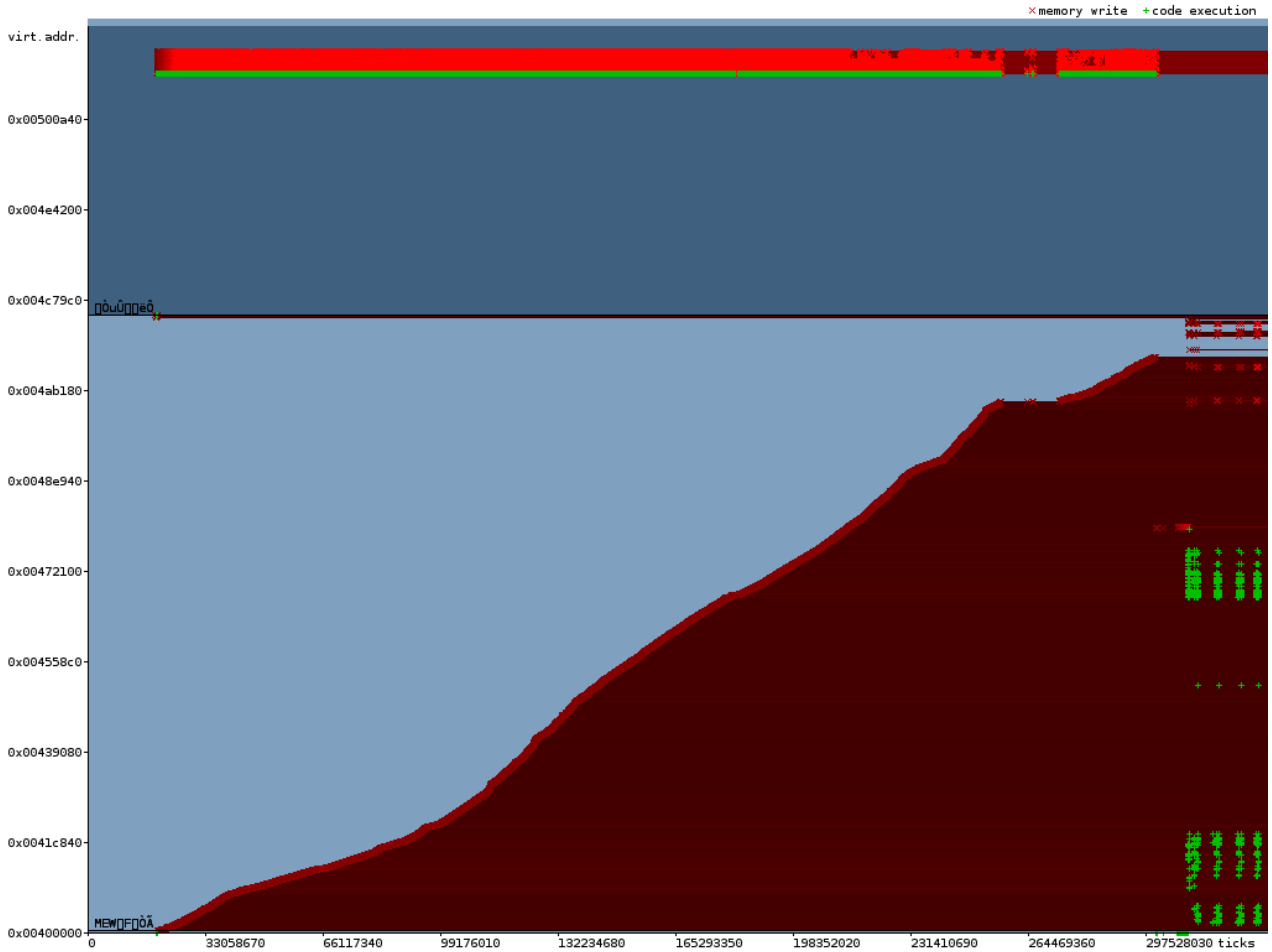


Figure: Entpacken von MEW11SE 1.2

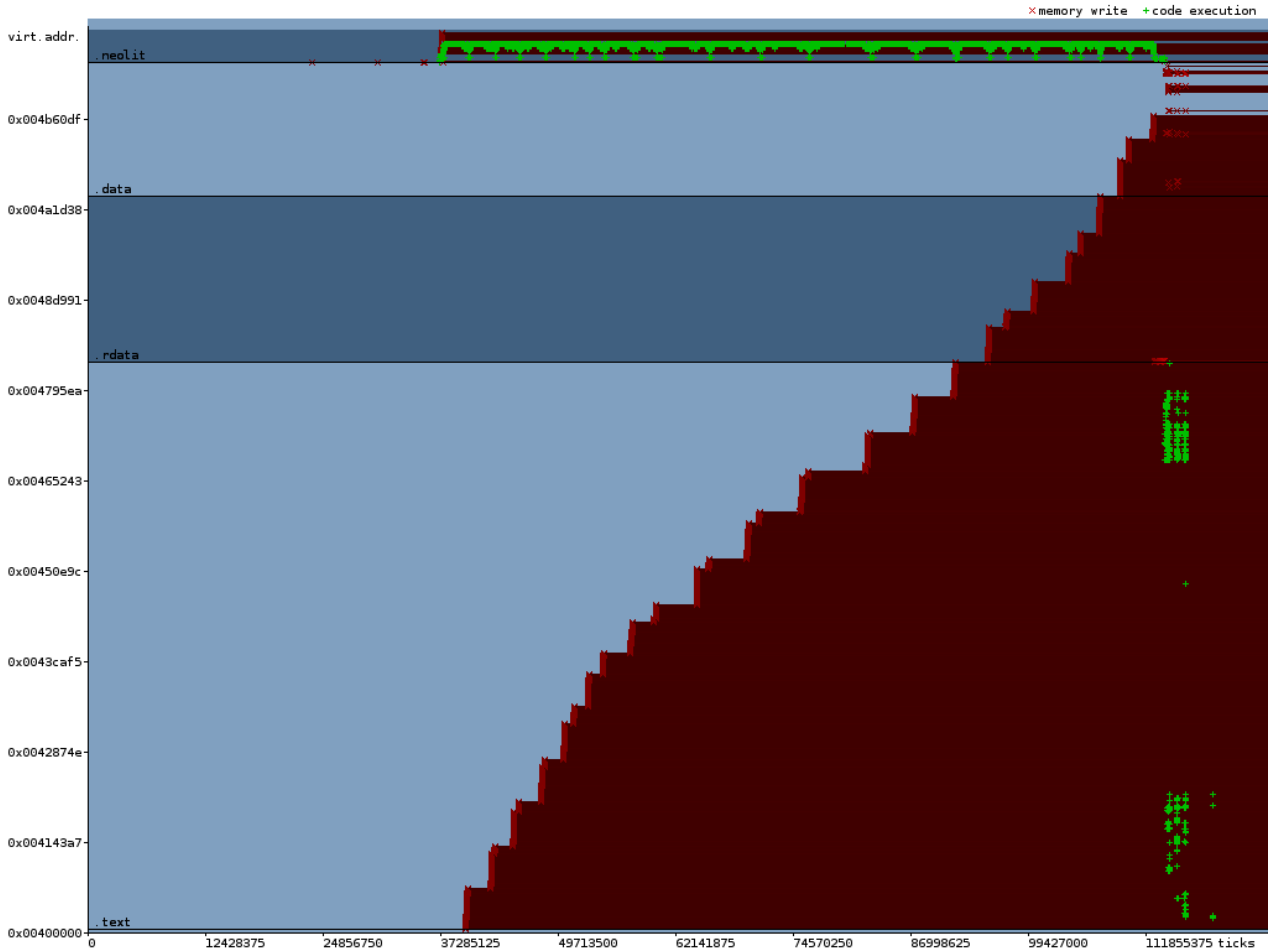


Figure: Entpacken von Neolite 2.0

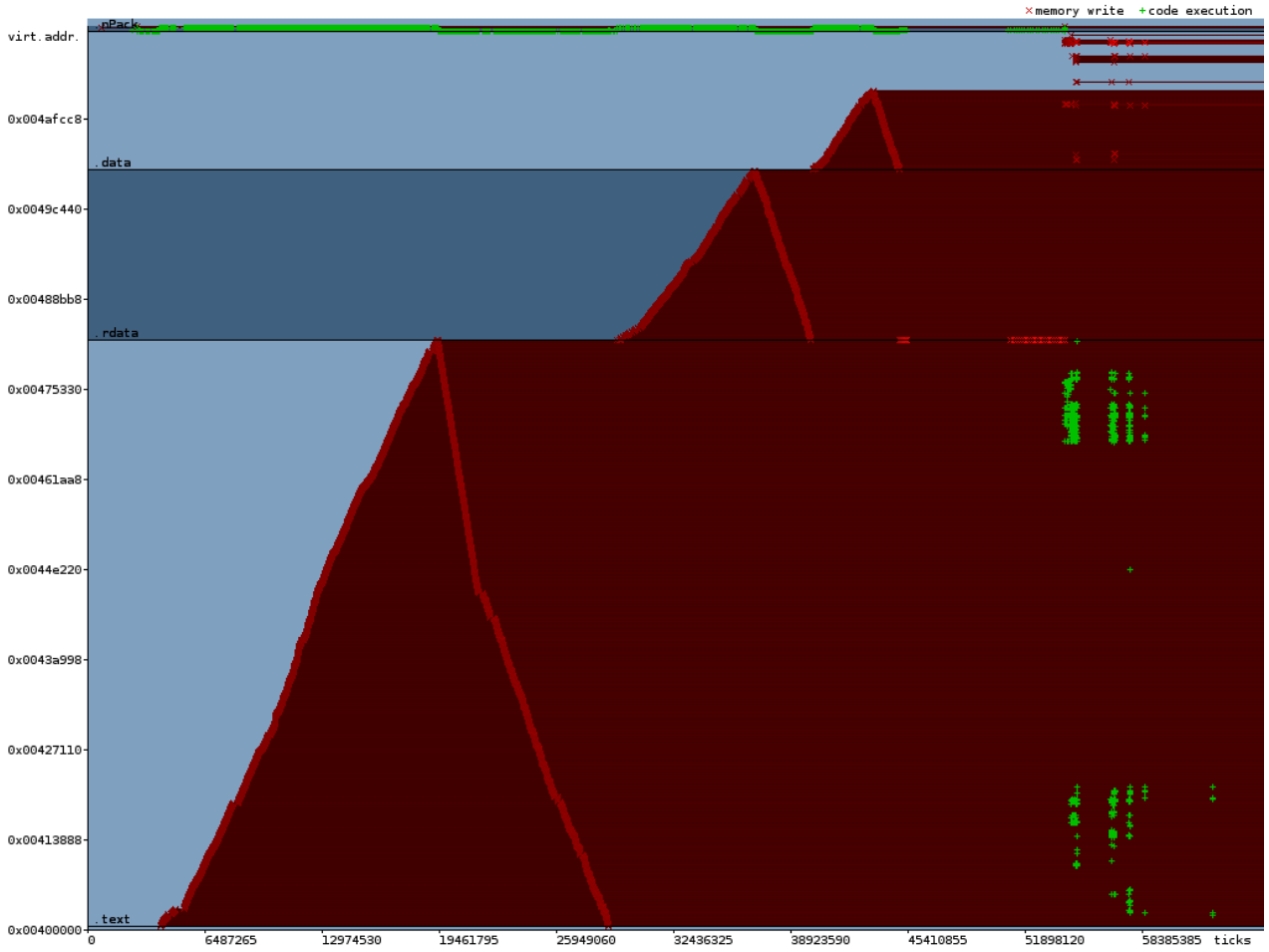


Figure: Entpacken von nPack 1.1300beta

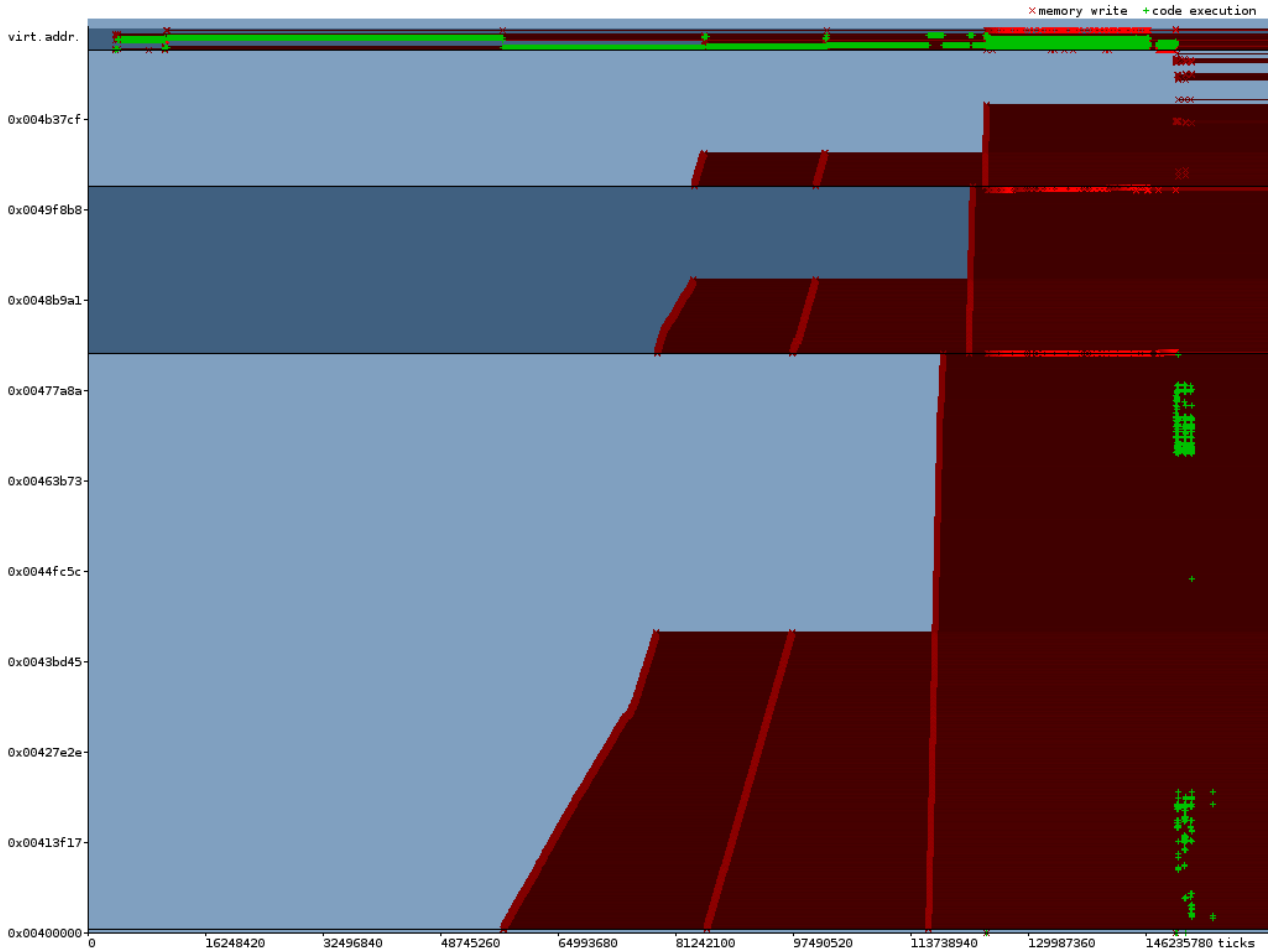


Figure: Entpacken von PESpin 1.304

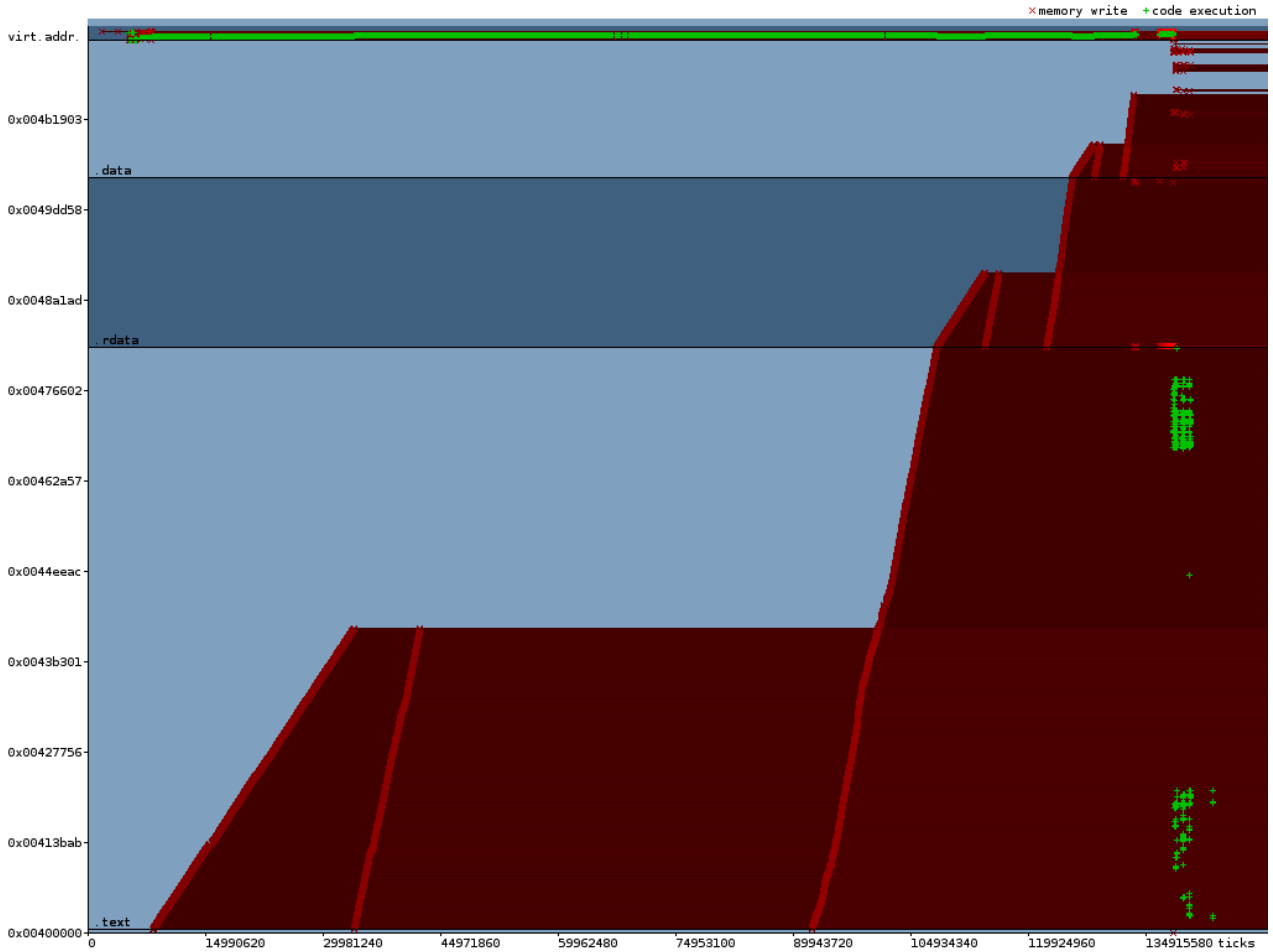


Figure: Entpacken von tELock 0.98

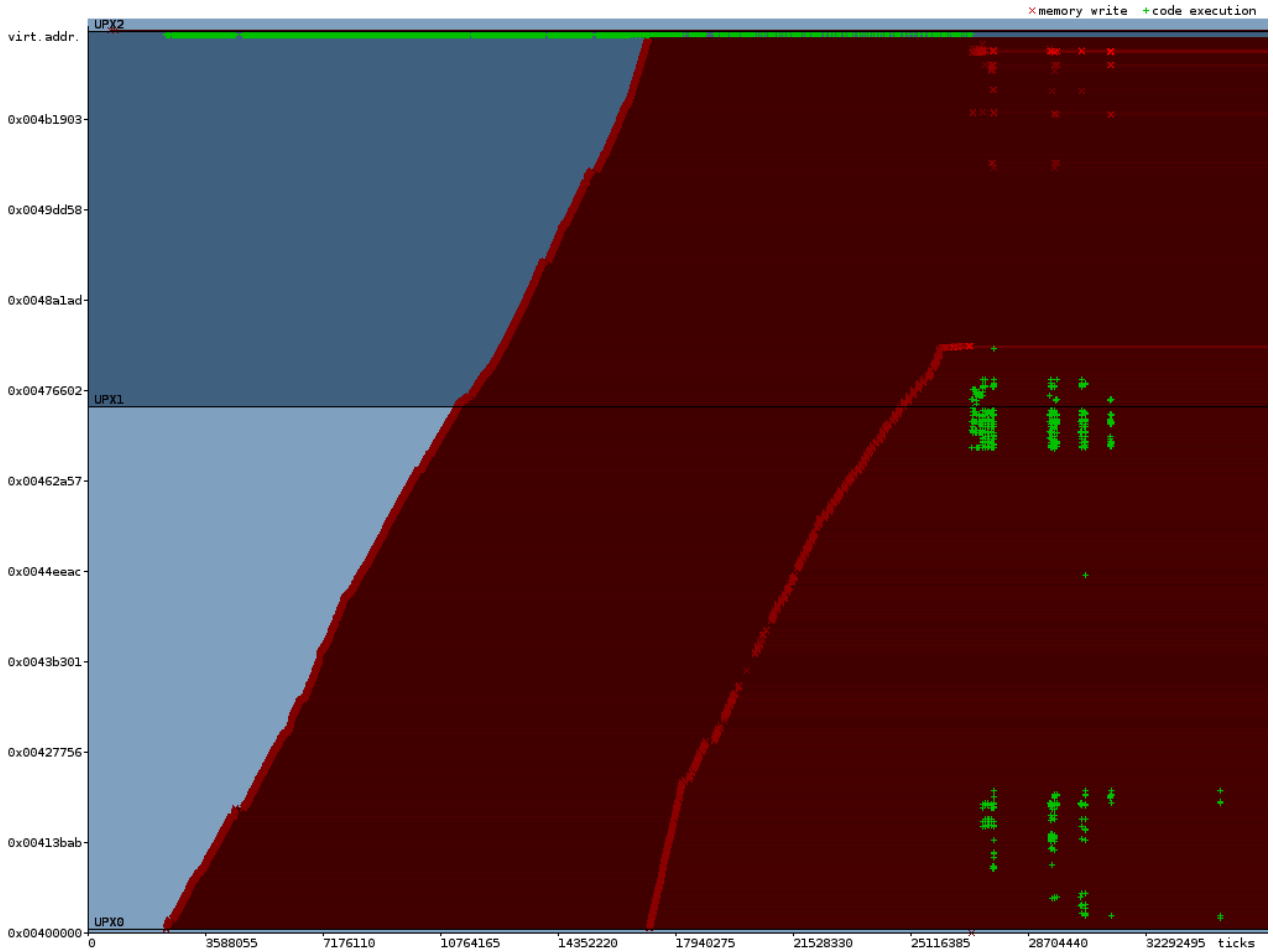


Figure: Entpacken von UPX 3.01