# Gliederung

1. Einleitung
2. Ziel
3. Aufbau und Komponenten
   1. Befehl ausführen
   2. Verhalten aufzeichnen
   3. Verhalten analysieren
   4. Auswertung speichern
4. Funktionen
   1. Daten sammeln
   2. Ausführbare Datei emulieren
5. Fazit & Ausblick
6. Anhang
7. Quellenverzeichnis

# Einleitung

Der Prozess des Software Reverse Engineerings beschäftigt sich mit der Nachkonstruktion vorhandener Produkte. Zum Erreichen des Ziels müssen verschiedene Eigenschaften von Software-Produkten Verstanden und extrahiert werden. Zu diesen Eigenschaften gehören beispielsweise der Aufbau, der Programmfluss, die Struktur von Objekten und die Funktion von Codesegmenten.

Hierfür können verschiedene Techniken angewandt werden. Die gängigste Vorgehensweise besteht darin, einen Disassemblierer (IDA, Hopper, objdump) zu verwenden und die Funktion von Codesegmenten anhand der erkannten Maschinenbefehle nachzuvollziehen. Dieser Vorgang ist sehr zeitintensiv und erfordert viel Erfahrung, vor allem, wenn aus der untersuchende Software hilfreiche Symboltabellen entfernt wurde, welche die in der Programmiersprache verwendeten Variablen- und Funktionsnamen enthalten können.

Um das Verstehen von Objekt-Dateien zu erleichtern, wird aktuell versucht aus dem schwer lesbaren Maschinencode für den Menschen leichter lesbaren (Pseudo-)Code zu generieren. Dieser Vorgang des Decompilierens befindet sich jedoch aufgrund vieler Schwierigkeiten bei der Umsetzung noch in einer frühen Phase. Das Problem besteht darin, dass zwischen mehreren Maschinenbefehlen, die auch an unterschiedlicher Stelle stehen können, ein Zusammenhang erkannt werden muss, der dem Benutzer leicht verstehbar ausgegeben werden muss. Ebenso können Missverständnisse entstehen, wenn Maschinenbefehle direkt in lesbaren Code umgewandelt werden oder erst zusammengefasst werden. Durch kleine Fehler in der Übersetzung kann die Darstellung des Codes verwirrender als der Maschinencode sein oder es können wichtige Schritte des Programms verloren gehen. Bei aktuellen Decompilern fällt auf, dass der Maschinencode lediglich Zeile für Zeile übersetzt wird. So werden beispielsweise Jump-Anweisungen nicht in eine besser verstehbare If-Anweisung umgewandelt, sondern direkt mit der im Binärcode entsprechenden Sprungadresse ausgegeben. Dadurch entsteht kein Mehrwert, weshalb diese simple Übersetzung als überflüssig angesehen werden kann.

Ein weiterer Bereich, der zum Software Reverse Engineering hinzugezählt werden kann, ist die Emulation. Hierbei wird eine Maschine durch Software nachgebildet und Maschinencode nur virtuell ausgeführt. Meist wird diese Technik benutzt um Maschinencode auf einem nicht mit diesem Code kompatiblen System auszuführen. Dies wird erreicht, indem der Maschinencode geparst wird und anschließend durch Datenbanken, welche die Bedeutung des Maschinencodes enthalten, die entsprechenden Änderungen am nachgebildeten System vorgenommen werden. So kann Software, bzw. Maschinencode, ohne direktem Zugriff zur Hardware ausgeführt werden. Die Vorteile zeichnen sich zum einen durch die Möglichkeit aus, nicht kompatiblen Bytecode auf einer Maschine laufen zu lassen und zu nutzen. Zum anderen bietet die Emulation die Möglichkeit Software schrittweise auszuführen und die Veränderungen zu beobachten und nachzuvollziehen. Auch Schadcode kann ohne Bedenken untersucht werden, weil die Maschine lediglich durch einen Prozess simuliert wird, welche durch beenden dieses Prozesses aus dem Speicher verschwindet.

Das schwierigste und zeitintensivste bei der Programmierung eines solchen Emulator besteht aus der Erstellung der benötigten Datenbanken. Es muss für jeden auf der Maschine verfügbaren Befehl ein Datensatz erstellt werden, der das Verhalten der Maschine beschreibt. Hierfür müssen große Dokumente, die sogenannten Befehlssatzarchitekturen, durchgearbeitet und für den Emulator abgespeichert werden. Genau dieses Problem ist der Gegenstand dieser Arbeit und soll durch automatische Generierung der benötigten Daten beseitigt werden.

# Ziel

Im Vordergrund dieser Arbeit steht die Untersuchung der Möglichkeit Informationen über Maschinenbefehle für einen Emulator automatisch zu generieren und für eine Emulation einer beliebigen ausführbaren Binärdatei zu nutzen. Hierfür müssen vorab die benötigten Komponenten mit ihren Funktionen definiert werden. Anschließend muss festgelegt werden, wie und mit welchen Hilfsmitteln diese Funktionen implementiert werden.

Zuerst stellt sich die Frage, wie diese Daten ohne weitere Kenntnisse ermittelt werden können. Dies kann erreicht werden, indem auf der zu untersuchenden Maschine alle möglichen Befehle ausgeführt werden und die Zustände des Systems vor und nach dem jeweiligen Befehl aufgezeichnet und untersucht werden. Nach der Analyse muss die Auswertung für den Emulator aufbereitet und abgespeichert werden. Nach erfolgreicher Abarbeitung aller Befehle soll es möglich sein, eine beliebige ausführbare Binärdatei zu emulieren und jeden einzelnen Schritt nachvollziehen zu können. Im Optimalfall soll der Benutzer über eine Schnittstelle sowohl mit dem Emulator, als auch mit der der emulierten Datei kommunizieren können.

Bei der Verwirklichung des Ziels treten verschiedene Probleme auf, die möglicherweise zeitaufwendig lösbar sind. Auch für die Geschwindigkeit des Sammelns der Daten, bzw. der Emulation einer Datei, muss auf eine effiziente Implementierung geachtet werden, welche meist viel Zeit benötigt. Aufgrund der begrenzten Zeit wurde in dieser Arbeit die Priorität auf die Kernfunktionen des Programms gelegt. Genauere Beschreibungen zu Problemen und Vernachlässigungen bei der Programmierung finden sich in den folgenden Kapiteln an den jeweiligen Stellen.

# Befehl ausführen

Hierbei müssen die Befehle auf verschiedene Eigenschaften untersucht werden.

# Auswertung speichern

Die nächste Frage besteht darin, wie die Datensätze dargestellt und gespeichert werden. Da die Art des Speicherns, bzw. das Speichermedium, (SQL, Textdatei) und das Format (Trennzeichen, JSON) nur schnittstellenentscheidend und deshalb nicht relevant für die Kernfunktion des Programms ist, wird hier nicht weiter darauf eingegangen. In der programmatischen Ausarbeitung wurde die Speicherung in üblichen Textdateien mit bestimmter Syntax mit Trennzeichen gewählt. Das wichtige ist eher die interne Darstellung der extrahierten Informationen. Mehr dazu im Kapitel „Darstellung