



**WHZ** Westsächsische  
Hochschule Zwickau  
Hochschule für Mobilität

Bachelorarbeit zum Thema

## Schadprogrammerkennung in Windows 10 mittels der Analyse von Aufrufen von Betriebssystem-Programmierschnittstellen

Vorgelegt von:

Khanh Dinh An

Tor Straße 02

08107 Kirchberg

E-Mail-Adresse: [ankhanhvn99@gmail.com](mailto:ankhanhvn99@gmail.com)

Matrikelnummer: 40990

Studiengang Informatik

9. Fachsemester

Betreuung: Prof. Dr. Frank Grimm

Datum der Abgabe: 06.10.2023

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abbildungsverzeichnis

Anlagenverzeichnis

1. Einleitung

1.1. Motivation

1.2. Zielsetzung

2. Definition von Begriffen und Symbolen

2.1. Kernelmodus und Benutzermodus

2.2. Event Tracing for Windows

2.3. Multiple Sequence Alignment

2.4. Longest Common Subsequence

3. Stand der Forschung

4. Methodologie

4.1. Überblick über das Verfahren

4.2. Analyse der Experimentierumgebung

4.3. Auswahl von Software und Werkzeugen

5. Umsetzung

5.1. Einrichtung der Experimentierumgebung

5.1.1. Einrichtung eines internen Netzwerks

5.1.2. Einrichtung der virtuellen Maschinen

5.1.3 Erstellung der Datenbanktabelle

5.2. Entwicklung des Programmcodes zur Ausführung von Schadprogrammen

5.2.1. Das FASTA-Dateiformat

5.2.2. Programmcode in VM1

5.2.3. Programmcode in VM2

5.3. Ausführung der Projekte

5.4. Kodierung der Ereignisse

5.5. Entfernung der duplizierte Zeilen

5.6. Entfernung der duplizierten Einträge in Log-Datei

5.7. Anwendung des MSA-Algorithmus auf die Daten

5.8. Berechnung der längsten gemeinsamen Teilsequenzen

5.8.1. Algorithmen zur Berechnung von LCS

5.8.2. Umsetzung der Algorithmen in C#

5.9. Analyse von Schadprogrammen

5.10. Unit Test des Programmcodes

5.11. Ergebnis der Umsetzung

6. Bewertung des Ergebnisses

7. Zusammenfassung

## Abkürzungsverzeichnis

ETW.....	Event Tracing for Windows
LCS.....	Longest common subsequences
MSA.....	Multiple sequence alignment
VM.....	Virtuelle Maschine

## Abbildungsverzeichnis

Abbildung 1: Die vier Privilegienstufen (Ringe) x86-kompatibler Prozessoren [BAUN22, S. 154].....	2
Abbildung 2: MSA in ugene [OKO12].....	3
Abbildung 3: Forschungsdesign von Ki et al.(2015) [KI15].....	5
Abbildung 4: Erstellung eines internen Netzwerks in VirtualBox (vgl. [ORAC]).....	9
Abbildung 5: Auflisten des Netzwerks in VirtualBox (vgl. [ORAC]; vgl. [CYBER21]).....	10
Abbildung 6: Ein Beispiel für das Konstrukt einer FASTA-Datei (vgl. [NCBI]).....	12
Abbildung 7: Methode ConnectServer() (vgl. [MICRO23 -b]).....	13
Abbildung 8: Methode StartProcess() (vgl. [MICRO23 -d]; vgl. [MICRO23 -e]; vgl. [MORRI18]; vgl. [MICRO23 -h]).....	14
Abbildung 9: Methode DisconnServer() (vgl. [MICRO23 -i]).....	16
Abbildung 10: Methode LogSyscalls() (vgl. [KURDI23]).....	17
Abbildung 11: Methode SendTxtToServer() (vgl. [MICRO23 -j]).....	17
Abbildung 12: Methode StartTcpServer() (vgl. [PINE22]; vgl. [MICRO23 -m]).....	18
Abbildung 13: Ereignissen im Kernelmodus nach Gruppe [ANDKA23].....	20
Abbildung 14: EventTraceHeader [ANDKA23].....	21
Abbildung 15: Gruppencode der Ereignisse im Kernelmodus (vgl. [ANDKA23]).....	22
Abbildung 16: Methode readSyscallEncoding() (vgl. [MICRO23 -p]; vgl. [MICRO23 -o]; vgl. [MICRO23 -n]).....	23
Abbildung 17: Duplikatentfernung in der FASTA-Datei (Eigene Darstellung).....	25
Abbildung 18: Duplikatentfernung mit der Methode removeDupLines() (vgl. [MICRO23 -d]; vgl. [MICRO23 -r]; [MICRO23 -p]; vgl. [MICRO23 -q]).....	27
Abbildung 19: Methode getCode() (vgl. [MICRO23 -n]).....	28
Abbildung 20: Ergebnis der Entfernung von duplizierten Zeilen (vgl. [ANDKA23]).....	28
Abbildung 21: Methode addSyscallToServer() (vgl. [MICRO23 -d]; vgl. [MICRO23 -p]; vgl. [MICRO23 -r]; vgl. [MICRO23 -u]; vgl. [MICRO23 -w]).....	31
Abbildung 22: Tabelle zur Speicherung von Ereignissen (vgl. [ABUSE]).....	32

Abbildung 23: Methode readFromDB() (vgl. [MICRO23 -w]; vgl. [KAM18]; vgl. [MICRO23 -t]).....	33
Abbildung 24: MAFFT als Kommandozeile (vgl. [OSAKA23]).....	35
Abbildung 25: MAFFT als Kommandozeile (Forts.) (vgl. [OSAKA23]).....	36
Abbildung 26: LCS-Algorithmus [HIRSCH75, S. 341].....	37
Abbildung 27: Beispiel für Berechnung von LCS (vgl. [CORMEN17, S. 398]).....	38
Abbildung 28: Methode calcLCS() (vgl. [MICRO23 -v]; vgl. [MICRO23 -w]).....	39
Abbildung 29: Methode GetSeqById() (vgl. [MICRO23 -x]; vgl. [MICRO23 -w]).....	39
Abbildung 30: Methode lcs() (vgl. [GEEKS23]; siehe Abbildung 1.3 und 1.4 in Anlage 1; vgl. [MICRO23 -y]).....	40
Abbildung 31: Methode Backtrack() (siehe Abbildung 1.3 und 1.4 in Anlage 1; vgl. [MICRO23 -y]).....	41
Abbildung 32: Methoden zum Schreiben und Lesen von der Datenbank (vgl. [MICRO23 -w]; vgl. [MICRO23 -x]).....	42
Abbildung 33: Methode AnalyseMalware() (vgl. [MICRO23 -w]; vgl. [KAM18]; vgl. [MICRO23 -x]).....	43
Tabellenverzeichnis	
Tabelle 1: Struktur der Tabelle (Eigene Darstellung).....	12

## Anlagenverzeichnis

### 1. Persönliche Kommunikation mit ChatGPT

Abbildung 1.1: Frage an ChatGPT [OPENAI23]

Abbildung 1.2: Antwort von ChatGPT [OPENAI23]

Abbildung 1.3: Antwort von ChatGPT (Fortsetzung) [OPENAI23]

Abbildung 1.4: Antwort ChatGPT (Fortsetzung) [OPENAI23]

Abbildung 1.5: Antwort ChatGPT (Fortsetzung) [OPENAI23]

## 1. Einleitung

### 1.1. Motivation

Die Anzahl von Cyber Angriffen durch Schadprogramme auf private und produktive IT-Systeme nehmen jährlich stetig an, Hacker werden immer neue anspruchsvolle Methoden und Techniken für die Angriffe gestalten, um die Erkennung der IT-Sicherheitssysteme umzugehen.

Schadprogramme sind Computerprogramme, die aus Programmierarbeit mit böswilliger Absicht resultieren und auf einem Rechner installiert werden. Die Autoren der Schadprogramme versuchen, Schaden an den Besitzern des Rechners zu verursachen. Der Schaden könnte wirtschaftlicher Schaden, betriebliche Störung oder Rufschädigung im Fall eines Unternehmens sein. Das Betriebssystem stellt verschiedene Funktionen zur Verfügung, um Computerprogramme zu ermöglichen, mit der Hardware des Rechners zu kommunizieren und genau diese Funktionen nutzen Schadprogramme aus, um Änderungen an dem Rechner vorzunehmen, indem sie es in Ihrem Programmcode aufrufen.

Um die Anwesenheit des Schadprogramms auf einem Rechner zu erkennen, können die Aufrufe an Systemfunktionen von jedem Prozess verfolgt und analysiert werden.

### 1.2. Zielsetzung

In der Arbeit soll eine Anwendung in C# programmiert werden, welche die Ereignisse im Kernelmodus von Schadprogrammen verfolgen und in einer Datenbank speichern können. Die Schadprogramme müssen in der Windows 10 64-Bit-Version ausführbar sein. Darüber hinaus soll durch die Benutzung von einer Open-Source-Anwendung Multiple Sequence Alignment(MSA) auf die Daten angewendet werden. Weiterhin soll der Algorithmus zur Berechnung von Longest common subsequences(LCS) implementiert werden. Die Anwendung soll die Ereignisse im Kernelmodus einer Anwendung mit den Datenbeständen vergleichen können.

In der Arbeit wird die Effektivität der Schadprogrammerkennung nicht gemessen, weil innerhalb des Forschungszeitraums nicht genügend Schadprogramm analysiert werden kann. In der Forschung von Ki et al.(2015) haben die Verfasser insgesamt 23080 Schadprogramme und harmlose Programme analysiert. Jedes Programm wurde in 2 Minuten ausgeführt. [Übers. durch den Verf.] (vgl. [K15]). Die Analyse von 23080 Programmen wird damit mindestens 32 Arbeitstage dauern.

## 2. Definition von Begriffen und Symbolen

### 2.1. Kernelmodus und Benutzermodus

Baun (2022) sagt, dass der Betriebssystemkern (englisch: Kernel) die grundlegenden Funktionen des Betriebssystems enthält. Er ist die Schnittstelle zur Hardware des Computers. x86-kompatible Prozessoren enthalten vier Privilegienstufen, die auch Ringe heißen. Jeder Prozess wird in einem Ring ausgeführt und kann sich nicht selbstständig aus diesem befreien. Im Ring 0, dem sogenannten Kernelmodus, läuft der Betriebssystemkern. Prozesse, die im Kernelmodus laufen, haben vollen Zugriff auf die Hardware. Im Ring 3, dem sogenannten Benutzermodus, laufen die übrigen Prozesse. Diese arbeiten ausschließlich mit virtuellem Speicher [BAUN22, S. 36,153-154] (siehe Abbildung 1).

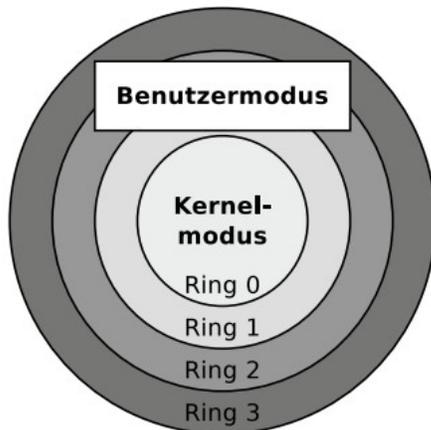


Abbildung 1: Die vier Privilegienstufen (Ringe) x86-kompatibler Prozessoren [BAUN22, S. 154]

### 2.2. Event Tracing for Windows

Bridge et al. (2021) sagt zu Event Tracing: „Event Tracing for Windows (ETW) is an efficient kernel-level tracing facility that lets you log kernel or application-defined events to a log file. You can consume the events in real time or from a log file and use them to debug an application or to determine where performance issues are occurring in the application“ [BRID21].

### 2.3. Multiple Sequence Alignment

Carrilo und David (1988) definiert MSA wie folgt: „An alignment of the sequences  $S_1, \dots, S_n$  is another set of sequences  $\bar{S}_1, \dots, \bar{S}_n$ , such that each sequence  $\bar{S}_i$

is obtained from  $S_i$  by inserting blanks in positions where some of the other sequences have a nonblank character“ [CARDA88, S. 1074].

Abbildung 2 stellt ein Beispiel von MSA durch die Verwendung von ugene dar

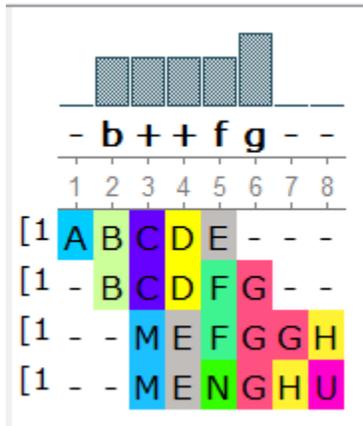


Abbildung 2: MSA in ugene [OKO12]

## 2.4. Longest Common Subsequence

LCS ist der englische Begriff für die längsten gemeinsamen Teilsequenzen.

Maier (1978) sagt zu LCS: „Given a finite sequence  $S = S_1, S_2, \dots, S_n$ , we define a subsequence  $S'$  of  $S$  to be any sequence which consists of  $S$  with between 0 and  $m$  terms deleted (e.g.  $ac$ ,  $ad$ , and  $abcd$  are all subsequences of  $abcd$ ). We write  $S' < S$  if  $S'$  is a subsequence of  $S$ . We also say that  $S$  is a supersequence of  $S'$ , and write  $S > S'$ . Given a set  $R = \{S_1, S_2, \dots, S_p\}$  of sequences, we speak of a Longest Common Subsequence of  $R$ ,  $LCS(R)$ , as a longest sequence  $S$  such that  $S < S_i$ , for  $i = 1, \dots, p$ . For example,  $abe = LCS(\{ababe, cabe, abdde\})$ . Actually,  $LCS(R)$  is a set of subsequences, since there may be more than one sequence fitting the definition“ [MAIER78, S. 1].

### 3. Stand der Forschung

Verschiedene Forschungen folgen im Bereich Schadprogrammerkennung dem dynamischen und verhaltensbasierten Ansatz. In der dynamischen Erkennung werden Schadprogramme in einer isolierten Umgebung ausgeführt und deren Funktionsweise wird analysiert. In dem verhaltensbasierten Ansatz wird versucht, das Verhalten der Schadprogramme anhand u.a. Funktionsaufrufe zu analysieren.

Die Forschung von Ki et al. (2015) hat den dynamischen und verhaltensbasierten Ansatz von Schadprogrammerkennung verfolgt, indem sie die Open-Source-Anwendung Microsoft Detours genutzt haben, um die Aufrufe an Systemfunktionen während der Laufzeit der Schadprogramme zu erfassen. Danach wurden durch verschiedene Algorithmen von MSA, die aus dem Bereich Genomsequenz-Analyse stammen, die typischen Muster von Aufrufe-Reihenfolgen bestimmt. Das Algorithmus LCS wurde danach auf diese gefundenen Muster angewendet. Diese Sequenzen sowie kritische Sequenzen von Schadprogrammen und die von gutartigen Programmen wurden in einer Datenbank gespeichert, die zum Vergleichen dient, um andere Schadprogramme zu erkennen (vgl. [KI15]).

Die Forschung von Ki et al. (2015) hat nur die Funktionsaufrufe im Benutzermodus analysiert, deswegen konnte Ihr System nicht die Funktionsaufrufe im Kernelmodus erkennen. Daneben hat die Forschung auf Schadprogramme in einem Computer mit Windows XP 32-Bit Betriebssystem getestet. Diese Arbeit soll auf das Windows 10 64-Bit Betriebssystem und Funktionsaufrufe im Kernelmodus zielen.

## 4. Methodologie

### 4.1. Überblick über das Verfahren

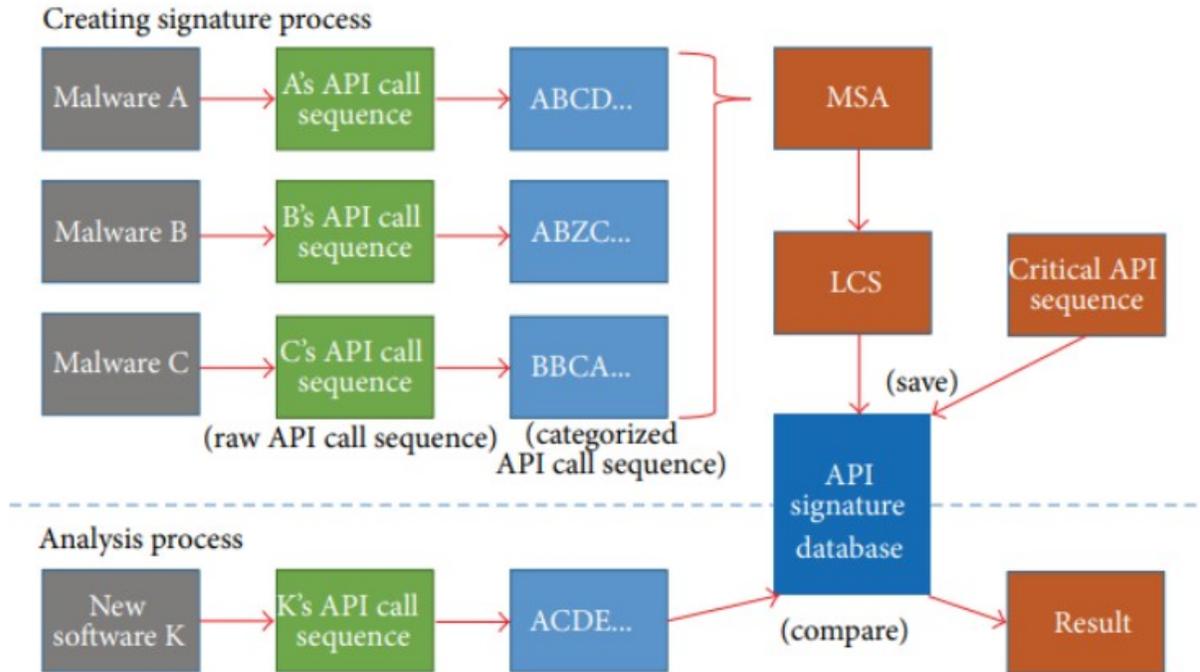


Abbildung 3: Forschungsdesign von Ki et al.(2015)  
[K115]

Abbildung 3 stellt eine Übersicht zur Methode in der Forschung von Ki et al. (2015) dar. Diese Arbeit soll die gleichen Methoden verfolgen. Mit ETW werden die Ereignisse im Kernelmodus von Schadprogrammen in einer Textdatei gespeichert. Die Textdatei wird zunächst in ClustalX importiert, um die Muster zu bestimmen. Danach wird LCS auf die Muster angewendet und die sortierten Sequenzen werden in einer Microsoft SQL Server Datenbank gespeichert. Eine C# - Anwendung wird entwickelt, um mittels der Bibliothek EventTracing die Ereignisse im Kernelmodus zu extrahieren und diese mit den Datenbeständen zu vergleichen.

Zwei VM mit dem Betriebssystem Windows 10 werden eingerichtet. Die Schadprogramme werden in einer virtuellen Maschine vom Internet heruntergeladen. Während der Ausführung der Schadprogramme werden Ereignisse im Kernelmodus von jedem Schadprogramm an eine andere VM gesendet, um die Daten dort abzuspeichern. Danach werden die Daten in die

Host-Maschine kopiert, in der Host-Maschine werden die Daten in einer Datenbank gespeichert. Der Sequenzvergleich der Daten anhand der MSA wird durch Open Source Anwendungen ausgeführt. Zum Schluss wird der Algorithmus zur Berechnung von LCS in C# entwickelt. (vgl. [KI15])

#### 4.2. Analyse der Experimentierumgebung

Die Schadprogramme müssen in einer virtuellen Maschine ausgeführt werden, weil die Hardware der virtuellen Maschine von der Host-Maschine getrennt ist, damit wird sichergestellt, dass die Schadprogramme nicht auf die Daten der Host-Maschine zugreifen können.

Eine zweite VM zum Abspeichern der Daten wird eingerichtet. Wenn die Datenspeicherung und Ausführung der Schadprogramme in einer VM geschehen, dann führt der Defekt der VM zum Löschen der Daten. Die VM muss so konfiguriert werden, dass sie keine Internetverbindung sowie keine Netzwerkverbindung mit der Host-Maschine hat, weil die Schadprogramme durch eine Netzwerkverbindung andere Computer infizieren können. Eine weitere Anforderung an die VM ist die Fähigkeit, jederzeit den aktuellen Zustand zu speichern und den früheren Zustand wiederherzustellen, welche die Wiederherstellung der Daten und Einstellungen ermöglicht.

Hackersplit (2019) sagt, dass die meisten moderne Schadprogramme mit dem Mechanismus gegen Schadprogrammerkennung programmiert sind. Wenn ein Schadprogramm die Ausführung in einer VM erkennt, würde er sich anderweitig verhalten [Übers. durch den Verf.] (vgl. [HACKER19]). Das würde eine gefälschte Reihenfolge von Ereignissen auslösen. Deshalb ist es wichtig, die VM so zu konfigurieren, dass sie möglichst einen produktiven Computer abbildet.

Weiterhin erläutert Hackersplit (2019), die VM muss mindestens einen 4 GB Arbeitsspeicher, 2 Hauptprozessoren und 100 GB Speicherplatz haben. Darüber hinaus müssen die häufig verwendeten Anwendungen installiert werden, dazu gehören u. a. Webbrowser, Datenkompression-Programme und Textverarbeitung-Programme. Während der Ausführung der Schadprogramme sollten einige Dateien geöffnet werden. Damit das Schadprogramm nicht die Ausführung in einer Sandbox Umgebung erkennt [Übers. durch den Verf.] (vgl. [HACKER19]).

### 4.3. Auswahl von Software und Werkzeugen

Mit der oben genannten Anforderung wird VirtualBox von Oracle (2023) gewählt. VirtualBox bietet die Nutzende die Möglichkeit, ein internes Netzwerk zwischen den VM einzurichten. Mit Snapshot in VirtualBox können die Benutzende eine Momentaufnahme der VM durchführen, in der Aufnahme sind alle Informationen zum Zustand der VM enthalten und der Zustand könnte dadurch wiederhergestellt werden (vgl. [ORAC23]).

Ein anderes bekanntes Werkzeug zur Virtualisierung ist VMware von VMware (2023). Sie wird nicht gewählt, weil die kostenlose Version nicht die Funktionalität zum Erstellen eines internen Netzwerks bietet (vgl. [VM23]).

In der Arbeit wird ETW eingesetzt, weil es direkt von Windows 10 entwickelt und dauerhaft unterstützt wird. Darüber hinaus bietet Microsoft die Programmierenden eine Bibliothek namens TraceEvent zur Verarbeitung der von ETW generierten Daten (vgl. [KURDI23]).

Die Bibliothek ist im .NET-Plattform nutzbar, deswegen wird in der Arbeit mit C# programmiert, zur Programmierung und Ausführung des Programmcodes wird Visual Studio benutzt. [MIRCO22]

Robbins (2018) sagt, dass die Bibliothek TraceEvent eine Klasse TraceEventSession anbietet, welche eine Sitzung zum Verfolgen der Ereignisse aufmacht. Das Einbinden eines Objekts der Klasse TraceEventSession im Programmcode bietet die Programmierenden die Freiheit, eigenen Code einzubinden [Übers. durch den Verf.] (vgl. [ROB18]).

Zum Speichern der Daten wird in der Arbeit Microsoft SQL Server 2022 Express gewählt. [MIRCO22 -b] Eine Tabellenspalte soll auf dem Datentyp VARCHAR(MAX) sein, welcher bis zu 2 GB Zeichen speichern kann (vgl. [RAY23]).

## 5. Umsetzung

Die vorherigen Kapitel schaffen einen Überblick über das gesamte Verfahren und erklären die Anforderungen an die Werkzeuge und daraus ausgewählte Anwendungen. Dieses Kapitel beschreibt die technische Umsetzung.

### 5.1. Einrichtung der Experimentierumgebung

#### 5.1.1. Einrichtung eines internen Netzwerks

In einem internen Netzwerk kann eine VM nur mit den anderen virtuellen Maschinen innerhalb des Netzwerks kommunizieren und somit keine Verbindung zum Internet sowie zur Host-Maschine herstellen [ORAC]. Damit können die Schadprogramme außerhalb der VM sich nicht verbreiten.

Das YouTube-Video von Cyber Studies (2021) zeigt in einem praktischen Beispiel die Anleitung zur Erstellung eines internen Netzwerkes in VirtualBox. Bei Einstellung einer virtuellen Maschine wird bei „Attached to“ die Auswahl „Internal Network“ gewählt, anschließend bei Name den Namen des Netzwerks eingeben. Nachdem die Eingabetaste geklickt wurde, wird ein internes Netzwerk mit dem eingegebenen Namen erstellt (siehe Abbildung 4 auf nächster Seite).

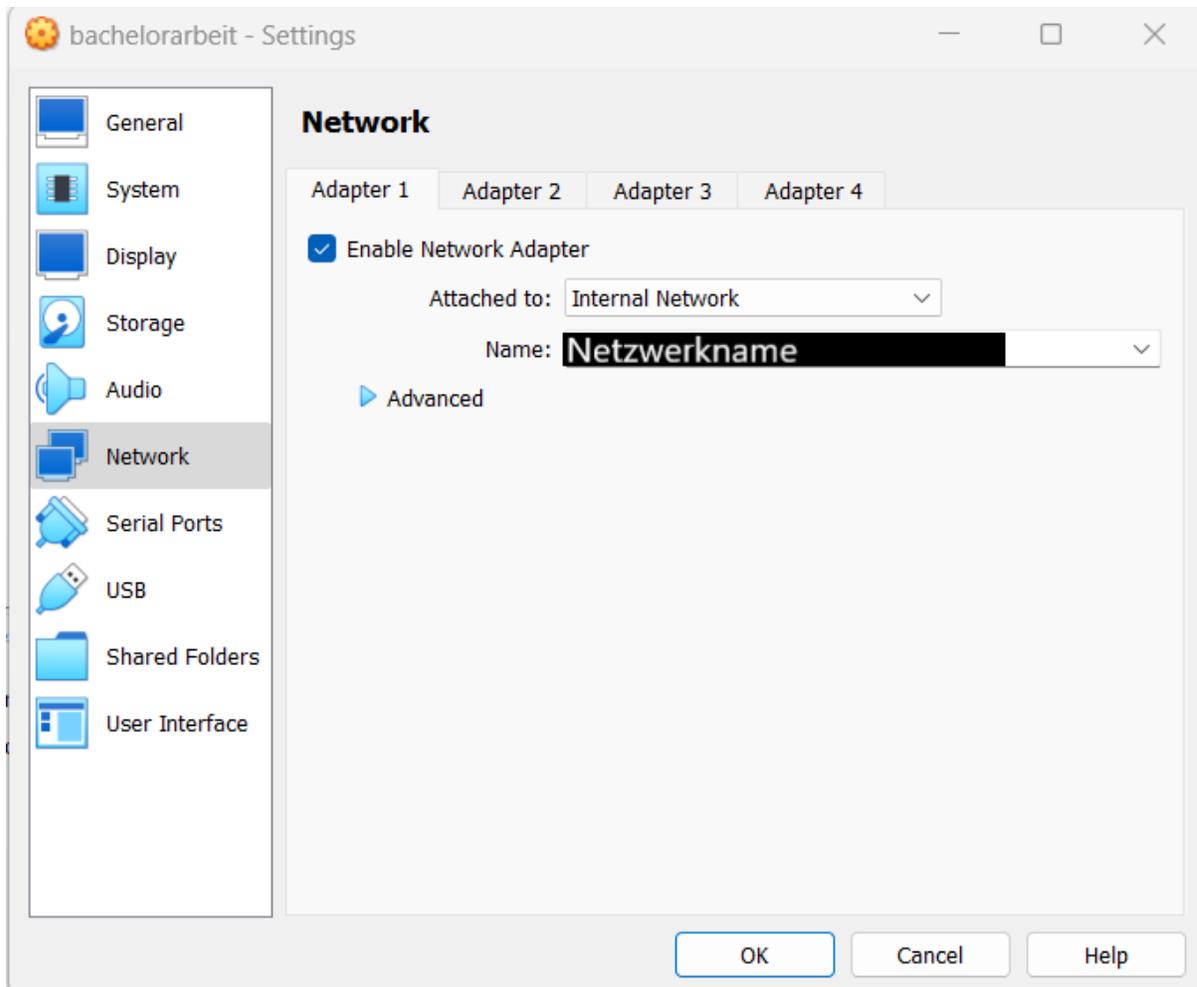


Abbildung 4: Erstellung eines internen Netzwerks in VirtualBox (vgl. [ORAC])

Zunächst wird ein DHCP-Server eingerichtet, damit die IP-Adresse an die VM automatisch vergeben wird. VirtualBox bietet die Möglichkeit, mit der Anwendung über die Kommandozeile mithilfe der Anwendung VBoxManage zu interagieren

Im Command Prompt zu dem Installationsort von VirtualBox navigieren und dort folgenden Befehl ausführen:

```
VBoxManage dhcpserver add --network=Netzwerkname --server-ip=192.168.3.1 --netmask=255.255.255.0 --Lower-ip=192.168.3.2 --upper-ip=192.168.3.254 --Enable.
```

Der Parameter network gibt den Namen des internen Netzwerks an, server-ip ist die Adresse des DHCP-Servers und wird nicht an DHCP-Client vergeben. Lower-ip und upper-ip geben jeweils die geringste nutzbare und höchste nutzbare IP-Adresse an.

Danach könnten mit dem Befehl: **VBoxManage list dhcpservers** die Einstellung des Netzwerks überprüft werden (siehe Abbildung 5) [Übers. durch den Verf.] (vgl. [CYBER21]).

```
D:\VirtualBox>VBoxManage list dhcpservers
NetworkName:      HackerLord
Dhcpd IP:         192.168.3.1
LowerIPAddress:   192.168.3.2
UpperIPAddress:   192.168.3.254
NetworkMask:      255.255.255.0
Enabled:          Yes
Global Configuration:
  minLeaseTime:    default
  defaultLeaseTime: default
  maxLeaseTime:    default
  Forced options:  None
  Suppressed opts.: None
  1/legacy: 255.255.255.0
Groups:           None
Individual Configs: None
```

Abbildung 5: Auflisten des Netzwerks in VirtualBox  
(vgl. [ORAC]; vgl. [CYBER21])

Wenn eine VM mit der Netzwerkeinstellung wie in der Abbildung 4 (S. 9) gestartet wird, bekommt sie eine IP-Adresse zugewiesen.

### 5.1.2. Einrichtung der virtuellen Maschinen

Zwei VM werden in VirtualBox aufgesetzt. Die ISO-Dateien werden von der Webseite von Microsoft (2023) heruntergeladen und damit wurde Windows 10 auf den VM installiert (vgl. [MICRO23]). Jede VM hat einen 6 GB Arbeitsspeicher, 4 Hauptprozessoren, 128 MB Video Speicherplatz und 250 GB

Speicherplatz. Jede VM wird mit der im vorherigen Abschnitt erläuterten Netzwerkeinstellung eingestellt. Die restlichen Einstellungen werden vom Standard in VirtualBox übernommen.

Die erste VM (VM1) wird zur Ausführung der Schadprogramme benutzt. Darin werden Visual Studio, ein Bildverarbeitungsprogramm, ein Webbrowser und ein Mediaplayer installiert.

In VM1 wurden Schadprogramme vom Internet heruntergeladen, die in komprimierten Dateien mit Passwort enthalten sind, damit sie nach dem Herunterladen sich nicht automatisch ausführen. Die Internetseite Malware-Bazaar bietet die Schadprogramme in einem Webserver an, alle Schadprogramme mit dem gleichen Erscheinungsdatum werden in einer komprimierten Datei angeboten (vgl. [ABUSE]). Folgende Dateien wurden in der Arbeit benutzt: 2020-10-22, 2022-07-20 und 2020-11-17. Der Dateiname entspricht dem Erscheinungsdatum im Server.

Die oben genannten Dateien wurden gewählt, weil sie nach Größe die größten im Webserver sind und das Erscheinungsdatum jeder Datei ist in einem bestimmten Zeitintervall von sich unterscheidet, damit Duplikate vermieden werden.

Die zweite VM (VM2) wird zur Speicherung der Ergebnisse der Ausführung benutzt. In der VM2 wartet ein TCP-Server auf die Anfrage von VM1 (siehe 5.3.3). Um die Verbindung von VM1 zu erlauben, muss im Windows Firewall eine Regel für die kommende Verbindung von IP-Adresse der VM1 hinzugefügt werden.

Nach der Einrichtung der VM werden Snapshot von beiden VM aufgenommen.

### 5.1.3. Erstellung der Datenbanktabelle

Eine Datenbank wird auf localhost in der Host-Maschine aufgesetzt. Die Datenbank enthält eine Tabelle zur Speicherung der von Schadprogramm ausgelösten Ereignisse (siehe Tabelle 1 auf nächster Seite).

Name	Datentyp	Bemerkung
id	int	Primärschlüssel
fileName	varchar(200)	Name des Schadprogrammes
sequence	varchar(MAX)	Ereignisse des Schadprogrammes

Tabelle 1: Struktur der Tabelle  
(Eigene Darstellung)

## 5.2. Entwicklung des Programmcodes zur Ausführung von Schadprogrammen

### 5.2.1. Das FASTA-Dateiformat

Das FASTA-Dateiformat (Abkürzung für fast alignment) ist ein textbasiertes Speicherformat in der Bioinformatik für Gensequenzen (vgl. [DANDE21], S.274).

National Center for Biotechnology Information sagt, dass eine FASTA-Datei mehrere Gensequenzen enthalten kann. Eine Sequenz wird mit dem Größer-als-Zeichen anfangen, unmittelbar darauf folgt der Sequenzname. Ab der nächsten Zeile folgen die Sequenzdaten [Übers. durch den Verf.] (vgl. [NCBI]).

Abbildung 6 stellt ein Beispiel für eine FASTA-Datei dar.

```
>Sequenz1
ABECDD
DEMANNEF
EFCCAAAA
>Sequenz2
SATBBEAAAAAMMEEEEEEB
BACSSSSSSSSSSEEEEE
DEFMSLLALALAL
DDMAVLELFSAM
```

Abbildung 6: Ein Beispiel für das Konstrukt einer FASTA-Datei  
(vgl. [NCBI])

In der Arbeit werden die Ereignisse im FASTA-Format in einer textbasierten Datei gespeichert, um die Datei in MSA-Anwendungen benutzen zu können.

### 5.2.2. Programmcode in VM1

In der VM1 wurde ein C#-Projekt in Visual Studio entwickelt, in dem die Schadprogramme ausgeführt und gleichzeitig die Ereignisse dieser Schadprogramme an den TCP-Server in VM2 gesendet werden.

Zuerst wird eine Verbindung mit dem TCP-Server durch die Methode ConnectServer() hergestellt (siehe Abbildung 7).

```
private static bool ConnectServer(String server, int port)
{
    try
    {
        IPEndPoint endPoint = new IPEndPoint(IPAddress.Parse(server), port);
        client = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(endPoint);
        return true;
    }
    catch (Exception e) { Console.WriteLine(n + e.Message); }
    return false;
}
```

Abbildung 7: Methode ConnectServer()  
(vgl. [MICRO23 -b])

Microsoft (2023) sagt, dass das Objekt der Klasse IPEndPoint einen Netzwerkendpunkt als eine IP-Adresse und einer Portnummer darstellt. Der Konstruktor des Objekts nimmt zwei Parameter entgegen: eine String-Variable als IP-Adresse des Servers und eine Integer-Variable als Portnummer des Servers (vgl. [MICRO23 -b]). Das Objekt der Klasse Socket implementiert die Berkeley-Sockets-Schnittstelle (vgl. [MICRO23 -c]). Durch das Aufrufen der Methode Connect() vom Objekt der Klasse Socket wird eine Verbindung mit dem TCP-Server hergestellt (vgl. [MICRO23 -c]).

Nachdem die Verbindung hergestellt wurde, werden alle Schadprogramme in heruntergeladenen Dateien in der Methode StartProcess() ausgeführt (siehe Abbildung 8 auf nächster Seite).

```

private static void StartProcess(String path, String server, int port)
{
    var monitoringTimeSec = 120;
    var allFiles = Directory.EnumerateFiles(path, "*", SearchOption.AllDirectories);
    if (ConnectServer(server, port))
    {
        foreach (var file in allFiles)
        {
            Process process = new Process();
            process.StartInfo = new ProcessStartInfo()
            {
                UseShellExecute = false,
                FileName = file.ToString(),
                CreateNoWindow = true,
                WindowStyle = ProcessWindowStyle.Hidden,
            };

            try
            {
                if (process.Start())
                {
                    SendTxtToServer(">" + file.ToString());
                    Console.WriteLine(p + file.ToString() + " was started with PID: " + process.Id);
                    procID = process.Id;

                    Console.CancelKeyPress += (object sender, ConsoleCancelEventArgs e) =>
                    {
                        if (kernelSession != null)
                            kernelSession.Dispose();
                        e.Cancel = true;
                    };

                    // Set up a timer to stop processing after monitoringTimeSec
                    var timer = new Timer(delegate (object state)
                    {
                        Console.WriteLine(a+"Stopped Monitoring after {0} sec", monitoringTimeSec);
                        if (kernelSession != null)
                        {
                            kernelSession.Dispose();
                        }
                    }, null, monitoringTimeSec * 1000, Timeout.Infinite);

                    using (kernelSession = new TraceEventSession(KernelTraceEventParser.KernelSessionName))
                    {
                        kernelSession.EnableKernelProvider(KernelTraceEventParser.Keywords.All);
                        kernelSession.Source.Kernel.All += LogSyscalls; ;

                        // process events until Ctrl-C is pressed or timeout expires
                        kernelSession.Source.Process();
                    }

                    timer.Dispose(); // Turn off the timer.
                    process.Kill();
                    process.WaitForExit(60000);
                    Console.WriteLine(p + file.ToString() + " finished processing");
                }
                File.Delete(file.ToString());
                Console.WriteLine(p + "File is deleted");
            }
            catch (Exception e){
                Console.WriteLine(n + e.Message);
                continue;
            }
        }
    }
    DisconnectServer();
}

```

Abbildung 8: Methode StartProcess()  
(vgl. [MICRO23 -d]; vgl. [MICRO23 -e]; vgl. [MORRI18]; vgl. [MICRO23 -h])

Durch Directory.EnumerateFiles() werden alle Dateien in einem Ordner gefunden (vgl. [MICRO23 -d]).

Für jede Datei wird ein neuer Prozess erstellt und mit einem Objekt der Klasse ProcessStartInfo konfiguriert. Microsoft beschreibt, dass ProcessStartInfo angibt, dass zum Starten des Prozesses die Betriebssystemshell nicht verwendet werden. Es gibt den Namen der auszuführenden Datei an und macht kein neues Fenster für den Prozess auf. Nachdem die Datei ausgeführt wurde, wird der Prozess-ID in einer globalen Variable gespeichert (vgl. [MICRO23 -e]).

Um die Ereignisse im Kernelmodus von der Datei aufzunehmen, wird ein Objekt der Klasse TraceEventSession benutzt. Robbins (2018) schreibt zu TraceEventSession: „A TraceEventSession represents a single Event Tracing for Windows (ETW) tracing session. A session is an event sink that can enable or disable event logging from **event providers**“ [ROB18]. Der Konstruktor von TraceEventSession nimmt einen Parameter entgegen, er gibt den Namen der Sitzung an (vgl. [ROB18]). KernelTraceEventParsers ist eine Klasse, welche die Ereignisse im Kernelmodus dekodieren und weiterleiten kann [Übers. durch den Verf.] (vgl. [ANDKA23]).

In der Methode StartProcess() wird **event provider** als Kernel des Betriebssystems durch die Methode EnableKernelProvider() festgelegt, der Betriebssystemkern ist der Anbieter, der die Ereignisse im Kernelmodus anbietet.

Robbins (2018) sagt, dass das Objekt TraceEventSession eine Variable namens Source vom Typ ETWTraceEventSource hat, welche die Reihe von Ereignissen darstellt, die innerhalb einem TraceEventSession gesammelt wurden. Durch Initialisierung der Variable Source wird eine ETW-Sitzung eröffnet, und daraus können Ereignisse gesammelt werden. Die Variable Source hat eine Variable namens Kernel vom Typ KernelTraceEventParsers (vgl. [ROB18]). Durch den Aufruf

**kernelSession.Source.Kernel.All += LogSyscalls**

wird eine Echtzeitsitzung eröffnet, die Methode LogSyscalls abonniert alle Ereignisse im Kernelmodus (vgl. [ROB18]).

Mit dem Methodenaufwurf **kernelSession.Source.Process()** wird das TraceEventSession solange weiterlaufen, bis Steuerung und C gedrückt oder den Zeitlimit von zwei Minuten überschritten wird.

Microsoft (2023) sagt, dass Console eine Oberklasse in C# ist, welche die Eingabe durch Tastatur lesen kann, durch den Aufruf Console.CancelKeyPress erkennt das Programm das Drücken der Tasten Steuerung und C. Bei Erfüllung der Bedingung wird die Echtzeitsitzung aufgehoben. Mit einem Objekt der Klasse Timer ist die Auslösung eines Ereignisses nach einem Zeitintervall möglich, der dritte Parameter von Timer gibt das Zeitintervall in Millisekunden an. Wenn das Zeitintervall überschritten ist, wird das Ereignis ausgelöst (vgl. [MICRO23 -g]). Nach zwei Minuten wird die Echtzeitsitzung aufgehoben.

Weiterhin erläutert Microsoft (2023), dass durch Kill() das Programm den aktuellen Prozess terminiert, aber eine sofortige Terminierung wird nicht sichergestellt, weil der Prozess erst vollständig terminiert wird, wenn alle gebundenen Ressourcen freigegeben wurden. Der Zustand der vollständigen Terminierung enthält man durch die Rückgabe von WaitForExit(), wobei der Methodenparameter die Wartezeit für die Terminierung festlegt. Während der Wartezeit wartet das Programm auf die Terminierung, statt den nächsten Befehl auszuführen (vgl. [MICRO23 -h]).

In der Arbeit wurde die Wartezeit auf 1 Minute konfiguriert, weil die Menge der zu analysierenden Dateien sehr groß war.

Nachdem alle Dateien analysiert wurden, ruft StartProcess() die Methode DisconnServer() auf, um die Verbindung mit dem TCP-Server zu trennen (siehe Abbildung 9). Laut Microsoft (2023) deaktiviert der Methodenaufwurf Shutdown() mit dem Parameter „SocketShutdown.Both“ das Senden und Empfangen für einen Socket. Close() schließt die Socket-Verbindung und gibt alle gebundenen Ressourcen frei. (vgl. [MICRO23 -i])

```
private static void DisconnServer()
{
    client.Shutdown(SocketShutdown.Both);
    client.Close();
}
```

Abbildung 9: Methode DisconnServer() (vgl. [MICRO23 -i])

```

private static void LogSyscalls(TraceEvent @event)
{
    if (@event.ProcessID == procID && procID != -1)
    {
        try { SendTxtToServer(@event.EventName); }
        catch (Exception e) { Console.WriteLine(n + e.ToString()); }
    }
}

```

Abbildung 10: Methode LogSyscalls()  
(vgl. [KURDI23])

Obwohl in der Methode StartProcess() kein Parameter an die Methode LogSyscalls() übergeben wird, verfügt LogSyscalls() über eine lokale Variable der Klasse TraceEvent, die ein Ereignis repräsentiert (siehe Abbildung 10). Die Variable enthält u.a. die Prozess-ID von dem Prozess, welcher das Ereignis auslöst, die Prozess-ID wird mit der globalen Variable procID verglichen, um nur das Ereignis des aktuellen Prozesses aufzunehmen. Landwerth erklärt, dass das Objekt der Klasse TraceEvent eine Variable EventName enthält, welche der Ereignisname ist. EventName fängt mit dem Namen der Gruppe an, gefolgt von einem Schrägstrich, danach folgt der Ereignisname wie z.B. ProcessTask/ProcessStart (vgl. [LAND14]). Die Methode LogSyscalls() gibt der Methode SendTxtToServer() den Ereignisnamen als Parameter über.

```

private static async void SendTxtToServer(String text)
{
    try
    {
        var msgBytes = Encoding.UTF8.GetBytes(text);
        var newLine = Encoding.UTF8.GetBytes("\n");
        _ = await client.SendAsync(msgBytes, SocketFlags.None);
        _ = await client.SendAsync(newLine, SocketFlags.None);
    }
    catch (Exception e) {
        Console.WriteLine(n + e.Message);
        Environment.Exit(0);
    }
}

```

Abbildung 11: Methode SendTxtToServer()  
(vgl. [MICRO23 -j])

Die Methode SendTxtToServer() kodiert den Text in Byte und ruft die Methode SendAsync() vom bereits initialisierten Objekt der Klasse Socket auf, um den Text zum Server zu senden (siehe Abbildung 11) (vgl. [MICRO23 -j]). Danach wird eine Kodierung für eine neue Zeile gesendet, damit der Server den

nächsten Text in einer neuen Zeile in die Textdatei schreibt. Wenn eine Ausnahme in dieser Methode vorkommt, wird mit der Methodenaufwurf Environment.Exit(0) das Programm beendet (vgl. [MICRO23 -k]).

### 5.2.3. Programmcode in VM2

In der VM2 wird ein C#-Projekt im Visual Studio programmiert, in dem ein Server mit TCP-Protokoll auf die Anfrage von dem TCP-Client in VM1 wartet. Der TCP-Server wird in der Methode StartTcpServer() aufgebaut (siehe Abbildung 12 auf nächster Seite).

```
private static void StartTcpServer(int port, String saveFileName)
{
    TcpListener listener = null;
    try
    {
        listener = new TcpListener(IPAddress.Any, port);
        listener.Start();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    byte[] recData = new byte[1024];
    int recBytes;
    while (true)
    {
        TcpClient client = null;
        NetworkStream netstream = null;
        try
        {
            client = listener.AcceptTcpClient();
            netstream = client.GetStream();
            Console.WriteLine("Received connection from client");
            int totalRecBytes = 0;
            using (FileStream fs = new FileStream(saveFileName, FileMode.Append, FileAccess.Write))
            {
                while ((recBytes = netstream.Read(recData, 0, recData.Length)) > 0)
                {
                    fs.Write(recData, 0, recBytes);
                    totalRecBytes += recBytes;
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        finally
        {
            netstream.Close();
            client.Close();
        }
    }
}
```

Abbildung 12: Methode StartTcpServer()  
(vgl. [PINE22]; vgl. [MICRO23 -m])

Microsoft (2023) sagt, dass die Bibliothek System.Net.Sockets von .NET Windows Socket implementiert und Funktionen anbietet, wodurch Programmierende das Netzwerk steuern können. U.A stehen die Klassen TcpClient, TcpServer und NetworkStream zur Verfügung (vgl. [MICRO23 -l]).

Pine, Schonning und Aksoy (2022) beschreibt in einem Artikel die Klassen TcpListener und TcpClient. Ein Objekt der Klasse TcpListener startet einen Tcp-Server und wartet auf die Anfrage vom Client. Im Konstruktor vom TcpListener wird der erste Parameter gleich IPAddress.Any initialisiert, damit der Server Anfragen von allen Netzwerkschnittstelle akzeptiert. Der zweite Parameter ist die Portnummer, auf die der Server wartet. In einer endlosen While-Schleife wird versucht, eine Verbindung von einem TCP-Client zu akzeptieren. Ein Objekt der Klasse TcpClient repräsentiert einen TCP-Client und ein Objekt der Klasse NetworkStream ermöglicht den Zugriff auf die vom Client gesendeten Datenstrom [Übers. durch den Verf.] (vgl. [PINE22]).

Microsoft (2023) sagt, dass mit der Klasse FileStream das Lesen und Schreiben einer Textdatei unterstützt ist. Der Konstruktor von FileStream nimmt drei Parameter entgegen, der erste gibt den Dateinamen an, der zweite gibt den Modus an, in diesem Fall ist er gleich „Append“, damit wird der Inhalt der Datei erweitert (vgl. [MICRO23 -m]). Nach einem Neustart des Projekts wird die Log-Datei um die neuen ankommenden Texte erweitert. Solange der Client Daten an Datenstrom sendet, liest das Objekt der Klasse NetworkStream die Daten und das Objekt der Klasse FileStream speichert die Daten in einer Textdatei.

### 5.3. Ausführung der Projekte

Nachdem die TCP-Client und TCP-Server implementiert wurden, werden die Projekte gestartet. Zuerst wird das Projekt in VM2 im Visual Studio ausgeführt, damit der TCP-Server bereits auf eine Anfrage von TCP-Client wartet. Vor dem Start des Projekts in VM1 müssen Windows Firewall und Windows Defender deaktiviert werden, damit der Sicherheitsmechanismus von Windows die Schadprogramme nicht hindern [Übers. durch den Verf.] (vgl. [HACKER19]). Darüber hinaus werden ein Bild in einem Bildbearbeitungsprogramm und zwei Textdateien im Textverarbeitungsprogramm geöffnet.

## 5.4. Kodierung der Ereignisse

In der Forschung von Ki et al. (2015) wurden die API-Aufrufe nach Funktionalität gruppiert. Die Aufrufe in derselben Gruppe werden mit der gleichen Kodierung in der Datenbank abgespeichert. Für jede Gruppe wird ein Zeichen als Kodierung gegeben (vgl. [K115]). In der Klasse KernelTraceEventParsers hat andkamashev (2023) die Ereignisse nach Funktionalität gruppiert. Jede Gruppe hat u.a. einen Namen und eine GUID (siehe Abbildung 13 auf nächster Seite) (vgl. [ANDKA23]).

```
public static readonly string ProviderName = "Windows Kernel";
public static readonly Guid ProviderGuid = new Guid(-1635693907, 12804, 4562, 154, 130, 0, 96, 8, 168, 105, 57);
private static volatile TraceEvent[] s_templates;
internal static readonly Guid EventTraceTaskGuid = new Guid(1761466624, 19006, 4561, 132, 244, 0, 0, 248, 4, 100, 227);
internal static readonly Guid ProcessTaskGuid = new Guid(1030727888, -507, 4560, 157, 218, 0, 192, 79, 215, 186, 124);
internal static readonly Guid ThreadTaskGuid = new Guid(1030727889, -507, 4560, 157, 218, 0, 192, 79, 215, 186, 124);
internal static readonly Guid DiskIOTaskGuid = new Guid(1030727892, -507, 4560, 157, 218, 0, 192, 79, 215, 186, 124);
internal static readonly Guid RegistryTaskGuid = new Guid(-1370262994, -14237, 4562, 134, 89, 0, 192, 79, 163, 33, 161);
internal static readonly Guid SplitIoTaskGuid = new Guid(-667432302, 4793, 17573, 173, 106, 58, 101, 179, 87, 138, 168);
internal static readonly Guid FileIOTaskGuid = new Guid(-1865688007, 19006, 4561, 132, 244, 0, 0, 248, 4, 100, 227);
internal static readonly Guid TcpIpTaskGuid = new Guid(-1708651840, -14112, 4561, 132, 226, 0, 192, 79, 185, 152, 162);
internal static readonly Guid UdpIpTaskGuid = new Guid(-1086697275, -22071, 18824, 160, 5, 45, 240, 183, 200, 15, 128);
internal static readonly Guid ImageTaskGuid = new Guid(749821213, 24513, 4562, 171, 225, 0, 160, 201, 17, 245, 24);
internal static readonly Guid MemoryTaskGuid = new Guid(1030727891, -507, 4560, 157, 218, 0, 192, 79, 215, 186, 124);
internal static readonly Guid MemoryProviderGuid = new Guid(-774291721, -7694, 20293, 153, 67, 3, 210, 69, 254, 108, 0);
internal static readonly Guid PerfInfoTaskGuid = new Guid(-836911180, 4990, 19878, 135, 176, 63, 89, 170, 16, 44, 188);
internal static readonly Guid StackWalkTaskGuid = new Guid(-554500538, 31702, 19328, 189, 148, 245, 127, 226, 13, 12, 227);
internal static readonly Guid EventTracingProviderGuid = new Guid(-1233785801, -16970, 17992, 188, 146, 243, 253, 199, 77, 60, 162);
internal static readonly Guid ALPCTaskGuid = new Guid(1171836109, 21407, 19314, 168, 183, 92, 104, 49, 66, 96, 154);
internal static readonly Guid LostEventTaskGuid = new Guid(1782160096, 19398, 19945, 135, 11, 54, 87, 248, 148, 126, 126);
```

Abbildung 13: Ereignissen im Kernelmodus nach Gruppe  
[ANDKA23]

In dieser Klasse hat andkamashev (2023) nach der Erstellung der Gruppen jedes Ereignis einer Gruppe zugeordnet. In Abbildung 14 auf nächster Seite wird das Ereignis der Gruppe EventTraceTask zugeordnet.

```

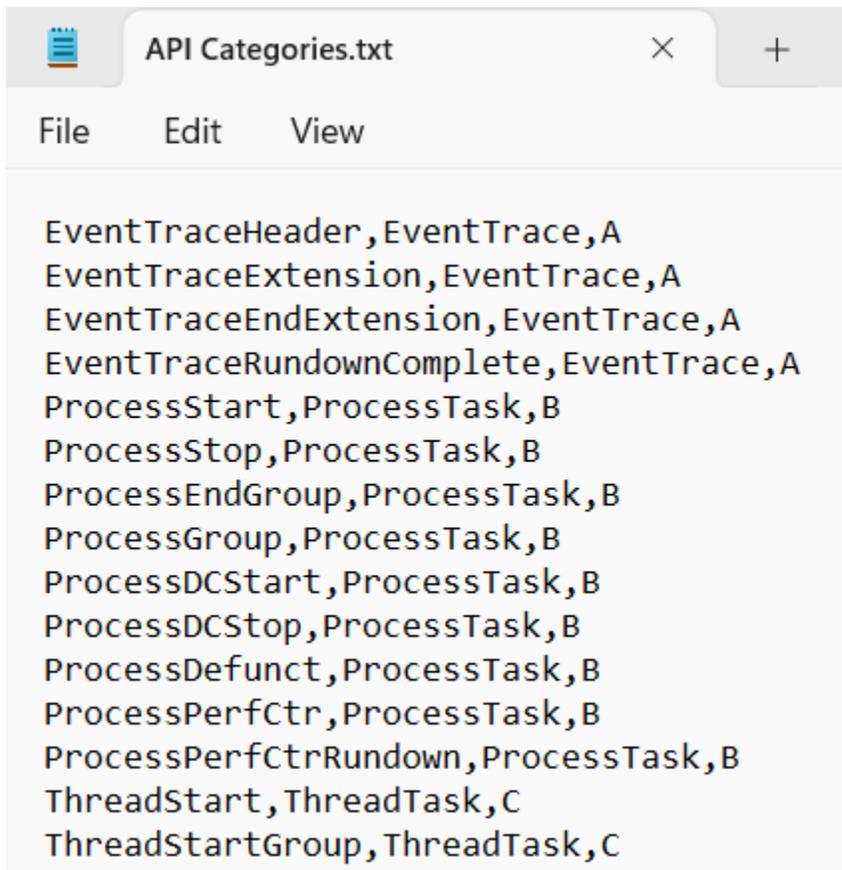
public event Action<EventTraceHeaderTraceData> EventTraceHeader
{
    add
        source.RegisterEventTemplate(new EventTraceHeaderTraceData(value, 65535, 0, "EventTrace", EventTraceTaskGuid, 0, "Header",
    remove
        source.UnregisterEventTemplate(value, 0, EventTraceTaskGuid);
}

```

Abbildung 14: EventTraceHeader  
[ANDKA23]

Insgesamt bietet die Bibliothek TraceEvent 193 Ereignisse an, sie werden in 22 Gruppen aufgeteilt (vgl. [ANDKA23]).

Anhand der Gruppierung wird eine textbasierte Datei zur Speicherung des Codes erstellt. Die Daten werden manuell in csv-Format erfasst. Abbildung 15 auf nächster Seite stellt einen Abschnitt der Datei dar. Die Datei wird „API Categories“ genannt. Die erste Spalte enthält den Ereignisnamen. Der Ereignisname besteht aus dem Gruppennamen, gefolgt von dem Ereignisnamen wie z.B. bei ProcessStart ist Process der Gruppename und Start der Ereignisname. Die zweite Spalte gibt den Gruppennamen an, welcher dem Ereignis zugeordnet ist. Die letzte Spalte enthält der Gruppencode. Die Gruppencodes werden alphabetisch von A bis Z entsprechend der Reihenfolge des Auftritts der Gruppe in der Klasse KernelTraceEventParsers vergeben (vgl. [ANDKA23]).



```
EventTraceHeader,EventTrace,A
EventTraceExtension,EventTrace,A
EventTraceEndExtension,EventTrace,A
EventTraceRundownComplete,EventTrace,A
ProcessStart,ProcessTask,B
ProcessStop,ProcessTask,B
ProcessEndGroup,ProcessTask,B
ProcessGroup,ProcessTask,B
ProcessDCStart,ProcessTask,B
ProcessDCStop,ProcessTask,B
ProcessDefunct,ProcessTask,B
ProcessPerfCtr,ProcessTask,B
ProcessPerfCtrRundown,ProcessTask,B
ThreadStart,ThreadTask,C
ThreadStartGroup,ThreadTask,C
```

Abbildung 15: Gruppencode der Ereignisse im Kernelmodus (vgl. [ANDKA23])

Um die Kodierung im C#-Projekt verwenden zu können, wird die Datei durch die Methode `readSyscallEncoding()` eingelesen (siehe Abbildung 16 auf nächster Seite). Die Kodierung wird in einem Dictionary gespeichert. Dictionary ist eine Datenstruktur in C#, der eine Liste von Einträgen enthält, jeder Eintrag besteht aus einem Schlüssel und einem Wert, wobei ein Schlüssel nur einmal im Dictionary vorkommen darf (vgl. [MICRO23 -n]).

Eine globale Variable zur Speicherung der Kodierung wird deklariert und initialisiert durch den Befehl:

```
static Dictionary<String, List<String>> syscallEncodeDict = new  
Dictionary<String, List<String>>();
```

Der Schlüssel ist der Code. Alle zu dem Code gehörenden Ereignisse werden in einer Liste gespeichert, die Liste wird als Wert des Schlüssels eingetragen. Die

Variable wird global deklariert und initialisiert, weil es später während der Duplikate Entfernung darauf zugegriffen wird (siehe Abschnitt 5.5).

```
public static void readSyscallEncoding(String path)
{
    String currCode = "";
    List<String> a = null;
    using (var reader = new StreamReader(path))
    {
        while (!reader.EndOfStream)
        {
            var line = reader.ReadLine();
            var values = line.Split(',');
            String code = values[2];
            String syscall = values[0];

            if (currCode.Equals(""))
            {
                a = new List<String>();
                currCode = code;
                a.Add(syscall);
            }
            else if (currCode != code)
            {
                syscallEncodeDict.Add(currCode, a);
                currCode = code;
                a = new List<String>();
                a.Add(syscall);
            }
            else if (currCode == code)
                a.Add(syscall);
        }
        if (a.Count != 0)
        {
            syscallEncodeDict.Add(currCode, a);
        }
    }
}
```

Abbildung 16: Methode readSyscallEncoding()  
(vgl. [MICRO23 -p]; vgl. [MICRO23 -o]; vgl. [MICRO23 -n])

Die Methode `readSyscallEncoding()` nimmt einen Parameter entgegen, er gibt den csv-Dateipfad an, welche durch `StreamReader` gelesen wird. Die lokale Variable `currCode` speichert den zuletzt gelesenen Code. Eine lokale Variable vom Typ `List` speichert alle Ereignisse einer Gruppe. In der `While`-Schleife wird die Datei zeilenweise durch den Methodenaufruf `ReadLine()` gelesen. Microsoft (2023) sagt, wenn die Methode **`Split(,)`** auf einen String angewendet wird, wird der String in mehrere String anhand des Kommazeichens geteilt. Die Methode gibt ein Array vom Typ `String` zurück (vgl. [MICRO23 -o]). Die Position des Teilstrings entspricht seiner Auftrittsreihenfolge im originalen String. In Hinsicht auf die csv-Datei enthält Index 0 des Arrays das Ereignis und Index 2 den Code.

Beim ersten Durchlauf der `While`-Schleife wird durch die `If`-Schleife die Liste initialisiert. Der Wert des zuletzt gelesenen Codes ist gleich dem Code der aktuellen Zeile und das Ereignis wird in die Liste hinzugefügt. Ab dem zweiten Durchlauf wird der Code der aktuellen Zeile mit dem zuletzt gelesenen Code verglichen. Bei Übereinstimmung der beiden Codes wird das Ereignis der aktuellen Zeile in die bereits initialisierte Liste hinzugefügt. Ansonsten werden der vorherige Code und seine Liste in das Dictionary hinzugefügt. Dann wird eine neue Liste für den neuen Code erstellt, der Wert des zuletzt gelesenen Codes gesetzt und das Ereignis in die Liste hinzugefügt.

Beim Erreichen des Dateiende wird die Liste der Ereignisse auf vorhandene Elemente geprüft. Ist in der Liste Element vorhanden, werden sie und der zuletzt gelesene Code in das Dictionary hinzugefügt.

### 5.5. Entfernung der duplizierte Zeilen

Nachdem die Schadprogramme komplett ausgeführt wurden, werden in der gespeicherten FASTA-Datei Duplikate entfernt und die Ereignisse werden kodiert. Jedes Ereignis, das mehrmals nacheinander vorkommt, sein erster Auftritt in der Reihenfolge wird beibehalten, die restlichen Auftritte werden gelöscht. Ein anschauliches Beispiel dafür stellt die Abbildung 17 auf nächster Seite dar. Das Zeichen B kommt viermal nacheinander vor, der erste Auftritt in der Zeile 3 wird beibehalten, die Zeichen in den Zeilen 4,5 und 6 werden gelöscht. Das gleiche Verfahren gilt für das Zeichen D in der Zeile 9.

>file1.exe	>file1.exe
A	A
B	B
B	C
B	E
B	D
C	G
E	
D	
D	
G	
Originale Datei	Nach der Duplikatentfernung

Abbildung 17: Duplikatentfernung in der FASTA-Datei  
(Eigene Darstellung)

Das Verfahren zur Duplikate Entfernung wird durch die Methode `removeDupLines()` abgewickelt (siehe Abbildung 18 auf S. 17). Die Methode nimmt drei Parameter entgegen, der erste Parameter gibt den originalen Dateipfad an, der zweite Parameter gibt den Zieldateipfad an, wo das Ergebnis des Verfahrens gespeichert wird. Der letzte Parameter gibt den Pfad zu der Kodierung Datei an, wo die Informationen zur Kodierung der Ereignisse gespeichert sind.

Jede Log-Datei wird mithilfe von `StreamReader` gelesen. `StreamReader` ist eine Klasse, die eine Zeichenreihe lesen kann (vgl. [MICRO23 -p]). Um eine Datei mit `StreamReader` zu lesen, muss der Dateiname als Parameter an `StreamReader` übergeben werden (vgl. [MICRO23 -p]).

Zuerst wird die Methode `readSyscallEncoding()` aufgerufen, um die Codes in die globale Variable vom Typ Dictionary zu speichern. Zunächst wird über alle Log-Dateien iteriert. Jede Datei wird mit dem `StreamReader` zeilenweise gelesen. Die Variable `lineNo` enthält die Zeilennummer und wird bei jeder Iteration um eins erhöht. Die zuletzt gelesene Zeile wird in der Variable `currLine` gespeichert, die Variable `line` enthält die aktuelle Zeile. Die Methode `Peek()` von `StreamReader` liest den zunächst verfügbare Zeile in der Datei (vgl. [MICRO23 -p]). Jede Zeile wird durch `String.IsNullOrEmpty()` geprüft, ob sie Zeichen enthält (vgl. [MICRO23 -q]). Wenn die Zeile Zeichen enthält, wird das erste Zeichen der Zeile geprüft, ob es gleich das Größer-als-Zeichen ist (vgl. [MICRO23 -r]). Wenn es zutrifft, wird der Name der Datei aus der Zeile ausgewählt.

Jeder Dateiname in der Log-Datei ist in Form eines vollständigen Pfads gespeichert, weil der Dateiname durch **SendTxtToServer(">" + file.toString())** aufgerufen wurde (vgl. [MICRO23 -d]). Um der Dateiname daraus zu entnehmen, wird ein Substring erstellt, das Substring enthält alle Zeichen vom ersten Index nach dem letzten Auftritt des Schrägstrichzeichens bis zum letzten Index der String (vgl. [MICRO23 -r]). z.B. wird aus >C:\Users\Documents\file.txt ein neues String file.txt. Der Substring wird durch das Stream Writer in die neue Datei geschrieben (vgl. [MICRO23 -r]).

Wenn die Zeile nicht einen Dateinamen enthält, wird sie mit der zuletzt gelesenen Zeile verglichen. Wenn die Zeilen sich unterscheiden, wird der Ereignisname aus dem String gebildet. Durch die Methode getCode() wird der Code des Ereignisnamen gefunden (siehe Abbildung 19 auf S. 28). Über jeden Eintrag vom Dictionary wird in einer Foreach-Schleife iteriert. Weil der Wert des Eintrags eine Liste ist, kann darüber in einer Foreach-Schleife iteriert werden. Nachdem der Ereignisname gefunden wurde, wird der Gruppencode zurückgegeben.

Wenn für den Ereignisnamen kein Code gefunden wird, werden die ganze aktuelle Zeile, den Dateiname und die Zeilennummer über die Konsole ausgegeben. Die Informationen dienen zur Korrektur der Zeile. Anschließend wird das Programm beendet.

Vor dem Ende jedes Durchlaufs wird die zuletzt gelesene Zeile gleich die aktuelle Zeile gesetzt.

Abbildung 20 auf S. 28 stellt ein anschauliches Beispiel einer Log-Datei nach diesem Schritt dar.

```

private static void removeDupLines(String logPath, String dest, String code)
{
    readSyscallEncoding(code);

    var allFiles = Directory.EnumerateFiles(logPath, "*", SearchOption.AllDirectories);

    StreamWriter sw = new StreamWriter(dest);

    foreach (var file in allFiles)
    {
        StreamReader sr = new StreamReader(file.ToString());
        int lineNo = 0;
        String currLine = "";
        String line;
        while (sr.Peek() >= 0)
        {
            line = sr.ReadLine();
            lineNo++;
            if (!String.IsNullOrEmpty(line))
            {
                if (line.StartsWith(">"))
                {
                    String name = line.Substring(line.LastIndexOf("\\") + 1);
                    sw.WriteLine(">" + name);
                }
                else
                {
                    if (!line.Equals(currLine))
                    {
                        try
                        {
                            int index = line.LastIndexOf("/");
                            String syscall = line.Remove(index, 1);
                            String syscallCode = getCode(syscall);
                            if (syscallCode.Equals(""))
                            {
                                Console.WriteLine(n + "cannot find encoding for syscall: " + line);
                                Console.WriteLine(n + "Error at file: " + file.ToString());
                                Console.WriteLine(n + "Error at line number: " + lineNo);
                                sw.Close();
                                sr.Close();
                                Environment.Exit(0);
                            }
                            sw.WriteLine(syscallCode);
                        }
                        catch (Exception e)
                        {
                            Console.WriteLine(e.Message);
                            Console.WriteLine(n + "Error at file: " + file.ToString());
                            Console.WriteLine(n + "Error at line number: " + lineNo);
                        }
                    }
                    currLine = line;
                }
            }
            sr.Close();
        }
        sw.Close();
    }
}

```

Abbildung 18: Duplikatentfernung mit der Methode removeDupLines() (vgl. [MICRO23 -d]; vgl. [MICRO23 -r]; [MICRO23 -p]; vgl. [MICRO23 -q])

```

private static String getCode(String syscall)
{
    foreach (KeyValuePair<String, List<String>> pair in syscallEncodeDict)
    {
        foreach (String ele in pair.Value)
        {
            if (ele.Equals(syscall))
                return pair.Key;
        }
    }
    return "";
}

```

Abbildung 19: Methode getCode()  
(vgl. [MICRO23 -n])

>file1.exe	>file1.exe
Process/Stop	B
Process/EndGroup	B
Process/Stop	B
Process/Stop	A
Process/Stop	A
EventTrace/Header	A
EventTrace/EndExtension	
EventTrace/Header	
<b>Vorher</b>	<b>Nachher</b>

Abbildung 20: Ergebnis der Entfernung von duplizierten Zeilen  
(vgl. [ANDKA23])

## 5.6. Entfernung der duplizierten Einträge in Log-Datei

Duplizierte Einträge existieren in der Log-Datei. Ein Grund dafür ist, dass ein Schadprogramm nach der Ausführung nicht gelöscht werden kann, weil der Prozess innerhalb einer Minute sich nicht terminieren lässt. Das liegt daran, dass die damit verbundenen Ressourcen sich nicht freigeben lassen. Der Mechanismus von Windows verhindert das Löschen einer Datei, während sie noch ausgeführt ist. Deswegen wird das Schadprogramm beim nächsten Durchlauf noch einmal ausgeführt.

Um die duplizierten Einträge in der Log-Datei zu löschen, wird jeder Eintrag als eine Zeile in eine Datenbanktabelle hinzugefügt. Die Tabelle wird mit folgendem SQL-Befehl erstellt:

```
CREATE TABLE malware_api(  
    id INT NOT NULL IDENTITY PRIMARY KEY,  
    fileName VARCHAR(200) NOT NULL,  
    sequence VARCHAR(max) NOT NULL,  
);
```

Die Methode `addSyscallToServer()` fügt die Daten aus der Log-Datei in die Datenbanktabelle hinzu. (siehe Abbildung 21 auf nächster Seite). Microsoft beschreibt, dass durch Initialisierung eines Objekts vom Typ `SqlConnection` eine Verbindung zu der Datenbank aufgebaut wird. Die Informationen der Datenbank werden als Parameter in den Konstruktor übergeben. `Data Source` gibt den Namen der Datenbank-Instanz an, `Initial Catalog` gibt den Tabellennamen an, `User ID` und `Password` sind die Anmeldedaten. Der Methodenaufruf `Open()` des Objekts öffnet die Verbindung. Der SQL-Befehl wird in eine String-Variable gespeichert (vgl. [MICRO23 -r]).

Die Variable `header` speichert den Dateinamen und die `sequence` enthält die Aufrufe. Wenn die aktuelle Zeile einen Dateinamen enthält und `header` auch einen Wert enthält, dann wird eine Datei vollständig durchgelesen. Der zuletzt gelesene Dateiname wurde in `header` gespeichert. Ein Objekt der Klasse `SqlCommand` wird für die Ausführung des SQL-Befehls verwendet (vgl. [MICRO23 -s]). Microsoft legt fest, dass der SQL-Befehl „VALUES (@fileName,@sequence)“ enthält. Das `@`-Zeichen wird vor dem Spaltennamen gesetzt, damit durch den Methodenaufruf `sqlComm.Parameters.Add()` der Wert in die entsprechende Spalte hinzugefügt wird. Durch `ExecuteNonQuery()` wird der SQL-Befehl ausgeführt (vgl. [MICRO23 -t]).

Nachdem die zuletzt gelesene Datei und ihre Ereignisse hinzugefügt wurden, wird der zuletzt gelesene Dateiname gleich der aktuellen Zeile gesetzt und der Aufruf gleich leerer Text gesetzt, weil für die Datei noch kein Ereignis gelesen wird, sondern erst ab dem nächsten Durchlauf. Anschließend wird der restliche Code der While-Schleife mit `continue` übersprungen.

Wenn die aktuelle Zeile einen Dateinamen enthält und header ein leeres String ist, wird header gleich der aktuellen Zeile gesetzt, dies trifft nur im ersten Durchlauf der While-Schleife zu (vgl. [MICRO23 -u]). Wenn die Zeile einen Aufruf enthält, wird die String-Variable zum Speichern der Aufrufe um die aktuelle Zeile erweitert.

Die If-Schleife nach der Do-While-Schleife fügt der zuletzt gelesene Dateiname und die Ereignisse in die Datenbank hinzu.

```

public static void addSyscallToServer(String path, String tabName, SqlConnection conn)
{
    try
    {
        var allFiles = Directory.EnumerateFiles(path, "*", SearchOption.AllDirectories);
        foreach (var file in allFiles)
        {
            using (StreamReader sr = new StreamReader(file.ToString()))
            {
                string command = "INSERT into " + tabName + " VALUES (@col1,@col2)";
                String header = "";
                String sequence = "";

                do
                {
                    String line = sr.ReadLine();
                    if (line.StartsWith(">") && !header.Equals(""))
                    {
                        using (SqlCommand sqlComm = new SqlCommand(command, conn))
                        {
                            sqlComm.Parameters.AddWithValue("@col1", header);
                            sqlComm.Parameters.AddWithValue("@col2", sequence);
                            sqlComm.ExecuteNonQuery();
                            header = line;
                            sequence = "";
                            continue;
                        }
                    }
                    if (line.StartsWith(">"))
                        header = line;
                    else
                        sequence += line;
                } while (!sr.EndOfStream);
                if (sr.EndOfStream)
                {
                    using (SqlCommand sqlComm = new SqlCommand(command, conn))
                    {
                        sqlComm.Parameters.AddWithValue("@col1", header);
                        sqlComm.Parameters.AddWithValue("@col2", sequence);
                        sqlComm.ExecuteNonQuery();
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}

```

Abbildung 21: Methode addSyscallToServer()  
(vgl. [MICRO23 -d]; vgl. [MICRO23 -p]; vgl. [MICRO23 -r]; vgl. [MICRO23 -u]; vgl. [MICRO23 -w])

Die Abbildung 22 auf nächster Seite stellt einen Abschnitt des Ergebnisses des Methodenaufrufs addSyscallToServer() in der Datenbank dar.

	id	fileName	sequence
1	2716	>000aba05d34281fa9a509e464d2c4ac3a2c94bff16928b73723abe274f516d98.exe	KKKGDGKKKGGGGKDBCGJGJGJGJGJGJGJGJGJGDCGKKMEGGKEEEK...
2	2717	>001006ecef6c369b3792886c8c961b6b3e3e8baa75388e91f52e2aba7fe6441.exe	OOOOEEEEEEEECOOCRKKCOOCOOCMCRKRKRMRGGGGGGGGGGKJK...
3	2718	>0013caaf2c495bfe6e3db2e0c5bead8fd41247e462f4218997eac64e82a302.exe	GDGKGDGKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK...
4	2719	>0024c6287f1047464cab4b7a210146f51e40f4ab4a041cac8c4af340e8d03afe.exe	EGGJEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEGGGGGGGGG...
5	2720	>0026743bd6479b1db7476326a2abb56e48e9fe6f74574a112799373ec8ea05cc.exe	OOOOOGGGGGGGGGGGGGGGGGGGGGGGEGJEEEEEEEEEEEEOOOOOOO...
6	2721	>002a25d22717f4c57eb76e117bf6e2598a3123160e34b4d242b896fe798da967.exe	GGGGGGKRRKRGDKKGGGGKGGKGGKGGKGGKGGKGGKGGKGGKGGKGGK...
7	2722	>002c3dfe6c2a402993f53e080caef084f6ddc31101aed77f3408b62e24b19afb.exe	KKKGDGKKGDKKRRKJKKRRKKKKKGRGKRKGKCKCKMKKGGGGGGG...
8	2723	>002f4285b3f478a6abd8eef22642b01e7c32a23656a4133828fe9c080dbded34.exe	KRKCCEERKMGGKGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG...
9	2724	>003355f19b740987d938a41e29c9d34ab1d70ac82e0b5999d8d0ab26486a4410.exe	KEGGKEEGGKRKEEEEEKGGEEEEEEEEKGGGGGGGGGGGGKJKKKKK...
10	2725	>00473a2f192785f6807389d6dfc0c9edebc9143a92302cec1cbf911ec4eafbd.exe	BCGJGJGJGJGJGJGDCCKGGKJKKKKKGMCKJKKKKKGGCOOGKKKKM...
11	2726	>0054c46caefae4eb027404216d953121cca4f5894fe5cb0a306976a68ce8467e.exe	OOOORRRKRGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG...
12	2727	>005b07f4994dac28c9c730744e73a4fde81ce34bd642f6835985871a2fb0084.exe	JKKKGOOOGKKKGGEGGKEGGKRRKEEEEEKGGEEEEEEEEKGGGGGG...
13	2728	>006843ca7be54a267c70bb5457b142d4a8853319edde1eedea5fc40a8b9149.exe	KRKORKRKCOCMOKRKKOCKKCKRKOCCORKOOCOOCOOCOORRKOOR...
14	2729	>0079ec8aea31765f622c69ec1321fcd6792d99ec0d733df83918ef4cb060152.exe	KKGDGKKKKKKKKKKKKKKKKKKKKKBCGJGJGJGJGJGJGJGJGJGJG...
15	2730	>00856da0582101cd2f62828eb6ab274860b9693dc2fe44f8db1e5926bcbcf0ba.exe	KKGDGKKGKJKKKKKKKKKKJKKKKGGKJKKKKKKKKKKGGKJKKKKK...

Abbildung 22: Tabelle zur Speicherung von Ereignissen (vgl. [ABUSE])

Die Duplikate wurden mit dem folgenden SQL-Befehl in SQL Server gelöscht:

```
DELETE T
FROM
    (
        SELECT *,
            DupRank = ROW_NUMBER() OVER (
                PARTITION BY CAST(sequence AS VARCHAR(max))
                ORDER BY (SELECT NULL)
            )
        FROM malware_api
    ) AS T
WHERE DupRank > 1. (vgl. [HAI23])
```

Nachdem die duplizierte Einträge gelöscht wurden, wird mit der Methode readFromDB() die Tabelleneinträge in eine textbasierte Datei gespeichert, um die Datei später in die MSA-Anwendungen zu importieren (siehe Abbildung 23 auf nächster Seite).

```

public static void readFromDB(String dest, SqlConnection connection, String tableName)
{
    try
    {
        using (SqlCommand command = new SqlCommand("select * from "+tableName, connection))
        {
            using (SqlDataReader reader = command.ExecuteReader())
            {
                using (StreamWriter writer = new StreamWriter(dest))
                {
                    while (reader.Read())
                    {
                        writer.WriteLine(reader["fileName"].ToString());
                        String sequence = reader["sequence"].ToString();
                        foreach (char s in sequence)
                        {
                            writer.WriteLine(s);
                        }
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

Abbildung 23: Methode readFromDB()  
(vgl. [MICRO23 -w]; vgl. [KAM18]; vgl. [MICRO23 -t])

Kamrushi beschreibt in einem Artikel, dass eine Datenbanktabelle mit der Klasse SqlDataReader zeilenweise gelesen werden kann. Durch den Aufruf `reader["Spaltenname"].ToString()` wird der Wert in String konvertiert (vgl. [KAM18]). Alle Ereignisse werden in einer String-Variable gespeichert, jedes Zeichen (character oder char) der Variable kann durch die ForEach-Schleife entnommen werden.

## 5.7. Anwendung des MSA-Algorithmus auf die Daten

Zur Anwendung des MSA-Algorithmus auf die Daten werden drei verschiedene Anwendungen benutzt.

Es wird zuerst die Anwendung MAFFT von Osaka University und RIMD(2023) benutzt. In Windows ist die Anwendung als interaktive Kommandozeile zu benutzen.

MAFFT von Osaka University und RIMD (2023) leitet den Benutzer durch den Verarbeitungsprozess an. Zuerst muss der Dateiname im Installationsort von MAFFT gespeichert werden. Dann muss der Dateiname in der Kommandozeile angegeben. Zunächst muss der Ausgabedateiname angegeben werden. MAFFT legt die Ausgabedatei automatisch an. Danach wird nach dem Ausgabedateiformat gefragt. Die Ausgabedatei ist im FASTA-Format und behält die inhaltliche Reihenfolge der importierten Datei (siehe Abbildung 24 auf nächster Seite).

```
C:\WINDOWS\system32\cmd. x + v
Active code page: 65001

Preparing environment to run MAFFT on Windows.
This may take a while, if real-time scanning by anti-virus software is on.

It may take a while before the calculation starts
if being scanned by anti-virus software.
Also consider using a faster version for Windows 10:
https://mafft.cbrc.jp/alignment/software/wsl.html

-----

MAFFT v7.520 (2023/Mar/22)

MBE 30:772-780 (2013), NAR 30:3059-3066 (2002)
https://mafft.cbrc.jp/alignment/software/

-----

Input file? (FASTA format; Folder="D:\mafft-win")
@ new_DB.txt
OK. infile = new_DB.txt

Output file?
@ DB_msa.txt
OK. outfile = DB_msa.txt

Output format?
1. Clustal format / Sorted
2. Clustal format / Input order
3. Fasta format / Sorted
4. Fasta format / Input order
5. Phylip format / Sorted
6. Phylip format / Input order
@ 4
OK. arguments = --inputorder
```

Abbildung 24: MAFFT als Kommandozeile  
(vgl. [OSAKA23])

Zunächst fragt MAFFT nach der Strategie zur Anwendung des Algorithmus. Hier soll MAFFT automatisch die beste Strategie anhand der Größe und Heterogenität der Daten auswählen. Weiterhin wird als zusätzliches Argument „--anysymbol“ festgelegt, damit MAFFT jedes Zeichen akzeptiert. Ohne das

Argument kann MAFFT die Datei nicht verarbeiten, weil eine Gensequenz in der Regel nur eine bestimmte Menge von Zeichen enthalten darf. Nach dem Drücken von Y und Eingabe wird die Verarbeitung gestartet (siehe Abbildung 25) (vgl. [OSAKA23]).

```
Strategy?
1. --auto
2. FFT-NS-1 (fast)
3. FFT-NS-2 (default)
4. G-INS-i (accurate)
5. L-INS-i (accurate)
6. E-INS-i (accurate)
@ 1
OK. arguments = --auto --inputorder

Additional arguments? (--ep # --op # --kappa # etc)
@ --anysymbol
OK. arguments = --anysymbol --auto --inputorder

command=
"/usr/bin/mafft" --anysymbol --auto --inputorder "new_DB.txt" > "DB_msa.txt"
Type Y or just enter to run this command.
@ Y
```

Abbildung 25: MAFFT als Kommandozeile (Forts.)  
(vgl. [OSAKA23])

Leider wurde die Verarbeitung nach drei Tagen noch nicht beendet. Deswegen wird eine weitere Anwendung benutzt, nämlich ClustalX von University College Dublin (2010). Bevor ClustalX die Datei verarbeitet werden kann, muss sie in ClustalX importiert werden (vgl. [UCD10]). Während des Importierens der Datei hängt ClustalX und nach einer gewissen Zeit endet die Anwendung automatisch. Nach zehn Versuchen kann die Datei in ClustalX noch nicht importiert werden.

Nachdem ClustalX kein Ergebnis geliefert hat, wurde die Open-Source-Anwendung MAGUS von Smirnov et al. (2022) benutzt. MAGUS zerlegt die Datensätze in mehreren Teilen und wendet MSA durch eine dritte auswählbare Anwendung auf jedes Teil an, anschließend fügt MAGUS die Teildatensätze zusammen. [Übers. durch den Verf.] (vgl. [SMIR22])

MAGUS kann als eine Open-Source-Anwendung vom GitHub-Repository von Smirnov et al. (2022) heruntergeladen werden. MAGUS wird durch den folgenden Befehl in einem Computer mit Linux-Betriebssystem gestartet:

```
python3 Installationsort/magus.py -i unaligned_sequences.txt -t clustal --
maxnumsubsets 100 --maxsubsetsize 50 -o magus_result.txt
```

Wobei die Parameter -i die Eingabedatei, -t die Strategie und -o die Ausgabedatei angeben. [Übers. durch den Verf.] (vgl. [SMIR22])

MAGUS hat nach drei Tagen wie die anderen Anwendungen in diesem Abschnitt kein Ergebnis geliefert. Deswegen wird dieser Schritt wegen zeitlichen Gründen übersprungen. Im nächsten Abschnitt werden die längsten gemeinsamen Teilsequenzen aus den Daten berechnet.

## 5.8. Berechnung der längsten gemeinsamen Teilsequenzen

### 5.8.1. Algorithmen zur Berechnung von LCS

Die Studie von Hirschberg (1975, S.341) stellt einen Algorithmus zur Berechnung von LCS dar (siehe Abbildung 26). Die Variablen A und B gibt jeweils die Sequenzen an, für die das LCS berechnet wird. m und n ist jeweils die Länge der Sequenz A und B. L gibt die zweidimensionale Matrix an, den das Ergebnis speichert. Die Zeile speichert die Zeichen der ersten Sequenz und die Spalten die Zeichen der zweiten Sequenz. Im ersten Schritt werden alle Elemente der Matrix mit 0 initialisiert. Anschließend wird über jedes Zeichen der beiden Sequenzen in zwei For-Schleifen iteriert. Wenn ein Zeichen in beiden Sequenzen existiert, ist das aktuelle Element gleich 1 plus das Element, welches diagonal zu dem aktuellen Element steht. Ansonsten wird das aktuelle Element gleich die größte Nummer von zwei Nummern gesetzt. Nämlich sind sie die Elemente, welche eine Einheit oben und eine Einheit links zu dem aktuellen Element stehen [Übers. durch den Verf.] (vgl. [HIRSCH75, S.341]).

**ALG A** (*m, n, A, B, L*)

1. Initialization:  $L(i, 0) \leftarrow 0$  [ $i=0 \dots m$ ];  
 $L(0, j) \leftarrow 0$  [ $j=0 \dots n$ ];
2. for  $i \leftarrow 1$  to  $m$  do  
begin
3. for  $j \leftarrow 1$  to  $n$  do  
if  $A(i) = B(j)$  then  $L(i, j) \leftarrow L(i-1, j-1) + 1$   
else  $L(i, j) \leftarrow \max\{L(i, j-1), L(i-1, j)\}$
- end

Abbildung 26: LCS-Algorithmus  
[HIRSCH75, S. 341]

Ein anschauliches Beispiel stellt die Abbildung 27 dar. Es wird LCS von den Sequenzen „ABC“ und „BC“ berechnet. Alle Elemente bei Index -1 werden gleich 0 gesetzt, z. B.  $L[-1,0]=0$ ,  $L[1,-1]=0$ . In diesem Beispiel ist  $L[0,1]=1$ , weil B in beiden Sequenzen vorkommt,  $L[0,1] = L[-1,0] + 1 = 0 + 1 = 1$ . Die gleiche Bedingung trifft  $L[1,2]$  zu, wo C in beiden Sequenzen vorkommt.  $L[1,1] = 1$ , weil 1 die größte Nummer ist von  $L[0,1]=1$  und  $L[1,0]=0$ .

Index		0	1	2
		A	B	C
0	B	0	1	1
1	C	0	1	2

Abbildung 27: Beispiel für Berechnung von LCS  
(vgl. [CORMEN17, S. 398])

Cormen, Thomas H. et al. (2017) stellt ein Algorithmus, um alle mögliche LCS zu bestimmen. Dazu wird ein Zeiger in jedem Element gespeichert, welches auf dem führenden Weg zu dem Element zeigt. Durch das Zurückgehen des Wegs von dem rechten untersten Element kann LCS erstellt werden (vgl. [CORMEN17, S. 398]). In dem obigen Beispiel wird es von  $L[1,2]$  in  $L[0,1]$  zurückgegangen, daraus entsteht die Sequenz „CB“. Aus der Spiegelung der Sequenz entsteht das Endergebnis „BC“.

### 5.8.2. Umsetzung der Algorithmen in C#

LCS aus allem Sequenz-Paar in der Datenbank wird durch die Methode `calcLCS()` und `lcs()` berechnet. (siehe Abbildung 28 und 29 auf nächster Seite)

Die Variable `rowNumb` speichert durch die Ausführung des SQL-Befehls die Zeilenanzahl der Datenbanktabelle (vgl. [MICRO23 -w]). Die ID der Zeilen startet mit 0 und wird um 1 addiert, deshalb kann in der For-Schleife mit dem Zähler `i` von 0 bis `rowNumb` minus 1 die entsprechende ID durch den Zähler abgefragt werden. Für jede Sequenz wird LCS zwischen ihm und jedem anderen Sequenz berechnet.

```

private static void calcLCS()
{
    try
    {
        using (SqlConnection connection = new SqlConnection("Data Source=DESKTOP-SUKQ2HQ\\SQLEXPRESS;Initial Catalog=Malware API;" +
            "User ID=sa;Password=1234"))
        {
            connection.Open();
            string sqlQuery = "SELECT COUNT(*) FROM malware_api";
            int rowNum = 0;
            using (SqlCommand command = new SqlCommand(sqlQuery, connection))
            {
                // Execute the query and retrieve the result
                rowNum = (int)command.ExecuteScalar();
            }

            for (int i = 0; i < rowNum; i++)
            {
                if (i != rowNum - 1)
                {
                    String currEle = GetSeqById(connection, i);
                    String nextEle = GetSeqById(connection, i + 1);
                    LCS.lcs(currEle, nextEle, currEle.Length, nextEle.Length, connection);
                    for (int k = i + 2; k < rowNum; k++)
                    {
                        String afterNextEle = GetSeqById(connection, k);
                        LCS.lcs(currEle, afterNextEle, currEle.Length, afterNextEle.Length, connection);
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

Abbildung 28: Methode calcLCS()  
(vgl. [MICRO23 -v]; vgl. [MICRO23 -w])

```

private static String GetSeqById(SqlConnection sqlConn, int id)
{
    String seq = "";
    using (SqlCommand sqlComm = new SqlCommand("SELECT sequence FROM malware_api WHERE id=@id", sqlConn))
    {
        sqlComm.Parameters.AddWithValue("@id", id);
        seq = (String)sqlComm.ExecuteScalar();
    }
    return seq;
}

```

Abbildung 29: Methode GetSeqById()  
(vgl. [MICRO23 -x]; vgl. [MICRO23 -w])

Der Algorithmus zum Berechnen von LCS zweier Sequenzen wird in der Methode lcs() entwickelt (siehe Abbildung 30 auf nächster Seite). Eine Datenbanktabelle als Simulation einer Matrix wird für die Speicherung des Ergebnisses verwendet. Die Tabelle hat drei Spalten, die jeweils die Zeilennummer, Spaltennummer und Wert des Elements speichern. Die Methoden InsertToTableTmpArr() und SelectFromTableTempArr() dienen zum Einfügen und Auslesen der Daten. (siehe Abbildung 32 auf S. 42)

In der Methode Backtrack wird durch die Elemente zurückgegangen, um LCS zu erhalten (siehe Abbildung 31 auf S. 41). Alle LCS werden in einem HashSet temporär gespeichert, weil in einem HashSet kein Duplikat existieren darf (vgl.

[MICRO23 -u]). Nachdem die LCS berechnet wurden, werden sie in eine Datenbanktabelle hinzugefügt. Danach werden alle Inhalte der HashSet sowie die temporäre Tabelle gelöscht, um die späteren Berechnungen sie wiederverwenden zu können.

```

static private SqlConnection sqlConn;
static HashSet<string> lcsSet = new HashSet<string>();
static String localTempTable;
static String localFinalTable;
5 references | 3/3 passing
public static void lcs(String X, String Y, int m, int n, SqlConnection sqlConn, String tempTable,
String finalTable)
{
    sqlConn = sqlConn;
    localTempTable = tempTable;
    localFinalTable = finalTable;
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            if (i == 0 || j == 0)
                InsertToTableTmpArr(i, j, 0);

            else if (X[i - 1] == Y[j - 1])
            {
                int val = SelectFromTableTmpArr(i - 1, j - 1) + 1;
                InsertToTableTmpArr(i, j, val);
            }
            else
            {
                int val1 = SelectFromTableTmpArr(i - 1, j);
                int val2 = SelectFromTableTmpArr(i, j - 1);
                int max = Math.Max(val1, val2);
                InsertToTableTmpArr(i, j, max);
            }
        }
    }

    Backtrack(X, Y, m, n, "");
    if (lcsSet.Count > 0)
    {
        foreach (string s in lcsSet)
        {
            Console.WriteLine(s);
            InsertIntoLcsTbl(s);
        }
    }
    lcsSet.Clear();
    DeleteRowsOfTable();
}

```

Abbildung 30: Methode lcs()

(vgl. [GEEKS23]; siehe Abbildung 1.3 und 1.4 in Anlage 1; vgl. [MICRO23 -y])

```

private static void Backtrack(String X, String Y, int i, int j, string currentLCS)
{
    if (i == 0 || j == 0)
    {
        lcsSet.Add(string.Concat(currentLCS.Reverse()));
        return;
    }

    if (X[i - 1] == Y[j - 1])
    {
        currentLCS += X[i - 1];
        Backtrack(X, Y, i - 1, j - 1, currentLCS);
    }
    else
    {
        if (SelectFromTableTmpArr(i - 1, j) >= SelectFromTableTmpArr(i, j - 1))
            Backtrack(X, Y, i - 1, j, currentLCS);
        if (SelectFromTableTmpArr(i, j - 1) >= SelectFromTableTmpArr(i - 1, j))
            Backtrack(X, Y, i, j - 1, currentLCS);
    }
    return;
}

```

Abbildung 31: Methode Backtrack()  
(siehe Abbildung 1.3 und 1.4 in Anlage 1; vgl. [MICRO23 -y])

```

private static void InsertIntoLcsTbl(string s)
{
    using (SqlCommand sqlComm = new SqlCommand("INSERT into "+ localFinalTable+" VALUES (@sequence)", sqlConn))
    {
        sqlComm.Parameters.AddWithValue("@sequence", s);
        sqlComm.ExecuteNonQuery();
    }
}

1 references
private static void DeleteRowsOfTable()
{
    using (SqlCommand sqlComm = new SqlCommand("DELETE FROM "+localTempTable, sqlConn))
    {
        sqlComm.ExecuteNonQuery();
    }
}

3 references
private static void InsertToTableTmpArr(int row, int col, int val)
{
    using (SqlCommand sqlComm = new SqlCommand("INSERT into "+localTempTable+" VALUES (@row,@col,@val)", sqlConn))
    {
        sqlComm.Parameters.AddWithValue("@row", row);
        sqlComm.Parameters.AddWithValue("@col", col);
        sqlComm.Parameters.AddWithValue("@val", val);
        sqlComm.ExecuteNonQuery();
    }
}

7 references
private static int SelectFromTableTmpArr(int row, int col)
{
    int val = 10;
    using (SqlCommand sqlComm = new SqlCommand("SELECT val FROM "+localTempTable+" WHERE row=@row AND col=@col", sqlConn))
    {
        sqlComm.Parameters.Add("@row", SqlDbType.Int, row);
        sqlComm.Parameters["@row"].Value = row;
        sqlComm.Parameters.Add("@col", SqlDbType.Int, col);
        sqlComm.Parameters["@col"].Value = col;
        val = (int)sqlComm.ExecuteScalar();
    }
    return val;
}

```

Abbildung 32: Methoden zum Schreiben und Lesen von der Datenbank (vgl. [MICRO23 -w]; vgl. [MICRO23 -x])

## 5.9. Analyse von Schadprogrammen

Die Analyse von Schadprogrammen kann durch die Methode analyseMalware() ausgeführt werden. Jede aufgenommene Ereignisse werden mit jedem LCS verglichen. Eine Tabellenspalte vom Typ CHAR speichert das Ergebnis der Analyse. Bei einer Übereinstimmung mit einer der LCS wird das Ergebnis gleich 1 gesetzt, ansonsten 0.

```

public static void analyseMalware(String colNameApiTab, String apiTabName, String seqColOfLcsTab, String lcsTab
, String primKeyOfApiTab, SqlConnection connection, String resultCol)
{
    using (SqlCommand selectFromApiTabCmd = new SqlCommand("select * from " + apiTabName, connection))
    {
        using (SqlDataReader reader = selectFromApiTabCmd.ExecuteReader())
        {
            while (reader.Read())
            {
                String id = reader[primKeyOfApiTab].ToString();
                String sequence = reader[colNameApiTab].ToString();
                using (SqlCommand selectFromLcsTabCmd = new SqlCommand("select " + seqColOfLcsTab + " from " + lcsTab +
" where sequence =@sequence", connection))
                {
                    SqlCommand updateApiTabCmd;
                    String val = "";
                    String query = "UPDATE " + apiTabName + " SET "+resultCol+"=''" + val + "' where id =@id";

                    selectFromLcsTabCmd.Parameters.AddWithValue("@sequence", sequence);
                    if (selectFromLcsTabCmd.ExecuteNonQuery() > 0)
                    {
                        val = "1";
                        using (updateApiTabCmd = new SqlCommand(query, connection))
                        {
                            updateApiTabCmd.Parameters.AddWithValue("@id", id);
                            updateApiTabCmd.ExecuteNonQuery();
                        }
                    }
                    else
                    {
                        val = "0";
                        using (updateApiTabCmd = new SqlCommand(query, connection))
                        {
                            updateApiTabCmd.Parameters.AddWithValue("@id", id);
                            updateApiTabCmd.ExecuteNonQuery();
                        }
                    }
                    val = "";
                }
            }
        }
    }
}

```

Abbildung 33: Methode AnalyseMalware()  
(vgl. [MICRO23 -w]; vgl. [KAM18]; vgl. [MICRO23 -x])

## 5.10. Unit-Test des Programmcodes

Unit-Test zur Überprüfung der Funktionalität der entwickelten Methoden wurden erfolgreich durchgeführt. Die Testmethoden wurden anhand der Anleitung von Jones (2023) entwickelt. Alle Unit-Tests sind im Quellcode enthaltend.

## 5.11. Ergebnis der Umsetzung

Durch die Einrichtung der VM1 entsteht eine VM, die zur allgemeinen Schadprogramm-Analyse benutzt werden kann. Weitere VM können durch die Verbindung mit dem internen Netzwerk miteinander verbunden werden. Eine kompilierbare C#-Anwendung wurde entwickelt, die zur Datenübertragung mit TCP-Protokoll zwischen den VM dient. Darüber hinaus kann die C#-Anwendung

durch die Nutzung von Bibliothek EventTracing die Ereignisse eines Schadprogramms dokumentieren und in eine Datenbank einfügen. Daneben kann die Anwendung die LCS von Ereignissen in der Datenbank berechnen. Weiterhin verfügt die Anwendung über Methoden, welche die Ereignisse einer beliebigen Anwendung mit den vorhandenen Datenbeständen vergleichen können. Die Anwendung von MSA auf die Datenbeständen hat innerhalb der Arbeitszeit kein Ergebnis geliefert.

## 6. Bewertung des Ergebnisses

Die Anwendung kann die Ereignisse im Kernelmodus von sowohl einem Schadprogramm als auch einer beliebigen Anwendung sammeln, welche in Windows 10 64-Bit-Version ausführbar ist. Daneben erfolgt die Berechnung von LCS durch die Implementierung des Algorithmus und Verwendung der Datenbank.

Die Umsetzung der automatischen Duplikate Entfernung und Kodierung in einer Log-Datei war erfolgreich.

Die Benutzung von verschiedenen MSA-Anwendungen hat innerhalb der Arbeitszeit kein Ergebnis geliefert. Es liegt an mangelnder Forschung zu Grenzen der einzelnen Anwendung.

## 7. Zusammenfassung

Die Aufnahme von Ereignissen im Kernelmodus von den in Windows 64-Bit-Version ausführbaren Schadprogrammen wurde durch die C#-Anwendung erreicht. Der Algorithmus zur Berechnung von LCS zwischen zwei Sequenzen wurde erfolgreich umgesetzt. Die entwickelte Anwendung kann die Ereignisse im Kernelmodus einer Anwendung mit den Datenbeständen vergleichen. Das Ziel der Anwendung von MSA auf die Datenbestände war nicht erfolgreich.

In den zukünftigen Arbeiten könnten bereits erwähnte bzw. weitere Anwendungen geforscht werden, ob sie MSA auf einem Datenbestand ab einer Größe von hunderten Megabytes anwenden könnten. Die Leistung des Erkennens von Schadprogrammen kann in zukünftigen Arbeiten geforscht werden. Darüber hinaus können durch den Einsatz von Data Mining und Künstliche Intelligenz die Ereignisse im Kernelmodus analysiert werden, um Informationen über das Verhalten und Eigenschaften eines Schadprogramms zu erhalten.

## Literaturverzeichnis

[KI15] Youngjoon Ki, Eunjin Kim und Huy Kang Kim: *A Novel Approach to Detect Malware Based on API Call Sequence Analysis*, In: International Journal of Distributed Sensor Networks, 2015. Online im Internet: <https://journals.sagepub.com/doi/10.1155/2015/659101>.

[BAUN22] Baun, Christian: *Betriebssysteme kompakt. Grundlagen, Hardware, Speicher, Daten und Dateien, Prozesse und Kommunikation, Virtualisierung*. IT kompakt. 3. Auflage. Springer Vieweg, Berlin, Heidelberg: 2022.

[BRID21] Bridge, Karl et al.: *About Event Tracing*. Online im Internet: <https://learn.microsoft.com/en-us/windows/win32/etw/about-event-tracing>, 01.07.2021. Stand 30.07.2023.

[CARDA88] Carrillo, Humberto und David, Lipman: *The multiple sequence alignment problem in Biology*, In: Siam Journal on Applied Mathematics, Society for Industrial and Applied Mathematics, 1988, Bd. 48, Nr. 5. Online im Internet: <https://epubs.siam.org/doi/10.1137/0148063>, S. 1073–1082.

[OKO12] Okonechnikov Konstantin, Olga Golosova, Mikhail Fursov und the UGENE team: *UNIPro UGENE: a Unified Bioinformatics toolkit*, In: Bioinformatics, Oxford University Press, 2012, Bd. 28, Nr. 8. Online im Internet: <https://academic.oup.com/bioinformatics/article/28/8/1166/195474>, S. 1166–1167.

[MAIER78] Maier, David: *The complexity of some problems on subsequences and supersequences*, In: Journal of the ACM, Association for Computing Machinery, 1978, Bd. 25, Nr. 2. Online im Internet: <https://dl.acm.org/doi/10.1145/322063.322075>, S. 322–336.

[HACKER19] HackerSploit: *How to setup a sandbox environment for malware analysis* [Video]. Online im Internet: <https://www.youtube.com/watch?v=oPxy9JF8FM>, 06.06.2019. Stand 30.07.2023.

[ORAC23] Oracle. (2023). *Oracle VM VirtualBox (7.0.10)* [Software]. Online im Internet: <https://www.virtualbox.org/wiki/VirtualBox>.

[VM23] VMware. (2023). *VMware Workstation 17 Player (17.0.2)* [Software]. Online im Internet: <https://www.vmware.com/de.html>.

[KURDI23] Kurdiumov, Andrii: *The Microsoft.Diagnostics.Tracing.TraceEvent Library*. Online im Internet: <https://github.com/microsoft/perfview/blob/main/documentation/TraceEvent/TraceEventLibrary.md>, 09.09.2023. Stand 02.10.2023.

[MICRO22] Microsoft. (2022). *Visual Studio 2022 (Community 2022 Version 17.6.3)* [Software]. Online im Internet: <https://visualstudio.microsoft.com/de/>.

[ROB18] Robbins, Brian: *TraceEventSession.cs at main*. Online im Internet: <https://github.com/microsoft/perfview/blob/main/src/TraceEvent/TraceEventSession.cs>, 02.10.2018. Stand 30.07.2023.

[RAY23] Ray, Mike et al.: *char und varchar (Transact-SQL)*, Online im Internet: <https://learn.microsoft.com/de-de/sql/t-sql/data-types/char-and-varchar-transact-sql?view=sql-server-ver16>, 23.05.2023, Stand 04.10.2023.

[MICRO22 -b] Microsoft. (2022). *Microsoft SQL Server 2022 Express (16.0.1000.6)* [Software]. Online im Internet: <https://www.microsoft.com/de-de/sql-server/sql-server-2022>.

[ORAC] Oracle: *Virtual Networking*. Online im Internet: [https://www.virtualbox.org/manual/ch06.html#network\\_internal](https://www.virtualbox.org/manual/ch06.html#network_internal). Stand 30.07.2023.

[CYBER21] Cyber Studies: *How to build a SECURE hacking lab (VirtualBox networking)* [Video]. Online im Internet: [https://www.youtube.com/watch?v=lm\\_4hoe-K7U](https://www.youtube.com/watch?v=lm_4hoe-K7U), 07.10.2021. Stand 30.07.2023.

[MICRO23] Microsoft: *Windows 10 herunterladen*. Online im Internet: <https://www.microsoft.com/de-de/software-download/windows10>. Stand 30.07.2023.

[ABUSE] abuse.ch: *Index of /malware-bazaar/daily*. Online im Internet: <https://datalake.abuse.ch/malware-bazaar/daily/>. Stand 30.07.2023.

[DANDE21] Dandekar, Thomas und Kunz, Meik: *Bioinformatik. Ein einführendes Lehrbuch*. 2. Auflage, Springer Spektrum, Berlin: 2021.

[NCBI] National Center for Biotechnology Information: *Query Input and database selection*. Online im Internet: <https://blast.ncbi.nlm.nih.gov/doc/blast-topics/>. Stand 30.07.2023.

[MICRO23 -b] Microsoft: *IPEndPoint klasse (System.Net)*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.net.ipendpoint?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -c] Microsoft: *Socket klasse (System.Net.Sockets)*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.net.sockets.socket?view=net-7.0>. Stand 30.07.2023.

[MORRI18] Morrison, Vance: *31\_KernelAndClrMonitor.cs*. Online im Internet: [https://github.com/microsoft/perfview/blob/main/src/TraceEvent/Samples/31\\_KernelAndClrMonitor.cs](https://github.com/microsoft/perfview/blob/main/src/TraceEvent/Samples/31_KernelAndClrMonitor.cs), 27.01.2018. Stand 30.07.2023.

[MICRO23 -d] Microsoft: *Directory.EnumerateFiles Methode*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.io.directory.enumeratefiles?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -e] Microsoft: *Process.StartInfo Eigenschaft*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.diagnostics.process.startinfo?view=net-7.0> . Stand 30.07.2023.

[ANDKA23] andkamashev: *KernelTraceEventParsers.cs*. Online im Internet: <https://github.com/microsoft/perfview/blob/main/src/TraceEvent/Parsers/KernelTraceEventParser.cs>, 09.02.2023. Stand 30.07.2023.

[MICRO23 -f] Microsoft: *Console.CancelKeyPress Ereignis*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.console.cancelkeypress?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -g] Microsoft: *Timer Klasse*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.threading.timer?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -h] Microsoft: *Process.Kill Methode*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.diagnostics.process.kill?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -i] Microsoft: *Socket.Shutdown(SocketShutdown) Methode*. Online im Internet:  
<https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.shutdown?view=net-7.0>. Stand 30.07.2023.

[LAND14] Landwerth, Immo: *The TraceEvent Library Programmers Guide*.  
Online im Internet:  
<https://github.com/microsoft/dotnet-samples/blob/master/Microsoft.Diagnostics.Tracing/TraceEvent/docs/TraceEvent.md>, 19.08.2014. Stand 30.07.2023.

[MICRO23 -j] Microsoft: *Socket.SendAsync Methode*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.net.sockets.socket.sendasync?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -k] Microsoft: *Environment.Exit(Int32) Methode*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.environment.exit?view=net-7.0>. Stand 30.07.2023.

[PINE22] Pine David, Schonning Nick und Aksoy Ahmet Ibrahim: *TCP overview*.  
Online im Internet:  
<https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/sockets/tcp-classes>, 06.12.2022. Stand 30.07.2023.

[MICRO23 -l] Microsoft: *System.Net.Sockets Namespace*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.net.sockets?view=net-7.0>.  
Stand 30.07.2023.

[MICRO23 -m] Microsoft: *FileStream Konstruktoren*. Online im Internet:  
[https://learn.microsoft.com/de-de/dotnet/api/system.io.filestream.-ctor?view=net-7.0#system-io-filestream-ctor\(system-string-system-io-filemode-system-io-fileaccess\)](https://learn.microsoft.com/de-de/dotnet/api/system.io.filestream.-ctor?view=net-7.0#system-io-filestream-ctor(system-string-system-io-filemode-system-io-fileaccess)). Stand 30.07.2023.

[MICRO23 -n] Microsoft: *Dictionary<TKey, TValue> Klasse*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.collections.generic.dictionary-2?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -o] Microsoft: *String.Split Methode*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.string.split?view=net-7.0>.  
Stand 30.07.2023.

[MICRO23 -p] Microsoft: *StreamReader Klasse*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.io.streamreader?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -q] Microsoft: *String.IsNullOrEmpty(String) Methode*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.string.isnullorwhitespace?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -r] Microsoft: *String.StartsWith Methode*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.string.startswith?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -s] Microsoft: *String.Substring Methode*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.string.substring?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -t] Microsoft: *StreamWriter Klasse*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.io.streamwriter?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -u] Microsoft: *String.Equals Methode*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.string.equals?view=net-7.0>. Stand 30.07.2023.

[MICRO23 -v] Microsoft: *SqlConnection.ConnectionString Eigenschaft*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.data.sqlclient.sqlconnection.connectionstring?view=dotnet-plat-ext-7.0#system-data-sqlclient-sqlconnection-connectionstring>. Stand 30.07.2023.

[MICRO23 -w] Microsoft: *SqlCommand Klasse*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.data.sqlclient.sqlcommand?view=dotnet-plat-ext-7.0>. Stand 30.07.2023

[MICRO23 -x] Microsoft: *SqlCommand.Parameters Eigenschaft*. Online im Internet:  
<https://learn.microsoft.com/de-de/dotnet/api/system.data.sqlclient.sqlcommand.parameters?view=dotnet-plat-ext-7.0#system-data-sqlclient-sqlcommand-parameters>. Stand 30.07.2023

[HAI23] Haiying, Yu und Padma, Jayaraman: *Remove duplicate rows from a table in SQL server*. Online im Internet: <https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/development/remove-duplicate-rows-sql-server-tab>, 20.06.2023. Stand 30.07.2023.

[KAM18] Kamrushi, Suresh: *C# Getting column name from dataset*. Online im Internet: <https://suresh-kamrushi.medium.com/c-getting-column-name-from-dataset-ce3c81a21505>, 02.11.2018. Stand 30.07.2023.

[OSAKA23] Osaka University und RIMD. (2023). *MAFFT (7.520)* [Software]. Online im Internet: <https://mafft.cbrc.jp/alignment/software/windows.html>.

[UCD10] University College Dublin. (2010). *Clustal X (2.1)* [Software]. Online im Internet: <http://www.clustal.org/clustal2/>

[SMIR22] Smirnov, Vladimir et al.: *MAGUS*. Online im Internet: <https://github.com/vlasmirnov/MAGUS>, 11.04.2022. Stand 30.07.2023.

[HIRSCH75] Hirschberg, Daniel S.: *A linear space algorithm for computing maximal common subsequences*, In: Communications of The ACM, Association for Computing Machinery, Bd. 18, Nr. 6, 1975, Online im Internet: <https://dl.acm.org/doi/10.1145/360825.360861>, S. 341–343.

[CORMEN17] Cormen, Thomas H. et al.: *Algorithmen – Eine Einführung*. 4. Auflage. Berlin, Boston: 2017.

[MICRO23 -y] Microsoft: *HashSet<T> Klasse*. Online im Internet: <https://learn.microsoft.com/de-de/dotnet/api/system.collections.generic.hashset-1?view=net-7.0>. Stand 30.07.2023.

[GEEKS23] GeeksforGeeks et al.: *Printing longest common subsequence*. Online im Internet: <https://www.geeksforgeeks.org/printing-longest-common-subsequence/>, 17.02.2023. Stand 30.07.2023.

[OPENAI23] OpenAI. (2023). *OpenAI GPT-3.5 Model* [Software]. Online im Internet: <https://openai.com>.

[JONES23] Jones, Mike et al.: *Walkthrough: Create and run unit tests for managed code*. Online im Internet: <https://learn.microsoft.com/en-us/visualstudio/test/walkthrough-creating-and->

[running-unit-tests-for-managed-code?view=vs-2022#create-a-unit-test-project](#),  
03.10.2023. Stand 03.10.2023.

# Anlagen

## 1. Persönliche Kommunikation mit ChatGPT

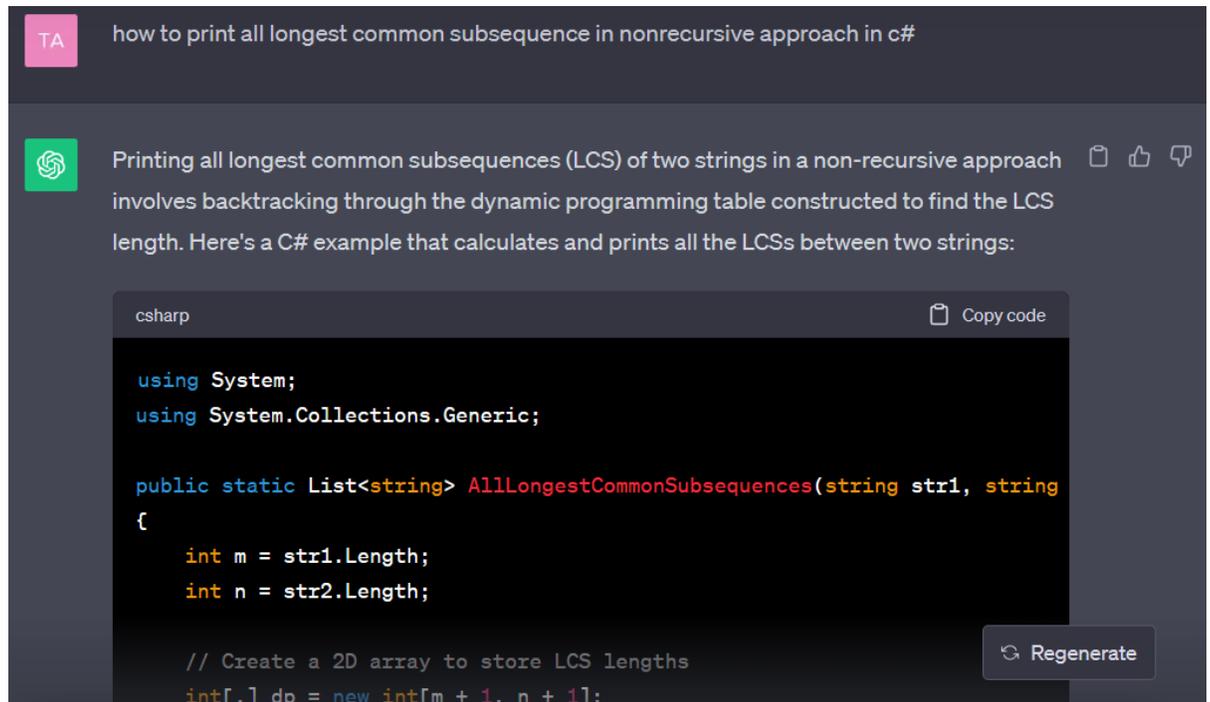


Abbildung 1.1: Frage an ChatGPT

[OPENAI23]

```
// Fill the dp array using dynamic programming
for (int i = 0; i <= m; i++)
{
    for (int j = 0; j <= n; j++)
    {
        if (i == 0 || j == 0)
            dp[i, j] = 0; // Base case: LCS of an empty string is 0
        else if (str1[i - 1] == str2[j - 1])
            dp[i, j] = dp[i - 1, j - 1] + 1; // Match, increment LCS len
        else
            dp[i, j] = Math.Max(dp[i - 1, j], dp[i, j - 1]); // No match
    }
}

int lcsLength = dp[m, n];

// Use a set to store unique LCSs
HashSet<string> lcsSet = new HashSet<string>();
```

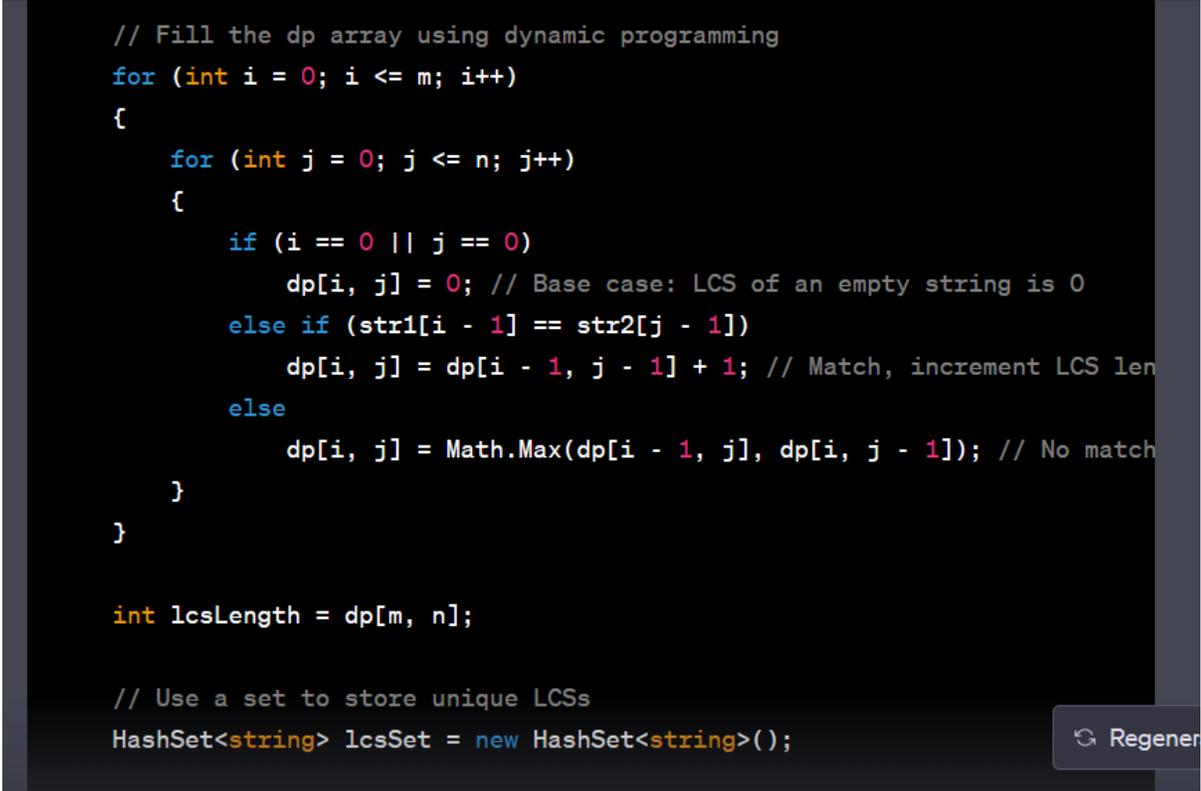


Abbildung 1.2: Antwort von ChatGPT

[OPENAI23]

```
// Helper function to backtrack and build LCSs
void Backtrack(int i, int j, string currentLCS)
{
    if (i == 0 || j == 0)
    {
        lcsSet.Add(string.Concat(currentLCS.Reverse()));
        return;
    }

    if (str1[i - 1] == str2[j - 1])
    {
        currentLCS += str1[i - 1];
        Backtrack(i - 1, j - 1, currentLCS);
    }
    else
    {
        if (dp[i - 1, j] >= dp[i, j - 1])
            Backtrack(i - 1, j, currentLCS);
        if (dp[i, j - 1] >= dp[i - 1, j])
            Backtrack(i, j - 1, currentLCS);
    }
}
```

↻ Regene

Abbildung 1.3: Antwort von ChatGPT (Fortsetzung)

[OPENAI23]

```

    }
}

Backtrack(m, n, "");

return new List<string>(lcsSet);
}

public static void Main()
{
    string str1 = "ABCBADAB";
    string str2 = "BDCAB";

    List<string> lcsList = AllLongestCommonSubsequences(str1, str2);

    Console.WriteLine("Longest Common Subsequences:");
    foreach (string lcs in lcsList)
    {
        Console.WriteLine(lcs);
    }
}

```

Regen

Abbildung 1.4: Antwort ChatGPT (Fortsetzung)

[OPENAI23]

In this code:

- We use a `HashSet` (`lcsSet`) to store unique LCSs, as there may be duplicates.
- The `Backtrack` function is a recursive helper function that builds LCSs by backtracking through the dynamic programming table. It uses string concatenation to construct each LCS.
- Finally, in the `Main` method, we call `AllLongestCommonSubsequences` to find all LCSs between two example strings and print them.

This code will correctly find and print all the longest common subsequences between the given strings.

Abbildung 1.5: Antwort ChatGPT (Fortsetzung)

[OPENAI23]