

Bachelor Arbeit  
**Automatisierung von Dynamischer Malware  
Instrumentierung**

im Studiengang Softwaretechnik der Fakultät Informationstechnik  
Sommersemester 2023

Muhammed Adel

**Zeitraum:** 01.12.22 - 31.5.23  
**Prüfer:** PROF. DR.-ING. Reinhard Schmidt  
**Zweitprüfer:** Ediz Turcan

---

**Firma:** adesso SE  
**Betreuer:** Hassan Sbeyti

# Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 31.05.2023 \_\_\_\_\_  
Unterschrift

# Abstract

The rapid evolution of malware and the persistent development of new attack vectors present a continuous challenge for cybersecurity professionals. Traditional signature-based malware detection techniques often fall short when confronted with unknown or changing malware, making it imperative to develop more efficient and automated approaches for malware detection. This thesis aims to address this challenge by automating dynamic malware detection and providing a tool for local malware analysis.

Our primary objective is to develop a fully functional test environment supporting automated dynamic malware analysis. By the end of this project, it should be possible for any user, with minimal requirements, to set up this test environment locally and conduct automated testing and analysis of malware. The central research question driving this study is, "How can the process of malware analysis be automated to the extent that researchers can perform local tests on their own systems to better prepare themselves for dealing with malware?"

Adopting a Scrum-based methodology, we first construct a secure test environment for malware analysis, using Oracle VirtualBox and automation scripts for independent setup. We then prepare virtual machines for malware analysis, taking numerous safety precautions to prevent potential damage from running malware. The significant part of the project involves data collection and preparation. We search for appropriate Goodware, which provides high-quality training data for the machine learning model, and use publicly available databases for malware samples.

The successful completion of this project results in an automated test environment for dynamic malware analysis. The system can now be built locally and is capable of testing 30 malware per hour. It enables efficient and rapid detection and analysis of malware, contributing to improved IT security.

# Inhaltsverzeichnis

1	Einleitung	1
1.1	Hintergrund und Motivation . . . . .	1
1.2	Problemstellung und Forschungsfrage . . . . .	2
1.3	Zielsetzung und Methodik . . . . .	3
1.4	Struktur der Arbeit . . . . .	4
2	Malware: Grundlagen und Typen	5
2.1	Einführung in die Malware . . . . .	5
2.2	Evolution der Malware über die Jahre . . . . .	5
2.2.1	Erste Phase der Malware-Entwicklung 1949 - 1991 . . . . .	5
2.2.2	Zweite Phase der Malware-Entwicklung 1992 - 1999 . . . . .	6
2.2.3	Dritte Phase der Malware-Entwicklung 2000 - 2008 . . . . .	7
2.2.4	Vierte Phase der Malware-Entwicklung 2005 - 2016 . . . . .	8
2.2.5	Fünfte Phase der Malware-Entwicklung 2010 - aktuell . . . . .	8
2.3	Malware-Typen . . . . .	9
2.3.1	Virus . . . . .	9
2.3.2	Wurm . . . . .	9
2.3.3	Trojaner . . . . .	10
2.3.4	Ransomware . . . . .	10
2.3.5	Botnetz . . . . .	10
2.3.6	Adware . . . . .	11
2.3.7	Spyware . . . . .	11
2.3.8	Rootkits . . . . .	11
2.3.9	Fileless Malware . . . . .	12
2.3.10	Malvertising . . . . .	12
2.3.11	Weitere Malware-Arten . . . . .	12
2.4	Verhalten und Eigenschaften von Malware . . . . .	12
2.4.1	Statische vs Dynamische Malware . . . . .	13
2.4.2	Malware Verbreitung . . . . .	17
2.4.3	Malware Obfuscation . . . . .	20
2.5	Malware-Analyse-Methoden . . . . .	24
2.5.1	Statische Analyse . . . . .	24
2.5.2	Dynamische Analyse . . . . .	25
2.6	Windows System Calls . . . . .	25
2.7	Automatisierung in der Malware-Erkennung und Analyse . . . . .	26
3	Analyse und Design-Vorbereitung	28
3.1	Problembeschreibung . . . . .	28

3.2	Softwareauswahl und Designplanung . . . . .	29
3.2.1	Anforderungen . . . . .	29
3.2.2	Virtualisierungssoftware . . . . .	30
3.2.3	ISO-Dateien . . . . .	31
3.2.4	Intel Pin Tool . . . . .	32
3.2.5	Automatisierungsprozess . . . . .	33
3.2.6	Isolierungsprozess und Hardening . . . . .	34
3.2.7	FTP-Server . . . . .	35
3.2.8	Aufbereitung von Goodware und Malware . . . . .	35
3.2.9	Herausforderungen mit dem Windows Defender . . . . .	36
3.2.10	Erhebung und Weiterleitung der Daten . . . . .	37
3.2.11	Dokumentation . . . . .	37
4	Implementierung . . . . .	38
4.1	Erstellung der Testumgebung . . . . .	38
4.2	Vorbereitung der Maschinen zur Malware-Analyse . . . . .	39
4.2.1	Host-Maschine . . . . .	39
4.2.2	Ubuntu-Maschine . . . . .	40
4.2.3	Windows-Maschine . . . . .	40
4.3	Automatisierung der Malware-Analyse . . . . .	43
4.3.1	Automatisierte Analyse von Goodware . . . . .	43
4.3.2	Automatisierte Analyse von Malware . . . . .	44
5	Auswertung und Ausblick . . . . .	48
5.1	Bewertung des Systems . . . . .	48
5.2	Ausblick . . . . .	49
6	Fazit . . . . .	50
A	Anhänge . . . . .	51
A.1	complete_setup.bat . . . . .	51
A.2	start_malware_analysis.py . . . . .	53
A.3	pin_tool_execution.py . . . . .	55
A.4	pin_tool_execution_arg.py . . . . .	57
A.5	add_goodware_exe_to_csv.py . . . . .	60
A.6	add_malware_exe_to_csv.py . . . . .	60
A.7	eula.py . . . . .	61
A.8	find_file_type_and_create_exe.py . . . . .	61
A.9	execute_pintool.sh . . . . .	62
A.10	ftp_setup.sh . . . . .	62
	Literaturverzeichnis . . . . .	64

# Abbildungsverzeichnis

2.1	Encrypted Malware [34]	14
2.2	Oligomorphic Malware [34]	14
2.3	Polymorphic Malware [34]	15
2.4	Metamorphic Malware [34]	16
2.5	Metamorphic Malware Struktur [34]	16
2.6	Drive-by-Download	18
2.7	Malware Verbreitung	20
2.8	Sample Code [52]	21
2.9	Dead-Code Insertion [52]	21
2.10	Dead-Code Insertion Sequence [52]	21
2.11	Register Reassignment [52]	22
2.12	Instruction Substitution [52]	23
2.13	Code Transposition based on Unconditional Branches [52]	23
2.14	Code Transposition based on Independent Instructions [52]	23
3.1	Interfaces for interacting with Oracle Virtuellen Maschinen (VM) VirtualBox [48]	30
3.2	VBoxManage unattended detect	31
3.3	Übersicht der Netzwerkmodi [49]	34
4.1	CSV Datei	43
4.2	Flussdiagramm der Prozessabläufe	44
4.3	Ausgabe der Goodware-Analyse im Terminal	44
4.4	Sequenzdiagramm	45
4.5	disable_windows_defender	46
4.6	Ausgabe bei Erfolgreicher Analyse	47
4.7	Ausgabe bei fehlgeschlagener Analyse	47
5.1	Ausgabe auf dem FTP-Server	48

# Abkürzungsverzeichnis

***API*** Application Programming Interface

***GUI*** Grafische Benutzeroberfläche

***CERT*** Computer Emergency Response Team

***APT*** Advanced Persistent Thread

***VM*** Virtuellen Maschinen

***CSV*** Comma-Separated Values

# 1 Einleitung

## 1.1 Hintergrund und Motivation

Die rasante technologische Entwicklung und ihre Integration in unseren Alltag haben zu zahlreichen Veränderungen geführt. Heutzutage spielen Computer und andere intelligente Geräte eine immens wichtige Rolle in unserem Leben. Sie unterstützen uns bei der Sicherung und dem Abruf von Daten, erleichtern unsere Aufgaben und ermöglichen die Automatisierung mühseliger Prozesse. Dabei arbeiten sie präziser und schneller als es ein Mensch jemals könnte.

Mit dem technologischen Fortschritt steigt auch der Bedarf an Datensicherung und -verarbeitung. Während früher eine Diskette mit wenigen Kilobytes Speicherplatz ausreichend war, sind heute selbst 100 GB Cloud-Speicher häufig nicht genügend. Wertvolle und persönliche Daten werden nun online und auf verschiedenen Geräten gespeichert, sodass sie von überall auf der Welt zugänglich sind. Idealerweise sind alle Geräte intelligent miteinander verknüpft und können somit von jedem Gerät aus erreicht werden. Durch den Anstieg der im Internet zirkulierenden Daten wächst jedoch auch das Risiko von Datenschutzverletzungen und Datendiebstahl.

In den letzten Jahren wurde ein stetiger Anstieg von Datenschutzverletzungen und Datendiebstählen beobachtet. Eine Datenschutzverletzung liegt vor, wenn eine unbefugte Person oder Organisation Zugang zu Daten erhält, die sie nicht haben sollte. Um die Ausmaße der Datenschutzverletzungen und Diebstähle in Zahlen zu verdeutlichen, zeigt der 2023 SONICWALL CYBER THREAT REPORT[42], dass es im Jahr 2022 allein 5,5 Milliarden Malware-Angriffe (+2%), 6,3 Billionen (+19%) Einbruchversuche, 139,3 Millionen (+43%) Cryptojacking-Angriffe und 112,3 Millionen (+87%) neu entdeckte IoT-Malware gab. Allein in Deutschland schätzt man den jährlichen gesamten Schaden der durch Cyber-Attacken entsteht auf ca. 203€ Milliarden [8]. Vor diesem Hintergrund gewinnt das Thema Cybersicherheit immer mehr an Bedeutung für Unternehmen und Regierungen, aber auch für Privatpersonen. Cybersicherheit basiert auf drei grundlegenden Prinzipien, die als CIA-Modell [41] bezeichnet werden.

Vertraulichkeit (Confidentiality) bezieht sich darauf, die Privatsphäre von Informationen zu schützen, indem ausschließlich autorisierten Personen Zugang zu den entsprechenden Daten gewährt wird. Um dies zu erreichen, kommen Authentifizierungsverfahren wie Benutzername und Passwort oder andere Sicherheitsmechanismen zum Einsatz. Ein Beispiel, wie Malware die Vertraulichkeit verletzen kann, sind Keylogger, die Tastatureingaben aufzeichnen, um Passwörter und andere sensible Daten abzugreifen. Integrität (Integrity) gewährleistet, dass Daten vor unberechtigten Änderungen geschützt sind und unverfälscht übermittelt werden. Bei der Kommunikation ist dies entscheidend, damit die gesendeten Informationen ohne Manipulationen beim Empfänger ankommen. Techniken wie Verschlüsselung, Entschlüsselung oder Hashing werden angewendet, um die Integrität der Daten sicherzustellen. Malware kann die Integrität beeinträchtigen, indem sie beispielsweise Daten manipuliert oder in einer Kommunikation zwischengeschaltete Nachrichten verändert. Verfügbarkeit (Availability) beschreibt die ständige Bereitschaft



und Zugänglichkeit von Daten für autorisierte Benutzer, wenn diese benötigt werden. In diesem Zusammenhang spielt auch die Konsistenz der bereitgestellten Daten eine wichtige Rolle. Bei gleicher Anfrage sollten stets identische Ergebnisse erzielt werden. Allerdings kann es bei einer Überlastung des Systems vorkommen, dass die Daten unvollständig oder sogar fehlerhaft übertragen werden. Deshalb ist es wichtig, dass Systeme und Informationen ohne unangemessene Verzögerungen oder Beeinträchtigungen zugänglich sein sollten. Ransomware ist ein Beispiel für Malware, die die Verfügbarkeit verletzt, indem sie Dateien und Systeme verschlüsselt und dadurch den Zugriff auf die betroffenen Daten blockiert, bis ein Lösegeld gezahlt wird [41].

Bösartige Software, kurz Malware, ist die Sammelbezeichnung für eine Vielzahl von feindlicher oder aufdringlicher Software. Cyberkriminelle entwickeln Malware, um Daten zu stehlen, Zugangskontrollen zu umgehen und sich Zugang zu einem persönlichen Computer zu verschaffen oder um den Zielcomputer, seine Daten oder Anwendungen zu schädigen. Heutzutage ist die Malware-Industrie sehr lukrativ geworden, was die Bemühungen von Cyber-Kriminellen verstärkt hat und zu einem exponentiellen Anstieg der Anzahl, der Arten und der Komplexität von Malware geführt hat [1].

Traditionelle Methoden zur Malware-Erkennung basieren häufig auf signaturbasierten Ansätzen, bei denen bekannte Malware-Signaturen in einer Datenbank gespeichert und mit verdächtigen Dateien verglichen werden. Diese Methode kann jedoch leicht durch polymorphe und metamorphe Malware umgangen werden, die ihre Signaturen bei jeder Infektion oder sogar während der Ausführung verändert, also hat man eine unendliche Menge an verschiedener Malware, die man versuchen will zu erkennen[1][15].

Dynamische Malware-Analyse ist ein Ansatz, der das Verhalten von Malware während der Ausführung analysiert, um ihre Erkennung zu verbessern. Dieser Ansatz ist besonders nützlich für die Erkennung von unbekannter oder sich ständig verändernder Malware, die traditionelle signaturbasierte Methoden umgehen kann. Eine wichtige Quelle für Informationen über das Verhalten von Malware sind die Windows System Calls, die von schädlichen Programmen verwendet werden, um auf Betriebssystem Ressourcen zuzugreifen[3][13][15][38][46].

Die Automatisierung der dynamischen Malware-Analyse und -Erkennung durch den Einsatz von maschinellem Lernen und künstlicher Intelligenz ist eine vielversprechende Möglichkeit, um die Erkennungsgeschwindigkeit und -genauigkeit zu erhöhen. Eine automatisierte Erkennung kann dazu beitragen, die Abhängigkeit von manuellen Analysen und menschlichen Experten zu reduzieren und schneller auf neue Bedrohungen zu reagieren[19].

In diesem Kontext besteht die Motivation dieser Bachelorarbeit darin, die dynamische Malware-Erkennung zu automatisieren und ein Werkzeug für die Malware-Analyse bereitzustellen. Es soll ein System entwickelt werden, das Nutzern ermöglicht, Malware-Analysen lokal auf ihrem eigenen System durchzuführen.

## 1.2 Problemstellung und Forschungsfrage

Die Erkennung und Abwehr von Malware ist eine kontinuierliche Herausforderung für IT - Sicherheitsexperten, da Cyberkriminelle ständig neue Angriffsvektoren und Techniken entwickeln [42]. Traditionelle signaturbasierte Malware-Erkennungsmethoden stoßen heutzutage schnell auf

ihre Grenzen, wenn es sich um die Erkennung von unbekannter oder sich verändernder Malware geht. Darüber hinaus sind manuelle Analyseverfahren zeitaufwändig und auf den Input von Experten angewiesen, was zu Verzögerungen bei der Reaktion auf neue Bedrohungen führen kann. Daher ist es erforderlich, effizientere und automatisierte Ansätze zur Malware-Erkennung zu entwickeln.

Dynamische Malware-Analyse, die das Verhalten von Malware während der Ausführung untersucht, ist ein vielversprechender Ansatz zur Erkennung von fortgeschrittener und unbekannter Malware. Die Analyse von Application Programming Interface (API)-Aufrufen, Systemaufrufen und Befehlsspuren bietet wertvolle Informationen über das Verhalten von Malware und kann dazu beitragen, automatisierte Erkennungssysteme zu entwickeln [3]. Jedoch bleiben Herausforderungen hinsichtlich der effektiven Identifikation von Mustern und Anomalien in dem Verhalten der Malware zu identifizieren. Zudem gilt es, geeignete maschinelle Lernmodelle zu identifizieren und zu trainieren, die eine hohe Erkennungsgenauigkeit bei gleichzeitig geringen Falsch-Positiv-Raten bieten.

Angesichts dieser Überlegungen formuliert sich die zentrale Forschungsfrage dieser Bachelorarbeit folgendermaßen: "Wie kann der Prozess der Malware-Analyse soweit automatisiert werden, dass Forscher die Möglichkeit haben, lokale Tests auf ihren eigenen Systemen durchzuführen, um sich besser auf den Umgang mit Malware vorzubereiten?" Durch die Beantwortung dieser Frage wird angestrebt, ein Instrument zu entwickeln, das die Analyse von Malware automatisieren kann. Obwohl das System selbst die analysierten Daten nicht auswerten wird, soll es den Forschern ermöglichen, neue Bedrohungen effizienter und schneller zu erkennen und anzugehen.

### 1.3 Zielsetzung und Methodik

Das primäre Ziel dieser Arbeit ist die Entwicklung einer voll funktionsfähigen Testumgebung, die die automatisierte dynamische Malware-Analyse unterstützt. Am Ende dieses Projekts sollte es jedem Anwender möglich sein, mit minimalen Anforderungen, diese Testumgebung lokal einzurichten und Malware automatisch zu testen und zu analysieren.

Die Methodik dieser Arbeit orientiert sich am Scrum-Modell. Mein Betreuer, Hassan Sbeyti, nimmt hierbei die Rolle des Product Owners ein, während ich die Aufgaben der Implementierung und des Systemdesigns übernehme. Wir haben uns in regelmäßigen Sprints, die in Zwei-Wochen-Zyklen stattfanden, getroffen, um die aktuellen Resultate zu evaluieren und das weitere Vorgehen zu diskutieren.

Der erste Schritt bestand darin, eine sichere Testumgebung für die Malware-Analyse zu konstruieren. Diese Testumgebung wurde mittels Oracle VirtualBox erstellt und kann durch Automatisierungsskripte vollständig unabhängig eingerichtet werden.

Als nächstes galt es, die virtuellen Maschinen zur Malware-Analyse vorzubereiten. Angesichts des inhärenten Risikos, das mit der Ausführung von Malware einhergeht, wurden verschiedene Sicherheitsvorkehrungen getroffen, um sicherzustellen, dass die Malware keinen Schaden anrichtet.

Ein wesentlicher und aufwendiger Teil des Projekts bestand in der Datenerfassung und -aufbereitung. Die Suche nach geeigneter Goodware war eine Herausforderung, aber notwendig, um qualitativ

hochwertige Trainingsdaten für das maschinelle Lernmodell zu gewährleisten. Details dazu finden sich im Kapitel 3. Bei Malware gestaltete sich die Datenbeschaffung einfacher, da zahlreiche öffentlich zugängliche Datenbanken mit Malware-Proben existieren.

Nach der Ausführung der Malware- und Goodware-Analyse auf der Windows-Maschine war es erforderlich, die daraus gewonnenen Daten zu sichern und aufzubereiten, um sie für die weitere Verarbeitung bereitzustellen.

Mit dem erfolgreichen Abschluss dieses Projekts wurde eine automatisierte Testumgebung für die dynamische Malware-Analyse geschaffen. Sie ermöglicht eine effiziente und schnelle Erkennung und Analyse von Malware und trägt somit zu einer verbesserten IT-Sicherheit bei.

## 1.4 Struktur der Arbeit

Diese Bachelorarbeit folgt einer traditionellen Struktur, wie sie in zahlreichen wissenschaftlichen Arbeiten angewendet wird.

Das einleitende Kapitel 2 legt die Grundlagen dieser Arbeit fest. Hier werden wichtige Definitionen, Konzepte und Methoden vorgestellt, um dem Leser ein fundiertes Verständnis des Themas und der Hintergründe dieser Arbeit zu ermöglichen.

In Kapitel 3 werden die spezifischen Anforderungen an das zu entwickelnde System definiert und erläutert. Weiterhin werden die getroffenen Design-Entscheidungen sowie die detaillierte Analyse des Systems, die im Vorfeld durchgeführt wurde, erläutert.

Kapitel 4 beschreibt den Prozess der Systemimplementierung. Dieser Abschnitt basiert auf den Erkenntnissen und Entscheidungen, die in Kapitel 3 beschrieben und getroffen wurden.

In Kapitel 5 werden die erzielten Ergebnisse dargestellt und es wird ein Ausblick gegeben, wie diese Arbeit in der Zukunft weitergeführt oder erweitert werden könnte. Abschließend werden in Kapitel 6 die Schlussfolgerungen dieser Arbeit gezogen.

Für interessierte Leser befindet sich der Quellcode der in dieser Arbeit verwendeten Skripte im Anhang A.

## 2 Malware: Grundlagen und Typen

### 2.1 Einführung in die Malware

Malware, eine Kontraktion aus "malicious software" (böartige Software), bezeichnet Programme, die entwickelt wurden, um Systeme oder Netzwerke zu schädigen. Sie stellt ein stetig wachsendes Problem dar, dem sich alle Internetnutzer gegenübersehen, da Cyber-Kriminelle zunehmend professioneller und innovativer werden. Das IT-Sicherheitsunternehmen Kaspersky berichtete 2021, dass sie täglich 380.000 neue Malware-Proben identifizierten. Nur ein Jahr später stieg diese Zahl auf 400.000 neue Malware-Proben pro Tag – ein Trend, der sich fortzusetzen scheint [24].

Laut AV-Atlas gab es seit Anfang 2023 23,4 Millionen neue Malware-Exemplare, wodurch die Gesamtzahl der seit 1984 erfassten Malware auf über 1,254 Milliarden anstieg [4]. Um den Umfang und die Bedrohung durch Malware zu verstehen, ist es zunächst wichtig, die verschiedenen Arten von Malware und ihre Ziele zu kennen. Dabei sollen auch Verbreitungswege von Malware, Faktoren, die ihre Erkennung erschweren, gegenwärtig verfügbare Analysemethoden und gegenwärtige Ansätze zur Malware-Analyse beleuchtet werden.

### 2.2 Evolution der Malware über die Jahre

Im Laufe der Jahre hat sich Malware auf unvorhergesehene Weise entwickelt. Zu Zeiten der ersten Viren und Würmer standen meist harmlose oder keine böartigen Ziele im Vordergrund. Heutzutage ist dies jedoch weit von dem entfernt, was man damals hätte erahnen können. Was damals als ein Scherz angefangen hat, hat sich heute in die größte Gefahr die wir in der Digitalen Welt begehen können entwickelt. Im Weiteren möchten wir kurz auf die verschiedenen Entwicklungsstufen und die Geschichte von Malware eingehen.

#### 2.2.1 Erste Phase der Malware-Entwicklung 1949 - 1991

Als Malware ursprünglich entwickelt wurde, lag die Absicht nicht darin, Schaden anzurichten, zu stehlen oder zu manipulieren. Dennoch hat sie sich zu einer ernsthaften Bedrohung für unsere Gesellschaft entwickelt. Der Begriff Malware wurde erstmals 1990 von dem Informatiker und Sicherheitsforscher Yisrael Radaï verwendet. Malware gab es jedoch schon lange vorher [1]. Das erste Beispiel für Malware, wäre der von John Von Neumann "self-reproducing string of code", welchen er 1949 vorgestellt hatte [1]. Dabei handelte es sich um einen selbstreplizierenden Automaten, der sich bei jeder Ausführung automatisch in eine neue Version verwandelte.

Die ersten Malware in dieser Phase waren oftmals dafür angewendet, um auf Sicherheitslücken in MS-DOS System aufmerksam zu machen. Diese haben in der Anfangsphase oft zu Systemabstürzen gesorgt, da zu viele Systemressourcen durch diese beansprucht wurden. Die Verbreitung der Viren und Würmer erfolgte in dieser Phase hauptsächlich durch infizierte Disketten oder durch das ARPANET statt [1]. Während dieser Zeit wurde auch die erste anerkannte Definition eines Virus von Fred Cohen definiert, welche lautete: “Ein Programm, das andere Programme infizieren kann, indem es sie so verändert, dass sie möglicherweise eine weiterentwickelte Kopie von sich selbst beinhalten” [11].

Ein charakteristisches Merkmal von Malware dieser Generation war, dass sie nicht versuchen, sich vor dem Benutzer zu verbergen. In nahezu allen Fällen wurden dem Benutzer auf der (Grafische Benutzeroberfläche (GUI)) eine Nachricht hinterlassen. Der erste bekannte Wurm von Robert H. Thomas, der als “Creeper Worm” bekannt war, hatte die Nachricht hinterlassen “I’m the creeper: catch me if you can” [17]. Ein weiteres Beispiel ist der Elk Cloner Virus, welcher ein kurzes Gedicht angezeigt hatte: “It will get on all your disks; It will infiltrate your chips; Yes, it’s Cloner!” [47].

Im Jahr 1986 entwickelten zwei pakistanische Brüder den Brain-Virus, um auf die Sicherheitsmängel von MS-DOS als Plattform aufmerksam zu machen. Es handelte sich dabei um den ersten bekannten Virus in der Form, wie wir ihn heute kennen. Der Brain-Virus infizierte weltweit zahlreiche Computersysteme und verbreitete sich über infizierte Disketten. Die Absicht hinter dem Brain-Virus bestand darin, die Schwachstellen im MS-DOS-System aufzudecken und das Potenzial der Gefahren von solcher Malware ins Bewusstsein zu rufen [1][18].

Nur zwei Jahre später wurde der wohl bekannteste Wurm, der “Morris Worm”, entwickelt, der frühzeitig das Potenzial und den Schaden aufzeigte, den ein Wurm verursachen kann [32]. Der Morris-Wurm war einer der ersten Würmer, die das Internet infizierten, und seine Verbreitung hatte große Auswirkungen auf das damals noch junge Internet. Der Wurm infizierte Tausende von Computern innerhalb weniger Stunden und verursachte erhebliche Störungen in Netzwerken und Systemen, indem er verschiedene Schwachstellen ausnutzte um sich in diesen System zu verbreiten. Die Verbreitung des Morris-Wurms führte zur Entwicklung neuer Sicherheitsmaßnahmen und der Schaffung von Computer Emergency Response Team (CERT)-Teams, um auf zukünftige Bedrohungen zu reagieren. Der Morris-Wurm gilt als eines der ersten Beispiele für eine groß angelegte Cyber-Attacke und hat dazu beigetragen, das Bewusstsein für die Bedeutung der Cybersicherheit zu schärfen[32].

### 2.2.2 Zweite Phase der Malware-Entwicklung 1992 - 1999

Während dieser Zeit konzentrierten sich Hacker und Angreifer auf das Windows-Betriebssystem, da es aufgrund seiner Einfachheit und leistungsstarken GUI viele Benutzer anzog. Die meisten Malware-Arten aus dieser Zeit waren daher für das Windows-Betriebssystem entwickelt[1]. Diese Phase war im Allgemeinen bekannt für Windows-Malware, die ersten E-Mail-Würmer und Makro-Würmer. Auch der Beginn der ersten Anti-Viren-Programme prägte diese Zeit.

Die erste Malware für Windows wurde 1992 unter dem Namen WinVir entwickelt [36]. Dieses Virus infizierte ausführbare Programme unter Windows und kopierte sich selbst in andere Programme und Systemdateien, hatte aber keine schwerwiegenden Folgen. Allerdings galt dies nicht für die meisten anderen Malware-Arten, die zu dieser Zeit entstanden sind. Viren wie V-Sign,

One-Half oder Slovak Bomber beeinträchtigten das System erheblich und verschlüsselten teilweise sogar Daten [1].

Der wohl berühmteste Virus dieser Zeit war der Melissa-Virus, welcher sich an eine E-Mail angehängt und durch Ausnutzung einer Schwachstelle in Microsoft Word sich verbreitete hat [16]. Er verschickte sich selbst an die ersten 50 Kontakte im Adressbuch einer infizierten Person. Im Anhang befanden sich Nacktbilder, deren Öffnung den Virus aktivierte. Dadurch wurde die Neugier des Nutzers ausgenutzt, um den Virus auszuführen. Melissa verlangsamte weltweit Netzwerke und Systeme oder legte sie sogar komplett lahm, was zu hohen wirtschaftlichen Verlusten führte. Man rechnet von einem Schaden von \$80 Millionen allein in Nord-Amerika. Da die E-Mail-Kommunikation zu dieser Zeit noch neu war, waren den Nutzern keine Vorsichtsmaßnahmen bekannt, und sie wussten nicht, dass sie keine unbekannten Anhänge öffnen sollten.

### 2.2.3 Dritte Phase der Malware-Entwicklung 2000 - 2008

Die dritte Phase der Malware-Entwicklung begann durch die weite Verbreitung des Internets. Die Malware wurde in dieser Phase hauptsächlich über E-Mail-Anhänge, kostenlose Downloads von kompromittierten Webseiten oder offene Netzwerkfreigaben übertragen [1]. Zu Beginn war Sicherheit in der IT und im Internet noch kein prominentes Thema, und in den meisten Fällen fand man nicht mehr als ein Anti-Viren-Programm als Schutzmaßnahme vor.

Würmer und Viren haben sich in dieser Zeit über offene Netzwerke verbreitet und jedes Gerät in den Netzwerken infiziert, welche keine Schutzmaßnahmen hatten und sich dann von dort aus weiter verbreitet. ILoveYou- oder Anna-Kurnikova-Virus, waren ähnlich wie der Melissa-Virus, Viren die an E-Mails angehängt wurden und durch Social Engineering der Nutzer dazu gelockt haben, diese Datei zu öffnen [1].

Der erste bekannte Wurm, der nach dem Morris Worm gefunden war, war der Code Red Worm, welcher innerhalb von nur 14 Stunden 359.000 Systeme infiziert hatte. Er hatte sich durch das Internet durch eine Buffer-Overflow-Schwachstelle im Microsoft's Internet Information Server (IIS) Web Server verbreitet und unzählige Systeme weltweit infiziert. Es hat dann die infizierten Geräte dafür genutzt, einen DDoS-Angriff auf das Weiße Haus zu starten. Ein weiterer bekannter Wurm, welcher noch gefährlicher als der Code Red Wurm war, war Nimda (Admin rückwärts). Dieser nutzte bereits mehrere verschiedene Schwachstellen aus, um sich noch schneller und weiter verbreiten zu können [29].

In dieser Phase sind auch die ersten weit verbreiteten Malware für Apache und Linux Server entstanden. Als Beispiel hatte Slammer damals in den USA dafür gesorgt, dass eine große Anzahl an Servern abgestürzt ist, was zu Verspätungen in Abflügen kam, nicht funktionsfähigen Bankautomaten, sowie auch, dass der Notruf nicht mehr erreichbar war in Bereich von Washington DC. Es hatte es sogar geschafft Systeme in einem Atomkraftwerk in Ohio zu infizieren und sich Zugang zu diesen zu verschaffen [1].

Die wahrscheinlich bekannteste Malware dieser Generation ist der Conficker Worm. Im Jahr 2008 tauchten der Conficker-Wurm und seine Varianten in freier Wildbahn auf und infizierten rund 9 Millionen Computer, indem sie Antiviren-Software ausschalteten, sich über das Intranet verbreiteten und USB-Laufwerke infizierten [1]. Conficker nutzte eine Sicherheitslücke in Windows aus. Er verbreitete sich, indem er schwache Passwörter knackte, nutzte aber nie sein sehr komplexes

Botnet-Netzwerk für einen einer Angriffe aus [1], daher ist die Absichten des Autors bis heute noch unbekannt.

#### 2.2.4 Vierte Phase der Malware-Entwicklung 2005 - 2016

Die vierte Phase der Malware-Entwicklung, die von 2005 bis 2016 andauerte, zeichnete sich durch die Einführung von Rootkits und Ransomware aus. Detaillierte Informationen zu diesen Malware-Arten finden sich im Abschnitt 2.3. Die gängigsten Infektionspfade für Malware in dieser Phase umfassten Phishing-E-Mails, Remote-Desktop-Protokolle und Downloads von kompromittierten Websites sowie über USB-Sticks oder andere Wechselmedien. Die in dieser Phase entwickelte Malware zielte vornehmlich auf finanziellen Gewinn oder auf illegale Kontrolle über die infizierten Computer ab [1].

In dieser Zeit versuchten sowohl Angreifer als auch andere Unternehmen, ohne das Wissen der Nutzer, ein Rootkit auf dem Zielsystem zu installieren, um vollen Zugriff auf das System zu erhalten. Ein Wurm, ähnlich dem ILoveYou-Wurm, namens Storm Worm, installierte auf infizierten Geräten ein Rootkit, das andere Malware verstecken konnte [1]. Storm Worm versuchte, Nutzer durch Schlagzeilen wie “230 Tote durch Sturm in Europa” dazu zu bringen, E-Mail-Anhänge zu öffnen, anstatt durch Liebesversprechen.

Im Jahr 2005 verbreitete sich die Malware GPCode. Ein Trojaner, der sich als Bewerbung ausgab. Nach dem Öffnen der Datei wurde die Ransomware jedoch aktiviert und verschlüsselte alle Daten auf dem System mit dem RSA-Algorithmus. Seitdem sind unzählige weitere Ransomware-Varianten in Umlauf gebracht worden, die alle das gleiche Ziel verfolgen: Daten zu verschlüsseln, sodass der Benutzer gegen eine bestimmte Geldsumme wieder Zugriff darauf erhält. Im Jahr 2011 gab es einen extremen Anstieg von Ransomware-Angriffen, da es zu dieser Zeit sehr einfach war, nicht zurückverfolgbare Überweisungen zu tätigen. In den ersten zwei Quartalen dieses Jahres wurden 30.000 Ransomware-Angriffe verzeichnet, die sich im dritten Quartal bereits verdoppelt hatten [1][37].

#### 2.2.5 Fünfte Phase der Malware-Entwicklung 2010 - aktuell

Die fünfte und aktuelle Phase der Malware-Entwicklung umfasst hauptsächlich Malware, die für Spionage und Sabotage eingesetzt wird. Während die Malware in den vorherigen Phasen hauptsächlich von Cyberkriminellen entwickelt wurde, sind Malware-Programme dieser Phase vorwiegend Produkte von Regierungen, Polizei und Geheimdiensten verschiedener Nationen. Malware in dieser Phase wird als Advanced Persistent Thread (APT) bezeichnet. Das beste Beispiel für eine APT ist Stuxnet [10].

Stuxnet ist eine äußerst raffinierte und fortschrittliche Malware, die erst nach der Erfüllung ihres Ziels entdeckt wurde. Stuxnet wurde entwickelt, um die Urananreicherung im Iran zu sabotieren. Durch einen USB-Stick gelang es, das Virus in das interne Netzwerk des Kraftwerks einzuschleusen, wo es gezielt von Gerät zu Gerät weitergegeben wurde, bis es das Ziel-System, einen Siemens Industriecontroller, erreichte. Dort hat es unbemerkt die Urananreicherung sabotiert, ohne dass dies über Jahre hinweg jemand bemerkte. Nachdem es sein Ziel erreicht hatte, hat es sich am 24.7.2012 selbst deaktiviert und gelöscht. Seitdem ist Stuxnet wahrscheinlich die am meisten untersuchte Malware der Welt, wobei man Spuren davon bereits im Jahr 2005 entdeckt hat. Dies



lässt vermuten, dass APTs schon viel früher von Geheimdiensten und Regierungen eingesetzt wurden. Viele vermuten, dass Stuxnet von amerikanischen und israelischen Geheimdiensten entwickelt wurde, da im Code Hinweise auf amerikanische Geheimdienste gefunden wurden. Die USA und Israel bestreiten jedoch alle Vorwürfe [1][6].

WannaCry war die erste Ransomware, die sich unter Ausnutzung einer von der National Security Agency (NSA) entwickelten Sicherheitslücke verbreitete [1]. Sie verschaffte sich Zugang zu wichtigen Computersystemen in rund 150 Ländern, wie Russland, China und den USA. Sie infizierte zahlreiche Krankenhäuser, Banken, Telekommunikationsunternehmen, Lagerhäuser und Industrieunternehmen. Die US-Regierung geht davon aus, dass Nordkorea hinter dem WannaCry-Angriff steckt.

APTs sind äußerst komplex und kostenintensiv entwickelte Malware-Programme, die von Fachleuten konzipiert werden. Im Unterschied zu anderen Formen von Malware liegt der Fokus hier nicht auf finanziellen Gewinn, sondern vielmehr auf Sabotageakte gegenüber Feinden. Oft wird dieser Typ von Angriffen auch als "Digitaler Kriegsführung" bezeichnet[27].

## 2.3 Malware-Typen

Zu Beginn ist es ratsam, sich zunächst mit den am häufigsten auftretenden Malware-Typen vertraut zu machen. Wir haben zwar über verschiedene Malware bereits geredet, jedoch nicht beschrieben was genau die Charakteristiken dieser Malware sind. Im Folgenden sind die zehn häufigsten Arten von Malware aufgeführt, die von Norton Antivirus 2022 [44] identifiziert wurden:

### 2.3.1 Virus

Ein Computervirus, vergleichbar mit einem biologischen Virus, verbreitet sich von System zu System, indem es sich an ausführbare Programme anheftet und sich durch deren Ausführung in einem System oder Netzwerk ausbreitet. Ein Virus kann nicht eigenständig existieren und muss sich stets an eine legitime Datei binden. Dabei kann ein Virus unerwartete oder schädliche Auswirkungen haben, beispielsweise die Systemsoftware durch Beschädigung oder Zerstörung von Daten beeinträchtigen. Zudem kann sich ein Virus im Normalfall nicht selbständig verbreiten und ausführen. In der Regel ist menschliche Interaktion erforderlich. Dennoch gibt es Viren, die nach der initialen Ausführung dazu in der Lage sind, sich ohne menschliche Interaktion zu verbreiten. Diese Viren sind als "selbstreplizierend" oder "selbstverbreitend" bekannt[11].

### 2.3.2 Wurm

Ein Wurm weist Ähnlichkeiten zu einem Virus auf, verbreitet sich jedoch eigenständig von System zu System, indem er Schwachstellen in Netzwerken oder Computersystemen ausnutzt und sich so fortlaufend weiter ausbreitet. Dies geschieht durch nicht gepatchte Sicherheitslücken, Zero-Day-Exploits oder andere Methoden, um weitere Systeme zu infizieren. Zudem besitzen Würmer die Fähigkeit zu erkennen, ob ein System bereits infiziert ist, um eine mehrfache Infektion zu vermeiden und damit eine Unbrauchbarkeit des Systems zu verhindern[44].



### 2.3.3 Trojaner

Ein Trojanisches Pferd, oft als “Trojaner” bezeichnet, ist eine Art von Schadsoftware, die sich als legitime Anwendung oder nützliches Werkzeug ausgibt, um den Benutzer zum Herunterladen und Ausführen zu verleiten. Nach der Aktivierung auf dem System des Benutzers kann ein Trojaner eine Reihe schädlicher Aktionen ausführen, wie zum Beispiel das Entwenden von Daten, das Übernehmen der Systemkontrolle oder das Installieren zusätzlicher Malware. Ursprünglich wurden Trojaner entwickelt, um dem Angreifer über eine Hintertür Remote-Zugang zum System zu ermöglichen. Heute jedoch besitzen sie ein wesentlich umfangreicheres Schadenspotential. Trojaner sind häufig schwer zu erkennen, da sie in der Regel verdeckt agieren und den Benutzer täuschen. Ein Trojanisches Pferd kann verschiedene Typen von Malware enthalten, darunter Viren, Würmer oder Spyware[44].

### 2.3.4 Ransomware

Ransomware, auch als Lösegeld-Software bekannt, gehört zu den bekanntesten und am häufigsten in den Medien diskutierten Malware-Arten. In den letzten Jahren hat Ransomware immer mehr an Bedeutung gewonnen, da eine wachsende Anzahl von Unternehmen und Organisationen von Ransomware-Angriffen betroffen ist. Diese Art von Malware infiziert und verschlüsselt Dateien auf einem System oder Netzwerk, wodurch sie unzugänglich werden, es sei denn, man verfügt über den entsprechenden Entschlüsselungsschlüssel. Angreifer fordern üblicherweise Lösegeldzahlungen, oft in Form von Kryptowährungen, im Austausch für den Entschlüsselungsschlüssel der betroffenen Dateien[33]. Während eines Ransomware-Angriffs ist das betroffene System oder Netzwerk größtenteils unbrauchbar, und auf den Bildschirmen der betroffenen Geräte werden in der Regel Informationen über die Angreifer, deren Forderungen und Anweisungen zur Zahlung angezeigt. Es gibt vorrangig zwei Arten von Ransomware: Locker-Ransomware und Crypto-Ransomware [1][39]. Locker-Ransomware verhindert, dass Benutzer auf ihr System zugreifen können, während Crypto-Ransomware Benutzerdaten durch Anwendung von Verschlüsselungsalgorithmen unbrauchbar macht. Von diesen beiden ist Crypto-Ransomware die verheerendere Variante.

### 2.3.5 Botnetz

Als Botnet bezeichnet man ein Netzwerk von infizierten Computern oder Geräten, die unter der Kontrolle eines Angreifers stehen und für illegale Aktivitäten genutzt werden können[44]. Die Software, die diese Kontrolle ermöglicht, wird als Bot oder Bot-Software bezeichnet. Botnets werden häufig für verschiedene Zwecke eingesetzt, darunter DDoS- (Distributed Denial-of-Service-) Angriffe auf Websites oder Online-Dienste, Versenden von Spam- und Phishing-E-Mails, Stehlen von Passwörtern und persönlichen Daten sowie für die Infektion weiterer Geräte. Die betroffenen Personen oder Organisationen sind sich oft nicht darüber im Klaren, dass ihre Systeme Teil eines Botnets sind, da die Bot-Software in der Regel darauf abzielt, im Hintergrund zu arbeiten und die Funktionalität des Systems möglichst wenig zu beeinträchtigen. Auf diese Weise kann der Angreifer das infizierte System länger für seine Zwecke nutzen.

### 2.3.6 Adware

Adware ist eine häufig anzutreffende Art von Malware im Internet. Sie verfolgt die Internetaktivitäten der Nutzer, um gezielte Werbung zu schalten. Adware zeigt unerwünschte Werbung in Form von Pop-up-Anzeigen oder Bannern an und kann sogar den Standardbrowser oder die Startseite ändern. Obwohl Adware auf den ersten Blick als relativ harmlos erscheinen mag, kann sie dennoch negative Auswirkungen auf das betroffene System haben. Sie kann die Systemleistung beeinträchtigen, indem sie Ressourcen beansprucht und den Browser verlangsamt. In einigen Fällen kann Adware auch als Einfallstor für andere schädliche Software dienen, die sich auf dem System einnisten und weiteren Schaden verursachen kann. Das Hauptziel dieser Malware ist es, durch gezielte Werbung und das Sammeln von Nutzerdaten finanziellen Gewinn zu erzielen[44].

### 2.3.7 Spyware

Spyware ist eine Art von Software, die ohne Wissen des Nutzers auf dem Gerät installiert wird und persönliche Daten stiehlt, um sie an Dritte weiterzuleiten. Im Gegensatz zur Adware, die sich hauptsächlich auf das Sammeln von Browser-Nutzerdaten für gezielte Werbung konzentriert, erfasst Spyware auch persönliche Informationen, Passwörter, Tastatureingaben usw. [44]. Diese Daten werden dann an die Entwickler der Spyware oder andere Interessenten weitergegeben, die sie für ihre Zwecke nutzen können. Es ist wichtig zu beachten, dass Spyware ein sehr umstrittenes Thema ist und immer wieder in den Medien diskutiert wird. Es ist auch Teil der breiteren Debatte über Datenschutz und digitale Rechte.

### 2.3.8 Rootkits

Rootkits zählen zu den gefährlichsten Malware-Arten, da sie Angreifern unbemerkt umfassenden Zugriff auf ein System gewähren[1][46]. In Quelle [3] wird ein Rootkit definiert als "jede Software, die fortlaufend privilegierten Zugriff auf einen Computer ermöglicht, während sie ihr Vorhandensein und andere Informationen aktiv vor Administratoren verbirgt, indem sie die Standardfunktionen des Betriebssystems oder anderer Anwendungen untergräbt". Sie können sich tief im Betriebssystem verankern und sogar Kernel-Ebene-Privilegien erlangen. Rootkits sind in der Lage, sich vor Antivirenprogrammen zu verstecken und als reguläre Software zu erscheinen, wodurch sie Sicherheitsmechanismen umgehen können. Mithilfe eines Rootkits kann weitere Malware ins System eingeschleust werden, die dann ebenfalls von Antivirenprogrammen nicht erkannt wird. Da Rootkits Sicherheitsfunktionen umgehen und schwer zu entdecken sind, gestaltet sich ihre Entfernung oftmals als äußerst schwierig. In vielen Fällen bleibt keine andere Option, als das System vollständig neu aufzusetzen, um sicherzustellen, dass alle infizierten Komponenten beseitigt werden. Es gibt 5 Haupttypen von Rootkits: Hardware- (Firmware-) Rootkit, Bootloader-Rootkit, Speicher-Rootkit, Anwendungs-Rootkit und Kernel-Mode-Rootkit [9]. Der Hauptunterschied zwischen diesen Rootkit-Typen besteht darin, dass sie unterschiedliche Methoden anwenden, um dieselben Berechtigungen zu erlangen. Zudem variieren sie in Bezug auf die Schwachstellen, die sie ausnutzen, und wo sie sich letztendlich im System einnisten.

### 2.3.9 Fileless Malware

Fileless Malware, auch oft als “In-Memory-Execution” bezeichnet, stellt eine neuartige und raffinierte Art von Schadsoftware dar. Sie hinterlässt keine Dateien auf der Festplatte, sondern nutzt den Arbeitsspeicher sowie vorhandene Systemtools, um schädliche Aktionen auszuführen [44]. Diese Art von Malware verwendet häufig Skriptsprachen wie PowerShell und ist daher schwerer von herkömmlichen Antivirenprogrammen zu erkennen. Sie tarnt sich als legitime Systemprozesse und bleibt dadurch häufig unentdeckt, wodurch sie Sicherheitsmechanismen umgehen kann. Aus diesem Grund ist es besonders wichtig, das System und die darauf befindliche Software regelmäßig zu aktualisieren, denn Fileless Malware sucht gezielt nach Schwachstellen im System, anstatt selbst eine Schwachstelle zu installieren.

### 2.3.10 Malvertising

Malvertising unterscheidet sich von Adware dadurch, dass es sich um eine Art von Schadsoftware handelt, die heruntergeladen wird, sobald der Benutzer auf eine infizierte Werbeanzeige klickt. Im Gegensatz zur Adware besteht das Ziel nicht darin, Geld durch die Werbung selbst zu verdienen, sondern die Malware wird als Werbung getarnt, in der Hoffnung, dass ein Benutzer dazu verleitet wird, darauf zu klicken. Statt auf der Seite des beworbenen Produkts zu landen, lädt der Benutzer unwissentlich Malware herunter, die anschließend Schaden auf dem System anrichten kann[44].

### 2.3.11 Weitere Malware-Arten

Natürlich gibt es noch viele weitere bekannte Malware-Arten, wie beispielsweise Keylogger, Sniffer, Spam usw. Deshalb ist es von großer Bedeutung, kontinuierlich an der Malware-Erkennung zu arbeiten und diese weiterzuentwickeln. Da sich Malware stetig weiterentwickelt und der Bereich immer lukrativer wird, können Malware-Entwickler damit erhebliche finanzielle Gewinne erzielen. Es ist daher entscheidend, dass Unternehmen und Privatpersonen stets wachsam bleiben, ihre Systeme regelmäßig aktualisieren und geeignete Sicherheitsmaßnahmen ergreifen, um sich vor diesen Bedrohungen zu schützen.

## 2.4 Verhalten und Eigenschaften von Malware

Um Malware effektiv bekämpfen und analysieren zu können, ist es entscheidend, die Verhaltensweisen und Eigenschaften von Malware zu verstehen. In diesem Abschnitt werden wir einige der häufigsten Verhaltensmuster und charakteristischen Merkmale von Malware untersuchen. Durch diese Untersuchung können wir besser nachvollziehen, wie sich Malware entwickelt und anpasst, um Sicherheitsmaßnahmen zu umgehen und ihre Ziele zu erreichen.

### 2.4.1 Statische vs Dynamische Malware

Es gibt unzählige Möglichkeiten, Malware zu kategorisieren. In verschiedener Literatur wird Malware nach unterschiedlichen Merkmalen eingeteilt, sodass es keine allgemeingültige oder falsche Kategorisierung gibt. Die Einteilung hängt vom jeweiligen Nutzen und Kontext ab. Eine verbreitete Methode besteht darin, Malware nach diesen zwei Merkmalen zu sortieren: Wie vermehrt sich die Malware und welche Aktionen führt die Malware-Nutzlast aus? Andere Definitionen berücksichtigen beispielsweise Angriffsvektoren oder die Auswirkungen der Malware. Eine weit verbreitete und anerkannte Kategorisierung unterscheidet jedoch zwischen statischer und dynamischer Malware.

Statische Malware, auch als Erste Generation Malware bezeichnet, wird anhand der Infektionstrategie klassifiziert. Die Struktur der Malware bleibt von Anfang bis Ende der Infektion unverändert. Im Gegensatz dazu verändert sich dynamische Malware, auch als Zweite Generation Malware bekannt, nach jeder Infektion. Dabei verwandelt sich die Malware in eine neue Form, wobei sich, bis auf die eigentliche Nutzlast, die Struktur der Malware vollständig ändert[38]. Dynamische Malware kann zudem noch weiter in Encrypted, Oligomorphic, Polymorphic und Metamorphic Malware unterteilt werden[38].

#### Verschlüsselte Malware

Verschlüsselte Malware war die erste Verschleiertechnik, welche in der zweiten Generation von Malware verwendet wurde. Der erste bekannte Virus, der diese Technik verwendete, war Cascade aus dem Jahr 1987 [52]. Die Malware besteht hier aus zwei Teilen: dem Virus-Körper (Virus Body) und dem Ver-/Entschlüssler (Decryptor), wie in 2.1 dargestellt. Die Malware ist hier ziemlich einfach aufgebaut. Der Decryptor wird verwendet, um den Code im Virus-Körper zu verschlüsseln und zu entschlüsseln. Der Virus-Körper ist die eigentliche Malware, der von Bedeutung ist, wenn er entschlüsselt wird. Der Prozess bei dieser Malware läuft folgendermaßen ab: Sobald die Malware ausgeführt wird, setzt die Entschlüsselungsroutine ein, welche den Virus-Körper entschlüsselt und in die Maschinensprache übersetzt. Die Verschlüsselung der Daten im Virus-Körper kann auf unterschiedlichste Art und Weise geschehen. Zu Beginn wurden noch einfachere Methoden verwendet, wie 1-zu-1-Mapping, um den Code Byte für Byte zu verwandeln. Es können auch anspruchsvollere Verschlüsselungstechniken verwendet werden, die umkehrbare Befehle verwenden, z.B. ADD oder XOR mit Zufallsschlüsseln. Darüber hinaus kann der Verschlüsselungsschlüssel ein konstanter Wert oder ein gleitender variabler Wert sein, der durch einen speziellen Algorithmus erzeugt wird. Herkömmliche Virens Scanner können solche Viren nicht anhand ihrer Signatur erkennen, da zunächst der Körper entschlüsselt werden muss, um den Inhalt zu erkennen. Allerdings ist es teilweise möglich, solche Viren anhand des Decryptors zu erkennen, wenn dessen Byte-Abfolge lang genug ist.

#### Oligomorphe Malware

Als Antwort auf die Fähigkeit von Antivirenprogrammen, verschlüsselte Malware zu erkennen, entstanden oligomorphe Viren. Diese unterscheiden sich von ihren Vorgängern dadurch, dass sie nicht nur einen, sondern mehrere Decryptoren besitzen und verwenden. Anfangs waren es nur

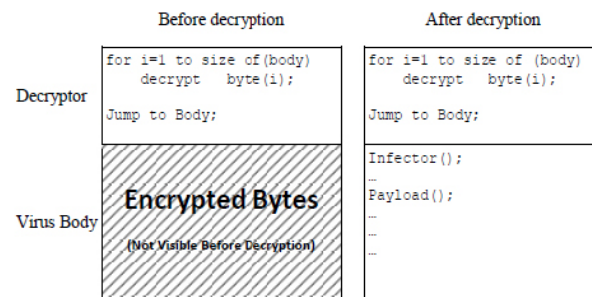


Abb. 2.1: Encrypted Malware [34]

wenige Decryptoren, die eine solche Malware besaß, doch recht bald fand man Malware mit mehreren hundert Decryptoren [38]. Jedes Mal, wenn die Malware ausgeführt wird, verwendet sie einen zufälligen Decryptor und verschlüsselt sich nach der Ausführung erneut mit einem anderen zufälligen Decryptor[34], um die Analyse so schwierig wie möglich zu machen. Abbildung 2.2 zeigt, dass zu Beginn der Infektion ein anderer Decryptor verwendet wird und nach der Infektion die Malware erneut durch einen anderen Decryptor verschlüsselt wird. Im Gegensatz zu verschlüsselten Viren muss die Antiviren-Engine alle möglichen Decryptor-Instanzen überprüfen, anstatt nur nach einem Decryptor zu suchen, was mehr Zeit benötigt [34]. Eine allgemeine Empfehlung hier lautet, keine signaturbasierte Analyse zur Erkennung dieser Art von Malware zu verwenden.

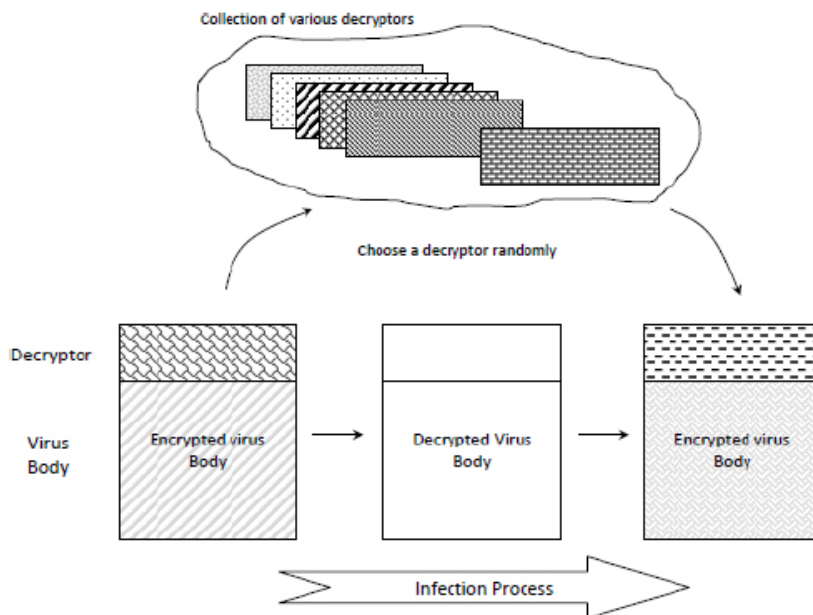


Abb. 2.2: Oligomorphic Malware [34]

### Polymorphe Malware

Polymorphe Malware ist als Reaktion auf Antivirenprogramme entstanden, die nun in der Lage waren, auch oligomorphe Malware zu erkennen[40]. Der Unterschied zu oligomorpher Malware besteht darin, dass polymorphe Malware Millionen von verschiedenen Decryptoren besitzen kann[38], um einer Erkennung durch signaturbasierte Techniken zu entgehen. Wie man in Abbildung 2.3 erkennen kann, besteht die Malware wieder aus zwei Teilen: dem Decryptor und dem Virus-Körper. Allerdings kommt hier die Mutations-Engine hinzu, die einen Verschlüsselungs- und Entschlüsselungsalgorithmus beinhaltet. Wird die Malware ausgeführt, wird der Virus-Körper entschlüsselt und durch die Mutations-Engine in einen völlig neuen Virus umgewandelt, der durch keine signaturbasierten Methoden erkannt werden kann. Die Malware wird durch verschiedene Verschleierungsmethoden in der Mutations-Engine erzeugt, auf die wir in Abschnitt 2.4.3 näher eingehen werden. Die einzige Möglichkeit, solche Malware mit signaturbasierten Methoden zu erkennen, besteht darin, die Malware in der Antiviren-Sandbox auszuführen und den Virus-Körper zu analysieren.

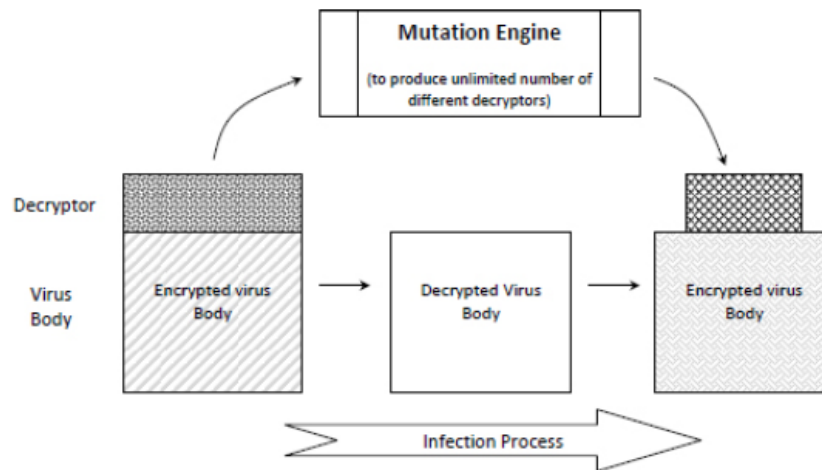


Abb. 2.3: Polymorphic Malware [34]

### Metamorphe Malware

Anders als die bisher besprochenen Malware-Arten besitzt Metamorphe Malware keinen verschlüsselten Code und folglich auch keinen Decryptor. Ähnlich wie die polymorphe Malware verfügt sie jedoch über eine Mutationsengine[34]. Bei dieser Form von Malware wird der Virus-Körper bei jeder Ausführung neu mutiert, was in der Literatur auch als “Body-Polymorphics”[34] bezeichnet wird. Trotz möglicher Änderungen in Struktur, Code-Sequenz, Größe und syntaktischen Eigenschaften bleibt das Verhalten des Virus konstant. Dieses Phänomen wird in Abbildung 2.4 dargestellt. Trotz ständiger Mutationen in ihrer physischen Form und internen Struktur führt die Malware stets die gleichen Aktionen aus.

Eine typische Struktur einer solchen Metamorphen Engine, zu sehen in Abbildung 2.5, beinhaltet einen Disassembler, einen Code Analyzer, einen Code-Transformer und einen Assembler[34][50].

Bei Ausführung der Malware sucht diese zunächst nach dem Startpunkt im eigenen Code und wandelt diesen durch den internen Disassembler in Assemblerbefehle um. Der Codeanalysator ist dabei für die Bereitstellung von Informationen für das Code-Transformer-Modul zuständig. Der Code-Transformer benötigt möglicherweise einige Informationen, wie die Struktur und das Flussdiagramm des Programms, Unterprogramme, Lebensdauer von Variablen und Registern. Diese Informationen helfen dem Code-Transformer, seine Aufgabe effizient zu erfüllen. Der Code-Transformer oder Obfuscator bildet das Herzstück der Mutationsmaschine. Er ist dafür verantwortlich, den Code zu verschleiern und die binäre Sequenz der Malware zu verändern. Die anderen Module sind so konzipiert, dass sie die Anforderungen des Obfuskationsmoduls unterstützen. Der Code-Transformer kann alle Verschleiertechniken verwenden, die in Abschnitt 2.4.3 genannt werden. Das abschließende Modul, der Assembler, wandelt den neu mutierten Assembler-Code des Virus in Maschinencode um.

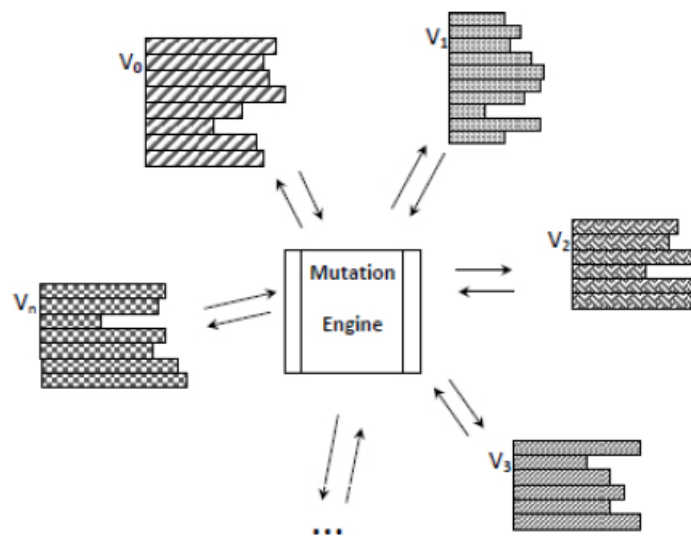


Abb. 2.4: Metamorphic Malware [34]

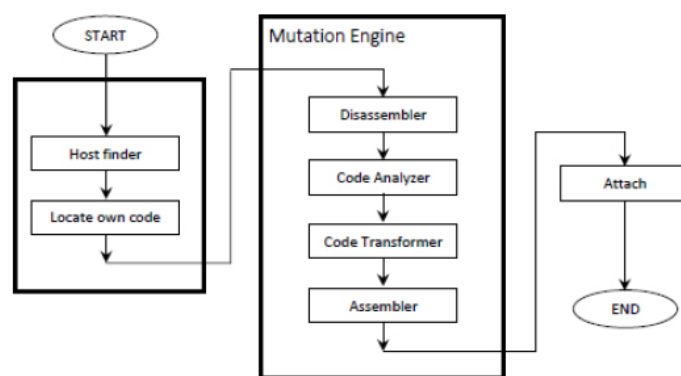


Abb. 2.5: Metamorphic Malware Struktur [34]



### 2.4.2 Malware Verbreitung

Die Verbreitung von Malware ist ein zentraler Aspekt ihres Verhaltens, da dies bestimmt, wie sie sich in Systemen und Netzwerken ausbreitet. Die Methoden zur Verbreitung von Malware haben sich über die Jahre hinweg kontinuierlich weiterentwickelt. Zu Beginn war der einzige Weg, andere Computer zu infizieren, über infizierte Floppy-Disks. Joseph Popp verschickte im Dezember 1989 20.000 Disketten an 90 Länder, getarnt als AIDS-Aufklärungssoftware. Der Versand von 20.000 Disketten muss ein logistischer Albtraum gewesen sein, der das Kopieren, Verpacken und Versenden beinhaltete[30]. In der heutigen vernetzten Welt ist ein physisches Medium zur Verbreitung von Malware nicht mehr erforderlich. Cyberkriminelle nutzen heute die Einfachheit des Internets, um Malware effizient und kostengünstig zu verbreiten. Um Malware effektiv analysieren zu können, ist es notwendig zu verstehen, wie sie sich verbreitet, damit man geeignete Schutzmaßnahmen ergreifen kann. Deshalb betrachten wir die derzeit vier am häufigsten genutzten Wege, wie Malware ein System infiziert[35].

#### Phishing E-Mails

Phishing-E-Mails sind wahrscheinlich die derzeit bekannteste Form der Malware-Verbreitung. Seit der zweiten Phase der Malware-Entwicklung wurden E-Mails genutzt, um Malware zu verbreiten. Anfangs war es für das geschulte Auge oft einfach zu erkennen, ob es sich um eine Phishing-E-Mail handelte oder nicht. Heutzutage sind Malware-Entwickler jedoch in der Lage, sehr gute Kopien von legitimen Seiten nachzuahmen, um die E-Mail so authentisch wie möglich erscheinen zu lassen. Die Zielperson erhält eine E-Mail, die entweder eine eingebettete URL (Uniform Resource Locator) enthält oder einen Downloader im Anhang hat. Wenn eine bössartige E-Mail geöffnet wird, wird ein Download-Mechanismus ausgelöst, der Malware herunterlädt. Dies geschieht, indem eine Verbindung zu einer bössartigen Website hergestellt wird, die Malware hostet, und einen Malware-Download auslöst[30].

Obwohl viele Menschen der Meinung sind, dass sie niemals auf Phishing-E-Mails hereinfallen würden und dass diese leicht zu erkennen sind, zeigen die Statistiken, dass sie die aktuell größte Gefahr darstellen und der Großteil der Malware-Angriffe auf diese Weise durchgeführt wird. Aktuelle Daten zeigen, dass seit 2018 ein Drittel aller Datendiebstähle auf Phishing-E-Mails zurückzuführen ist. Darüber hinaus entsteht im Durchschnitt alle 20 Sekunden eine neue Phishing-Seite. Mehr als 70% aller Empfänger öffnen ihre Phishing-E-Mails. Dies verdeutlicht die Effektivität und Gefährlichkeit solcher Phishing-E-Mails[21]. Daher empfehlen Experten-Unternehmen, jährliche Sicherheitsunterweisungen durchzuführen, um das Bewusstsein der Angestellten zu schärfen. Auch firmeninterne Phishing-Tests können genutzt werden, um zu überprüfen, ob die Schulungen wirksam waren, und gegebenenfalls erneut durchgeführt werden.

#### Drive-by-Downloads

Ein weiterer aktuell sehr populärer Weg für Malware-Entwickler und -Betreiber, Systeme zu infizieren, sind Drive-by-Downloads, da sie keine Nutzerinteraktion benötigen. Bei Drive-by-Download-Angriffen betten die Angreifer bössartige Inhalte entweder in die vom Nutzer besuchte Webseite, die als Landing Page bezeichnet wird, oder in einen Inhalt, der direkt oder indirekt



auf einen Inhalt verweist, welcher normalerweise auf einem kompromittierten Webserver gehostet wird. Wenn ein Browser diese Webseiten aufruft, versucht der bösartige Inhalt, eine Sicherheitslücke im Browser auszunutzen. Ein erfolgreicher Ausnutzungsversuch führt oft dazu, dass Malware heruntergeladen wird, die es dem Angreifer ermöglicht, die Kontrolle über das zugrunde liegende Betriebssystem für verschiedene bösartige Aktivitäten zu übernehmen, z. B. für den Diebstahl von Informationen oder Denial-of-Service-Angriffe[53].

Bei einem Drive-by-Download-Angriff sind in der Regel mehrere Schritte beteiligt, wie in Abbildung 2.6 dargestellt. Der bösartige Inhalt, der in die kompromittierte Webseite eingebettet ist und zunächst vom Browser gerendert wird, nutzt normalerweise nicht direkt die Browser-Schwachstelle aus. Stattdessen leitet er den Browser auf andere Umleitungsserver um, die entweder externe Webseiteninhalte bereitstellen oder auf weitere Server umleiten. Nach dem Besuch von einem oder mehreren möglicherweise gutartigen Umleitungsservern wird der Browser schließlich auf einen bösartigen Umleitungsserver umgeleitet, der versucht, den Browser auszunutzen und Malware herunterzuladen. Der bösartige Umleitungsserver kann verwendet werden, um die Angriffe zu verwalten und den zu verwendenden Exploit-Server zu bestimmen, welcher den am besten passenden Satz von Exploits hat (z.B. IE-Exploits für IE-Browser). Ein Satz verschiedener Drive-by-Downloads kann von denselben Angreifern für einen bestimmten Zweck verwaltet werden (z.B. die Verteilung derselben Malware-Binärdatei für ein Botnet) und bildet ein Malware-Verteilungsnetzwerk (MDN)[53]. Daher wird empfohlen, das Betriebssystem und jede verwendete Software regelmäßig zu aktualisieren, um bekannte Sicherheitslücken zu schließen.

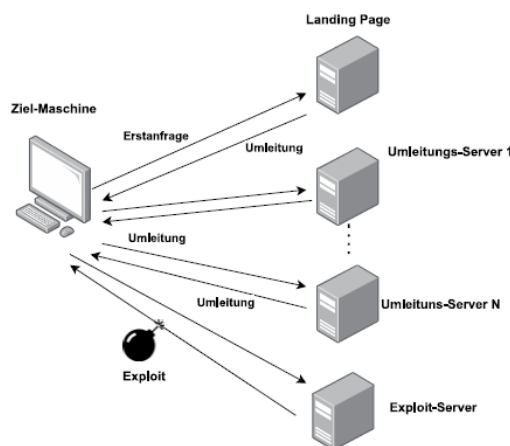


Abb. 2.6: Drive-by-Download

### Exploit-Kits

Ein Exploit-Kit ist ein Software-Tool, das auf dem Schwarzmarkt gehandelt wird, um Drive-by-Downloads auszuführen. Aus der Implementierungssicht handelt es sich dabei um nichts weiter als eine serverseitige HTTP-Applikation[26], die basierend auf den HTTP-Kopfzeilen eine Seite mit den passenden Schwachstellen zurückliefert. Seine Hauptaufgabe besteht darin, unbemerkt Malware auf den Systemen eines Opfers herunterzuladen und auszuführen. Fehler in den verwendeten Programmierschnittstellen oder auf Speicherkorruption basierende Schwachstellen ermöglichen es

einem Exploit, eine Reihe von Anweisungen in den Opferprozess einzuschleusen. Der Shellcode wiederum lädt eine ausführbare Malware-Datei auf die Festplatte des Opfers herunter und führt sie aus. Die auf dem Zielsystem installierte ausführbare Datei ist völlig unabhängig von dem Exploit-Kit. Obwohl der Großteil aller Drive-by-Downloads mithilfe von Exploit-Kits ausgeführt wird, ist nicht jedes Exploit-Kit Teil eines Drive-by-Downloads und umgekehrt. Ähnlich wie bei Drive-by-Downloads ist die Empfehlung, das System regelmäßig zu aktualisieren, um bekannte Sicherheitslücken zu schließen.

### Infizierte USB-Sticks und andere externe Speichergeräte

Obwohl wir zuvor betont hatten, dass physische Speichermedien heutzutage eine untergeordnete Rolle bei der Verbreitung von Malware spielen, da das Internet den Zugriff auf nahezu jedes System weltweit ermöglicht, zeigt die neueste Umfrage von Apricorn, dass noch immer ein Drittel aller Dateneinbrüche in Unternehmen durch infizierte Speichergeräte verursacht wird [2]. In der Umfrage gaben zwar 97% der Befragten an, dass nur verschlüsselte USB-Sticks in Unternehmen verwendet werden sollten, jedoch nutzten nur 37% tatsächlich einen solchen USB-Stick. Ein Angriff über einen USB-Stick kann auf verschiedene Weisen durchgeführt werden. Zunächst präpariert der Angreifer den USB-Stick, indem er Malware darauf einspielt. Eine mögliche Methode ist der sogenannte “Drop Attack”, bei dem der Angreifer einen solchen USB-Stick absichtlich in der Nähe des Ziels liegen lässt, in der Hoffnung, dass jemand aus Neugier den USB-Stick anschließt und somit die Malware aktiviert. Eine andere Möglichkeit besteht darin, den USB-Stick so zu modifizieren, dass er bei Anschluss nicht als USB-Stick, sondern als anderes Peripheriegerät erkannt wird, um andere Schwachstellen auszunutzen. Eine weitere beliebte Methode unter Angreifern ist die Verwendung eines sogenannten “Rubber Ducky”. Ein Rubber Ducky ähnelt äußerlich einem USB-Stick, enthält jedoch eine Hardwarekomponente, die eine Tastatur simuliert und bei Anschluss an ein Gerät vordefinierte Tastatureingaben ausführen und Code starten kann. Ähnlich wie bei Phishing-E-Mails ist es von großer Bedeutung, dass Angestellte regelmäßig geschult werden, um sicherzustellen, dass sie ausschließlich verschlüsselte USB-Sticks verwenden und keine gefundenen USB-Sticks an ihre Geräte anschließen.

### Innere Funktionsweise und Selbstverbreitung von Malware

Wir haben nun die gängigsten Verbreitungswege von Malware diskutiert. Allerdings verfügt die Malware selbst auch über Fähigkeiten, sich zu verbreiten und weitere Systeme zu infizieren. Wie in Abbildung 2.7 dargestellt, kann man Malware zunächst in zwei Gruppen aufteilen: solche, die sich selbst reproduzieren, und solche, die dies nicht tun.

Die selbstreproduzierende Malware kann weiter unterteilt werden in solche, die dies automatisch tun, ohne Interaktion mit dem Nutzer, und solche, die eine Nutzerinteraktion benötigen. Ein Virus beispielsweise kann sich nach Aktivierung durch einen Nutzer in einem Netzwerk und System selbst ausbreiten und weitere Geräte infizieren. Würmer hingegen haben die Fähigkeit, dies vollautomatisch und ohne jegliche Benutzerinteraktion zu tun.

Die nicht selbst reproduzierende Malware wird wiederum unterteilt in solche, die eine Benutzerinteraktion benötigen, und solche, bei denen der Angreifer selbst aktiv werden muss. Bei der Malware, die eine Benutzerinteraktion benötigt, gibt es solche, die offen als Malware erkennbar

ist, wie Adware, und solche, die versuchen, sich dem Nutzer gegenüber als legitime Software auszugeben, wie Trojaner. Bei der Malware, bei der der Angreifer selbst aktiv werden muss, haben wir zum Beispiel Backdoor-Software oder Rootkits.

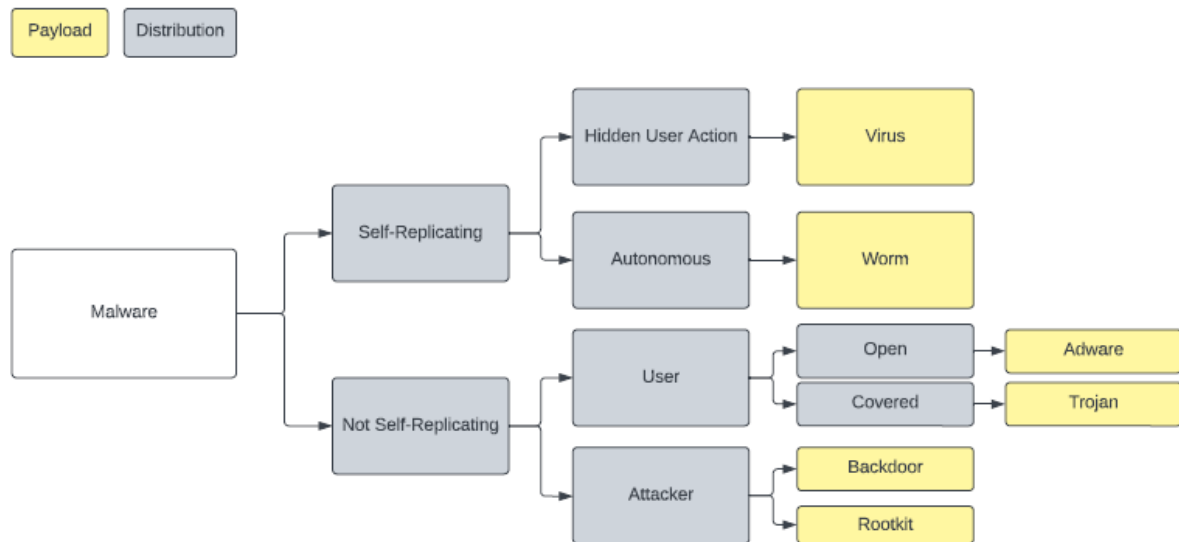


Abb. 2.7: Malware Verbreitung

### 2.4.3 Malware Obfuscation

Obfuscation (auf Deutsch “Verschleierung”) ist eine Technik, die dazu dient, die Lesbarkeit und Verständlichkeit von Programmcode zu erschweren. Sie verwandelt ein Programm in eine neue Version, die die gleiche Funktion hat, aber schwerer zu analysieren ist[7]. Ursprünglich wurde diese Technologie entwickelt, um das geistige Eigentum von Softwareentwicklern zu schützen und die unerlaubte Kopie oder den Diebstahl ihrer Arbeit zu verhindern. Doch schnell haben Malware-Entwickler diese Methode für ihre Zwecke adaptiert, um die Erkennung ihrer schädlichen Programme zu erschweren[52]. Um Malware effektiv analysieren zu können, ist es daher notwendig, sich mit den gängigsten Verschleierungstechniken vertraut zu machen. In Abbildung 2.8 präsentieren wir einen Ausschnitt aus einem Code, anhand dessen wir verschiedene Verschleierungstechniken demonstrieren werden.

00401005	8BF0	MOV ESI,EAX
00401007	3E:8A00	MOV AL,BYTE PTR DS:[EAX]
0040100A	84C0	TEST AL,AL
0040100C	74 46	JE SHORT Test.00401054
0040100E	53	PUSH EBX
0040100F	3E:8F05 74F940	POP DWORD PTR DS:[40F974]
00401016	D3DB	RCR EBX,CL
00401018	0FCB	BSWAP EBX
0040101A	68 56104000	PUSH Test.00401056
0040101F	5B	POP EBX
00401020	3E:8903	MOV DWORD PTR DS:[EBX],EAX
00401023	43	INC EBX
00401024	0FBDC2	BSR EAX,EDX
00401027	A9 46A978DC	TEST EAX,DC78F946
0040102C	8BC2	MOV EAX,EDX
0040102E	52	PUSH EDX
0040102F	B6 86	MOV DH,86
00401031	B3 27	MOV BL,27
00401033	B8 7CFAA17F	MOV EAX,7FA1FF7C
00401038	EB 01	JMP SHORT Test.0040103B
0040103A	90	NOP
0040103B	0FBCC2	BSF EAX,EDX
0040103E	3E:C705 FC8841	MOV DWORD PTR DS:[4188FC],0
00401041	2D 210DE8B9	SUB EAX,B9E80C21
00401049	69DA E577D49D	IMUL EBX,EDX,90D477E5

Abb. 2.8: Sample Code [52]

### Dead-Code Insertion

Die Technik der “Dead-Code Insertion” ist eine verhältnismäßig einfache Methode, die sich dadurch auszeichnet, dass sie redundante Befehle zu einem Programm hinzufügt. Diese zusätzlichen Befehle haben allerdings keinen Einfluss auf das Verhalten des Programms [52]. Abbildung 2.9 zeigt ein Beispiel für Dead-Code Insertion, bei dem die in Rot markierten Zeilen zusätzlich eingefügt wurden, jedoch keinen Einfluss auf die Funktionsweise des Programms haben.

Allerdings sind solche Änderungen für Virens Scanner leicht zu erkennen, da diese oft den Code analysieren, nachdem alle redundanten Codezeilen entfernt wurden. Dadurch kann die Malware anhand von Signaturvergleichen identifiziert werden.

Als Reaktion auf die Fähigkeit von Antivirenprogrammen, solche Techniken zu identifizieren, wurden umfangreichere Codeabschnitte hinzugefügt, wie in Abbildung 2.10 dargestellt. Dies erschwert die Erkennung, da der Virens Scanner nicht mehr eindeutig feststellen kann, ob es sich um redundanten Code handelt oder nicht.

00401005	8BF0	MOV ESI,EAX	00401005	8BF0	MOV ESI,EAX
00401007	3E:8A00	MOV AL,BYTE PTR DS:[EAX]	00401007	3E:8A00	MOV AL,BYTE PTR DS:[EAX]
0040100A	84C0	TEST AL,AL	0040100A	84C0	TEST AL,AL
0040100C	74 49	JE SHORT Test.00401057	0040100C	74 40	JE SHORT Test.0040105B
0040100E	53	PUSH EBX	0040100E	53	PUSH EBX
0040100F	3E:8F05 74F940	POP DWORD PTR DS:[40F974]	0040100F	3E:8F05 74F940	POP DWORD PTR DS:[40F974]
00401016	90	NOP	00401016	03DB	RCR EBX,CL
00401017	03DB	RCR EBX,CL	00401018	0FCB	BSWAP EBX
00401019	0FCB	BSWAP EBX	0040101A	68 5D104000	PUSH Test.0040105D
0040101B	68 59104000	PUSH Test.00401059	0040101F	5B	POP EBX
00401020	5B	POP EBX	00401020	3E:8903	MOV DWORD PTR DS:[EBX],EAX
00401021	3E:8903	MOV DWORD PTR DS:[EBX],EAX	00401023	43	INC EBX
00401024	90	NOP	00401024	0FBDC2	BSR EAX,EDX
00401025	43	INC EBX	00401027	A9 46A978DC	TEST EAX,DC78A946
00401026	0FBDC2	BSR EAX,EDX	0040102C	8BC2	MOV EAX,EDX
00401029	A9 46A978DC	TEST EAX,DC78A946	0040102E	90	NOP
0040102E	8BC2	MOV EAX,EDX	0040102F	90	NOP
00401030	52	PUSH EDX	00401030	42	INC EDX
00401031	90	NOP	00401031	52	PUSH EDX
00401032	B6 86	MOV DH,86	00401032	FE0C24	DEC BYTE PTR SS:[ESP]
00401034	B3 27	MOV BL,27	00401035	40	DEC EDX
00401036	B8 7CFAA17F	MOV EAX,7FA1FA7C	00401036	B6 86	MOV DH,86
00401038	EB 01	JMP SHORT Test.0040103E	00401038	B3 27	MOV BL,27
0040103D	90	NOP	0040103A	B8 7CFAA17F	MOV EAX,7FA1FA7C
0040103E	0FBCC2	BSF EAX,EDX	0040103F	EB 01	JMP SHORT Test.00401042
00401041	3E:C705 FC8841	MOV DWORD PTR DS:[4188FC],0	00401041	90	NOP
0040104C	2D 210DE8B9	SUB EAX,B9E80C21	00401042	0FBCC2	BSF EAX,EDX
00401051	69DA E577D49D	IMUL EBX,EDX,90D477E5	00401045	3E:C705 FC8841	MOV DWORD PTR DS:[4188FC],0
			00401048	2D 210DE8B9	SUB EAX,B9E80C21
			00401055	69DA E577D49D	IMUL EBX,EDX,90D477E5

Abb. 2.9: Dead-Code Insertion [52]

Abb. 2.10: Dead-Code Insertion Sequence [52]

### Register Reassignment

Register Reassignment ist, ähnlich wie Dead-Code Insertion, eine relativ einfache Methode der Malware-Verschleierung. Hierbei werden lediglich die Register von einer Mutation zur nächsten ausgetauscht [52]. In Abbildung 2.11 kann man beobachten, wie in der ersten Zeile das Register EAX durch EBX ersetzt wurde. Auch hier wird durch subtile Änderungen versucht, signaturbasierten Erkennungsmethoden zu entgehen

00401005	8BF3	MOV ESI,EBX
00401007	3E:8A1B	MOV BL,BYTE PTR DS:[EBX]
0040100A	84DB	TEST BL,BL
0040100C	74 48	JE SHORT Test.00401056
0040100E	52	PUSH EDX
0040100F	3E:8F05 74F940	POP DWORD PTR DS:[40F974]
00401016	D3DA	RCR EDX,CL
00401018	0FCA	BSWAP EDX
0040101A	68 58104000	PUSH Test.00401058
0040101F	5A	POP EDX
00401020	3E:891A	MOV DWORD PTR DS:[EDX],EBX
00401023	42	INC EDX
00401024	0FBDD8	BSR EBX,EAX
00401027	F7C3 46A978DC	TEST EBX,DC78A946
0040102D	8BD8	MOV EBX,EAX
0040102F	50	PUSH EAX
00401030	B4 86	MOV AH,86
00401032	B2 27	MOV DL,27
00401034	BB 7CFAA17F	MOV EBX,7FA1FA7C
00401039	EB 01	JMP SHORT Test.0040103C
0040103B	90	NOP
0040103C	0FBDD8	BSF EBX,EAX
0040103F	3E:C705 FC8841	MOV DWORD PTR DS:[4188FC],0
00401040	81FB 210DF8B9	SUB EBX,B9E80021
00401050	6900 E577D49D	IMUL EDX,EAX,9DD477E5

Abb. 2.11: Register Reassignment [52]

### Subroutine Reordering

Die Technik der Subroutine Reordering ermöglicht eine Verschleierung des ursprünglichen Codes durch zufällige Änderungen in der Anordnung seiner Unterprogramme[51]. Mit dieser Methode können  $n!$  verschiedene Varianten erzeugt werden, wobei  $n$  die Anzahl der Unterprogramme darstellt. Zum Beispiel hatte die Malware Win32/Ghost zehn Unterprogramme, was zu  $10! = 3.628.800$  unterschiedlichen Kombinationsmöglichkeiten führt[51].

### Instruction Substitution

Instruction Substitution ist eine Technik, bei der der Originalcode durch äquivalente Befehle ersetzt wird. Dabei werden die ursprünglichen Befehle durch solche ersetzt, die dieselbe Funktion erfüllen. So kann beispielsweise der Befehl XOR durch SUB ersetzt werden und MOV durch eine Kombination von PUSH und POP, wie in Abbildung 2.12 dargestellt[52].



00401005	8BF0	MOV ESI, EAX
00401007	3E:8000	MOV AL, BYTE PTR DS:[EAX]
0040100A	0AC0	OR AL, AL
0040100C	74 46	JE SHORT Test.00401054
0040100E	53	PUSH EBX
0040100F	3E:8F05 74F940	POP DWORD PTR DS:[40F974]
00401016	030B	RCR EBX, CL
00401018	0FCB	BSWAP EBX
0040101A	68 56104000	PUSH Test.00401056
0040101F	5B	POP EBX
00401020	3E:8903	MOV DWORD PTR DS:[EBX], EAX
00401023	43	INC EBX
00401024	0FB0C2	BSR EAX, EDX
00401027	00 46A9780C	OR EAX, DC78A946
0040102C	8BC2	MOV EAX, EDX
0040102E	52	PUSH EDX
0040102F	B6 86	MOV DH, 86
00401031	B3 27	MOV BL, 27
00401033	B8 7CF8A17F	MOV EAX, 7FA1FA7C
00401038	EB 01	JMP SHORT Test.0040103B
0040103A	90	NOP
0040103B	0FBCC2	BSF EAX, EDX
0040103E	3E:C705 FC8841	MOV DWORD PTR DS:[4188FC], 0
00401049	20 210DE8B9	SUB EAX, B9E88021
0040104E	690A E577D49D	IMUL EBX, EDX, 90D477E5

Abb. 2.12: Instruction Substitution [52]

## Code Transposition

Die Technik der Code Transposition ändert die Reihenfolge der Codeausführung, ohne das Verhalten der Anwendung zu verändern[52]. Dies kann durch zwei verschiedene Methoden erreicht werden:

Die erste Methode, dargestellt in Abbildung 2.13, mischt die Anweisungen zufällig und stellt dann die ursprüngliche Ausführungsreihenfolge durch das Einfügen unbedingter Verzweigungen oder Sprünge wieder her. Diese Methode kann jedoch einfach rückverfolgt werden, um die ursprüngliche Reihenfolge zu ermitteln[52].

Die zweite Methode, ist fortschrittlicher und ändert bei jeder Mutation die Reihenfolge der unabhängigen Codeabschnitte, die keinen Einfluss aufeinander haben. Diese Methode ist zwar komplexer zu implementieren, bietet aber den Vorteil, dass sie schwerer zu erkennen ist. Ein Beispiel für diese Art der Mutation ist in Abbildung 2.14 dargestellt[52].



Abb. 2.13: Code Transposition based on Unconditional Branches [52]

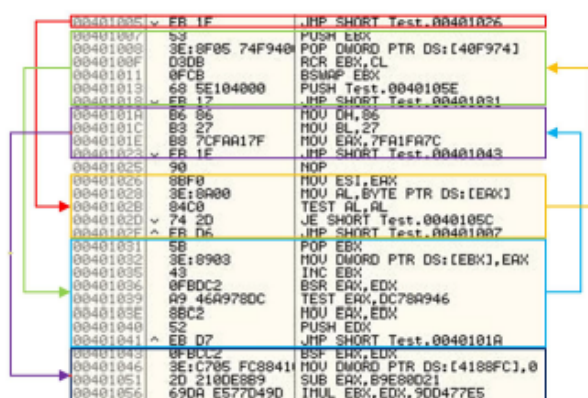


Abb. 2.14: Code Transposition based on Independent Instructions [52]

## Code Integration

Die Technik der Code-Integration gehört zu den fortschrittlichsten Methoden der Malware-Verschleierung. Ursprünglich wurde sie von der Win95/Zmist-Malware verwendet[25], die sich an den Zielcode anpasst und in ihn integriert. Um diese Technik anzuwenden, dekompiert die Malware zunächst das Zielprogramm in handhabbare Objekte, fügt sich nahtlos zwischen diese ein und setzt den integrierten Code in einer neuen Mutation wieder zusammen. Aufgrund ihrer Komplexität kann die Code-Integration die Erkennung und Wiederherstellung erheblich erschweren und stellt somit eine besonders raffinierte Verschleierungstechnik dar[52].

## 2.5 Malware-Analyse-Methoden

Diese Arbeit konzentriert sich auf die Erstellung einer vollautomatischen Umgebung zur Malware-Analyse. Eine effektive Gestaltung eines solchen Systems erfordert ein tiefgreifendes Verständnis der gegenwärtigen Analyseverfahren. Die Malware-Analyse ist der erste und entscheidende Schritt zur Malware-Erkennung[46].

Eine tiefgehende Kenntnis der Funktionsweise von Malware und der Absichten, die deren Entwicklung zugrunde liegen, ist unerlässlich. Nur so können wir effektive Abwehrmechanismen in Malware-Detektionssystemen implementieren[46].

Die Malware-Analyse hat darüber hinaus die Aufgabe, nicht nur den Code zu untersuchen, um Experten ein Verständnis für die Malware zu verschaffen, sondern auch “noch nie dagewesene Malware” einer Signatur zuzuordnen. Dies ermöglicht die Erkennung solcher Malware in Zukunft[31].

Nach einer ausführlichen Betrachtung der verschiedenen Arten von Malware und ihren charakteristischen Eigenschaften, konzentrieren wir uns nun auf die Analysetechniken. Diese lassen sich nach Zeitaufwand und spezifischer Analysetechnik in drei Hauptkategorien einteilen.

### 2.5.1 Statische Analyse

Die statische Analyse charakterisiert eine Sammlung von Methoden, bei denen der Code zur Untersuchung nicht ausgeführt wird [15] [46]. Diese Methoden umfassen eine Reihe von Techniken, darunter String-Signaturen, N-Gramme von Byte-Sequenzen, statische Bibliotheksaufrufe, Kontrollflussdiagramme und Opcode-Häufigkeitsverteilungen [15].

Bevor die statische Analyse durchgeführt werden kann, muss die Malware zunächst entpackt und entschlüsselt werden. Ein Disassembler oder Debugger kommt hierbei zum Einsatz, um den kompilierten Code in seinen Originalzustand zurückzuführen und somit den Zugriff auf den eigentlichen Code zu ermöglichen.

In der Anfangszeit der Malware-Erkennung waren statische Methoden oftmals ausreichend und ermöglichten die schnelle Identifizierung einer großen Anzahl an Malware. Sie sind insbesondere effektiv bei der Erkennung bereits bekannter Malware. Allerdings ist es wichtig zu beachten, dass Malware, wie in den Abschnitten 2.4.3 und 2.4.1 diskutiert, verschiedene Techniken einsetzen kann, um der Analyse zu entgehen. Trotz ihrer Nützlichkeit ist die statische Analyse anfällig

für diverse Umgehungsstrategien, insbesondere wenn Malware-Autoren Verschleiertechniken nutzen. Da nahezu jede moderne Malware solche Techniken verwendet, ist die statische Analyse allein oftmals nicht ausreichend, was zur Entwicklung dynamischer Analysemethoden geführt hat [15].

### 2.5.2 Dynamische Analyse

Die dynamische Analyse zeichnet sich dadurch aus, dass sie den Inhalt der Software nicht direkt analysiert, sondern das Verhalten nach deren Ausführung beobachtet. Daher wird sie auch als Verhaltensanalyse (engl. "behavioral analysis") bezeichnet [46].

In der dynamischen Analyse wird die Malware in einer geschützten Umgebung, auch als Sandbox bezeichnet, ausgeführt und beobachtet. Es gibt viele verschiedene Aspekte, die man während dieser Beobachtung aufzeichnen kann, um die Malware zu klassifizieren. Zu den verschiedenen Techniken, die bei der Durchführung einer dynamischen Analyse eingesetzt werden können, gehören die Überwachung von Funktionsaufrufen, die Parameteranalyse von Funktionen, das Verfolgen des Informationsflusses, das Erfassen von Befehlsspuren und Autostart-Erweiterungspunkten, das Überprüfen von aufgerufenen API-Aufrufen sowie das Beobachten der Verwendung von verschiedenen Systemaufrufen [46] [15].

Die dynamische Analyse ist im Vergleich zur statischen Analyse effektiver und erfordert keine Disassemblierung der ausführbaren Datei. Sie offenbart das natürliche Verhalten der Malware, das gegenüber der statischen Analyse widerstandsfähiger ist. Allerdings ist diese Methode in Bezug auf Aufwand und Ressourcen deutlich umfangreicher, da ein System erstellt werden muss, in dem die Malware getestet werden kann. Dies ist nicht nur sehr zeitaufwendig, sondern auch kosten- und ressourcenintensiv [15].

Ein weiteres Problem bei der dynamischen Analyse besteht darin, dass sich die Malware in der virtuellen Umgebung anders verhalten kann als auf einem echten System. Dies kann dazu führen, dass die Malware nur ein künstliches Verhalten vorspielt, um einer Analyse zu entgehen. Zusätzlich können verschiedene Malware-Typen nur unter bestimmten Bedingungen ausgeführt werden, welche in einer virtuellen Umgebung nicht immer nachgeahmt werden können [15].

Trotz dieser Herausforderungen ist die dynamische Analyse deutlich effizienter und effektiver in der Malware-Analyse als rein statische Methoden. Obwohl sie mehr Zeit benötigt, um die Malware zu analysieren, sind die Ergebnisse aussagekräftiger.

## 2.6 Windows System Calls

Ein Systemaufruf stellt eine Methode dar, durch die Programme mit dem Betriebssystem interagieren können. Wenn ein Programm eine Aktion ausführen muss, die spezielle Berechtigungen erfordert (wie beispielsweise das Lesen oder Schreiben in einer Datei oder einem Netzwerk, oder der Zugriff auf Systemressourcen), sendet es einen Systemaufruf an das Betriebssystem, das dann die Anfrage bearbeitet. Im Kern bieten Systemaufrufe eine Schnittstelle zwischen einem Prozess und dem Betriebssystem [22].



Die Anwendungsprogrammierschnittstelle (API) fungiert als Brücke zwischen den Funktionen des Betriebssystems und den Benutzerprogrammen. Sie ermöglicht es Programmen auf Benutzerebene, Dienste des Betriebssystems anzufordern. Zugriff auf das Kernel-System kann ausschließlich über Systemaufrufe erlangt werden. Alle Programme, die Ressourcen nutzen, benötigen Systemaufrufe [22].

Im Kontext von Windows werden Systemaufrufe hauptsächlich zur Interaktion mit der Hardware, dem Windows-Kernel oder den Windows-Betriebssystemdiensten genutzt. Sie können für Aufgaben wie das Erstellen eines neuen Prozesses, das Öffnen einer Datei, das Schreiben in eine Datei, das Senden von Netzwerkdaten oder das Beenden eines Prozesses eingesetzt werden. Die Anzahl der auf einem System existierenden Systemaufrufe variiert je nach Betriebssystem. Bei Windows wird mit etwa 2000 unterschiedlichen Systemaufrufen gerechnet, wobei diese Zahl je nach Windows-Version und Update variieren kann [23].

Ein wichtiger Aspekt, der bei Systemaufrufen berücksichtigt werden muss, ist ihr potentielles Sicherheitsrisiko. Da sie Programmen die Ausführung von Aktionen ermöglichen, die spezielle Berechtigungen erfordern, können sie von bösartiger Software, wie Malware, zur Durchführung von Angriffen auf das System ausgenutzt werden [14]. Deshalb ist es von entscheidender Bedeutung, Systemaufrufe sorgfältig zu verwalten und zu überwachen. Ein großer Teil der Forschung im Bereich der Computersicherheit konzentriert sich daher auf Systemaufrufe.

Eine Systemaufrufspur (engl. "System call trace") ist die geordnete Sequenz von Systemaufrufen, die ein Prozess während seiner Ausführung tätigt. Diese Spur kann durch verschiedene Werkzeuge, wie zum Beispiel das Intel Pin Tool, aufgezeichnet werden. Dadurch können Angriffe, wie etwa Malware, die von Benutzerrechten zu Administratorenrechten gelangt, aufgedeckt werden [14]. Typischerweise unterscheidet sich die Systemaufrufspur eines Programms, welches Schwachstellen ausnutzt, erheblich von derjenigen unter normalen Bedingungen. Dies liegt daran, dass Malware eine breite Palette an verschiedenen Systemaufrufen ausführt, die deutlich erkennbar sind. Zum Beispiel kann Malware vorgeben, eine einfache Anwendung zu sein, die grundlegende Funktionen ausführt, während sie im Hintergrund versucht, einen Pufferüberlauf oder andere Schwachstellen auszunutzen. Diese Muster sind erkennbar und sehr wertvoll für die Malware-Analyse.

## 2.7 Automatisierung in der Malware-Erkennung und Analyse

Die Automatisierung der Malware-Erkennung und -Analyse stellt einen bedeutenden Aspekt bei der Bekämpfung dieser Bedrohung dar. Wie wir bereits in der Einleitung dieser Arbeit festgestellt haben, werden täglich nahezu 400.000 Malware-Instanzen erfasst [24]. Obwohl wir in vorherigen Abschnitten gelernt haben, dass es sich oft um dieselbe Malware in verschiedenen Formen durch Mutation handelt, müssen unsere Virens Scanner diese neuen Formen idealerweise sofort identifizieren und entsprechende Maßnahmen ergreifen. Die Automatisierung bietet dabei folgende Vorteile:

- **Höhere Geschwindigkeit und Effizienz:** Die manuelle Malware-Analyse kann zeitaufwendig sein und benötigt erhebliche Ressourcen. Durch die Automatisierung des Prozesses können Sicherheitsteams eine große Menge an Malware-Samples schnell und effizient analysieren, wodurch sie Bedrohungen schneller erkennen und darauf reagieren können [45].

- **Konsistenz und Genauigkeit:** Die Automatisierung der Malware-Analyse kann menschliche Fehler minimieren und konsistente sowie genaue Ergebnisse liefern. Automatisierte Tools können bei jeder Probe denselben Prozess durchlaufen, was das Risiko von Fehlern oder Auslassungen verringert [45].
- **Bessere Ressourcenzuweisung:** Durch die Automatisierung der Malware-Analyse können sich Sicherheitsexperten auf wichtigere Aufgaben konzentrieren, wie zum Beispiel die Entwicklung neuer Gegenmaßnahmen und die Verbesserung der allgemeinen Sicherheitslage des Unternehmens [45].
- **Kosteneinsparungen:** Die manuelle Malware-Analyse kann teuer sein und erfordert viel Zeit und Ressourcen. Die Automatisierung des Prozesses kann Unternehmen dabei unterstützen, Kosten zu senken und die Gesamteffizienz ihrer Sicherheitsabläufe zu verbessern [45].

## 3 Analyse und Design-Vorbereitung

### 3.1 Problembeschreibung

Der Aufbau einer lokalen Testumgebung für die Malware-Analyse kann ein aufwändiges Unterfangen sein, das viel Zeit und Geld in Anspruch nimmt. Es gibt viele Aspekte, die dabei beachtet werden müssen.

Zunächst muss man sich für eine geeignete Virtualisierungssoftware entscheiden. Nach der Auswahl einer passenden Software müssen passende ISO-Dateien gefunden werden, die das zu testende System darstellen. Dabei ist es wichtig, dass diese das tatsächliche System so genau wie möglich abbilden.

Die (VMs) müssen außerdem einem sogenannten “Hardening”-Prozess unterzogen werden. Dieser stellt sicher, dass die Malware die VMs nicht verlassen und das Host-System infizieren oder beschädigen kann.

Ein weiteres Problem stellt die Beschaffung von Goodware und Malware dar, die getestet werden kann. Darüber hinaus bleibt die Frage, wo die Daten der Analyse gespeichert werden und wie diese anschließend verarbeitet werden können.

Zudem muss geklärt werden, wie der gesamte Prozess automatisiert werden kann. Im derzeitigen Zustand des Unternehmens werden all diese Schritte manuell durchgeführt und jede einzelne Malware oder Goodware wird individuell getestet. Dieser manuelle Ansatz ist zeitaufwendig und ineffizient und verlangt eine Verbesserung durch Automatisierung.

## 3.2 Softwareauswahl und Designplanung

### 3.2.1 Anforderungen

Die Anforderungen an das System wurden in regelmäßigen Abständen von zwei Wochen gemeinsam mit meinem Betreuer festgelegt. Wir haben dabei nach dem Scrum-Modell gearbeitet und jeweils die Ziele definiert, die bis zum nächsten Scrum Review/Planning erreicht sein sollten. In diesem Abschnitt werden die wichtigsten Anforderungen, die es in das finale Produkt geschafft haben, zusammengefasst.

- Die erste Anforderung an das System war, dass es sowohl auf Windows als auch auf macOS lokal laufen kann. Dies war besonders wichtig, da verschiedene Forscher auf unterschiedlichen Plattformen arbeiten und wir ein universelles System anbieten wollten, das von jedem System aus aufgebaut werden kann. Letztendlich haben wir uns darauf geeinigt, dass das System auf Windows 10 und auf macOS Catalina und neuer laufen soll.
- Eine weitere wichtige Anforderung war, dass das System ausschließlich Open-Source-Software verwendet, sodass keine Lizenzen beantragt werden müssen, um die Testumgebung aufbauen zu können. Dies hat die Auswahl an Software stark eingeschränkt, gleichzeitig aber auch vereinfacht. Es bietet mehrere Vorteile, da man schnell und ohne Kosten die passende Software herunterladen und das System aufbauen kann. Da wir planen, die Ergebnisse am Ende zu veröffentlichen, bieten wir auch anderen Forschern die Möglichkeit, mit wenig Aufwand das System lokal aufzubauen.
- Abgesehen vom Open-Source-Kriterium war die einzige Softwareeinschränkung, dass wir zur Analyse der Malware das Intel Pin Tool verwenden mussten. Da es bereits vorher genutzt und soweit konfiguriert wurde, dass es alle benötigten Funktionen für unsere Analyse bereitstellt, haben wir uns entschieden, dies nicht weiter zu ändern.
- Die Software sollte in der Lage sein, automatisch zu laufen. Dadurch wäre nach der initialen Einrichtung keine manuelle Intervention mehr erforderlich, was den Arbeitsaufwand für die Anwenderinnen und Anwender erheblich reduziert und die Effizienz des Systems erhöht.
- Wie bereits erwähnt, sollte die Malware mit dem Intel Pin Tool getestet werden, wobei als Ergebnis eine Comma-Separated Values (CSV)-Datei erstellt wird. Das Dateiformat des Outputs sollte jedoch flexibel änderbar sein. Wir haben uns für unsere Testzwecke auf CSV geeinigt, da der Machine-Learning-Algorithmus, der im Unternehmen verwendet wird, CSV-Dateien zum Einlesen benötigt.
- Das zu untersuchende System ist Windows 10 32-Bit. Dies hat mehrere Gründe. Zum einen ist Windows das weltweit am häufigsten genutzte Betriebssystem im Desktop-Bereich [43]. Da ältere Windows-Versionen keine Software- und Sicherheitsupdates mehr erhalten und nur noch Windows 10 und neuere Versionen von Microsoft im Desktop-Bereich unterstützt werden, war dies ein weiterer Grund für diese Entscheidung. Zudem wird die Mehrheit der Malware für Windows-Maschinen, insbesondere für 32-Bit-Systeme, geschrieben.
- Das zu untersuchende Betriebssystem sollte in einer Virtualisierungssoftware laufen. Dabei muss sichergestellt werden, dass die darin analysierte Malware nicht aus der virtuellen

Umgebung ausbricht und das Host-System infiziert oder beschädigt. Dies bedeutet, dass die VM entsprechend "gehardet" sein muss, um ein Ausbrechen der Malware zu verhindern.

- Die Windows-Maschine muss einem echten System so ähnlich wie möglich sein, da wir das Verhalten der Malware in einem realen Umfeld beobachten wollen. Dazu müssen verschiedene Maßnahmen ergriffen werden, die wir später im Detail beschreiben werden.
- Die durch die Analyse gewonnenen Daten sollen per FTP an einen FTP-Server auf einer weiteren virtuellen Maschine im selben Netzwerk wie die Windows-Maschine gesendet werden. Von dieser Maschine aus können die Daten, sobald genug davon gesammelt wurden, sicher an das Host-System übertragen werden.
- Es müssen sowohl Malware als auch Goodware gefunden werden, die im Intel Pin Tool analysiert werden können. Es ist wichtig, geeignete Goodware zu finden, damit wir die Windows-Systemaufrufe aufzeichnen und mit denen der Malware vergleichen können. Diese Vergleiche helfen uns, das Verhalten der Malware besser zu verstehen und effektive Gegenmaßnahmen zu entwickeln

### 3.2.2 Virtualisierungssoftware

Verschiedene Virtualisierungssoftware wurde im ersten Schritt evaluiert, wobei mehrere Kriterien beachtet wurden. Da die Analyse automatisiert durchgeführt werden sollte, musste die Möglichkeit bestehen, die virtuellen Maschinen über ein Skript zu steuern. In Anbetracht unseres Fokus auf Open-Source-Software zeichnete sich Oracle VirtualBox schnell als unsere bevorzugte Wahl ab. Ein Vorteil von VirtualBox ist die integrierte Command-line Interface (CLI) sowie eine API, die leicht ansprechbar ist. Abbildung 3.1 veranschaulicht die verschiedenen Möglichkeiten zur Interaktion mit der Virtualisierungs-Engine. In unserem Fall standen uns sowohl die CLI als auch die API zur Verfügung. Es ist jedoch zu beachten, dass wir mit einigen Herausforderungen konfrontiert waren, insbesondere in Bezug auf die API, da deren Dokumentation seit Jahren nicht aktualisiert wurde. Trotz dieser Schwierigkeiten war es uns gelungen, sowohl die CLI als auch die API für unsere Zwecke zu nutzen.

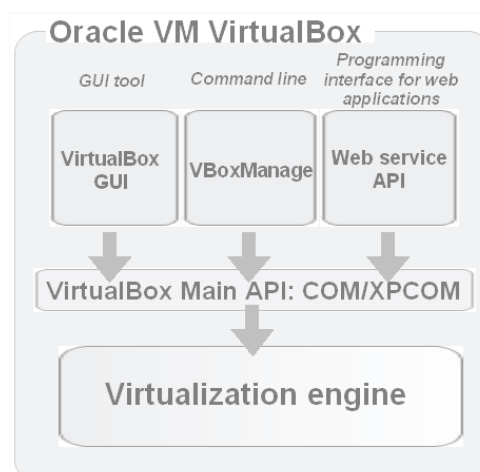


Abb. 3.1: Interfaces for interacting with Oracle VM VirtualBox [48]

Mit dem von Oracle bereitgestellten CLI-Tool, bekannt als VBoxManage, war es uns möglich, den Prozess der Erstellung virtueller Maschinen zu automatisieren. Allerdings erforderte dies eine vorbereitende Konfiguration der ISO-Dateien, um sie mit VBoxManage kompatibel zu machen. Der Befehl `VBoxManage unattended install` ermöglichte eine Installation ohne Benutzerinteraktion, wobei festgestellt wurde, dass nicht alle ISO-Dateien für diesen Befehl geeignet sind, und deshalb eine spezifische Konfiguration erforderlich ist.

Um die Eignung einer ISO-Datei für diesen Installationsprozess zu prüfen, kann der Befehl `VBoxManage unattended detect <-iso=install-iso>` verwendet werden. Die Ausgabe dieses Befehls gibt Aufschluss darüber, ob die jeweilige ISO-Datei für eine automatisierte Installation verwendet werden kann. Ein Beispiel für die Ausgabe dieses Befehls ist in Abbildung 3.2 dargestellt.

Eine detailliertere Beschreibung der Anpassungen, die für die ISO-Dateien vorgenommen wurden, findet sich im Abschnitt 3.2.3.

```
PS D:\VM_ISO> VBoxManage unattended detect --iso=Win10Unattended.iso
VBoxManage.exe: error: Code E_NOTIMPL (0x80004001) (extended info not available)
VBoxManage.exe: error: Context: "DetectIsoOS()" at line 2029 of file VBoxManageMisc.cpp
VBoxManage.exe: info: Detected 'D:\VM_ISO\Win10Unattended.iso' to be:
    OS TypeId      = Windows10
    OS Version     = 2004
    OS Flavor      =
    OS Languages   = en-us
    OS Hints       =
    Unattended installation supported = yes
```

Abb. 3.2: VBoxManage unattended detect

Ein Großteil der Funktionen, die die CLI bietet, sind allerdings nur zugänglich, wenn die Virtual-Box Guest Edition auf der virtuellen Maschine installiert ist. Dies stellt ein Problem dar, da diese Software auf unserer Windows-Maschine für die Malware-Analyse nicht vorhanden sein darf. Die Einzelheiten dazu werden im Abschnitt 3.2.6 ausführlicher erläutert. Daher war es notwendig, auch die [API](#) zu nutzen.

Die Nutzung der [API](#) brachte allerdings eine Reihe von ungelösten Problemen mit sich, weshalb wir uns entschlossen, die Python-Bibliothek `virtualbox 2.1.1` [12] zu verwenden. Diese Bibliothek verfügte über alle Funktionen, die auch die Standardbibliothek von VirtualBox anbot, und löste einige der Probleme und Bugs, die in der Standardbibliothek vorhanden waren.

### 3.2.3 ISO-Dateien

Überraschend stellten die ISO-Dateien ein weiteres Problem dar, das jedoch schnell behoben werden konnte. Da unser Testsystem auf Windows basieren sollte, war zunächst die Suche nach einer passenden Windows-ISO-Datei erforderlich. Viele der im Internet verfügbaren ISO-Dateien werden allerdings nicht von VBoxManage unterstützt und enthalten vorkonfigurierte Einstellungen, die sich als ungünstig herausstellen können. Daher entschieden wir uns dafür, unsere eigene ISO-Datei zu erstellen.

Das Windows MediaCreationTool ist ein hilfreiches Werkzeug, mit dem man eine ISO-Datei basierend auf dem eigenen System erstellen kann. Da mein Betriebssystem Windows 10 ist, habe ich eine ISO-Datei auf Basis meines Systems generiert. In diesem Tool lässt sich auch

die Architektur der ISO-Datei festlegen, in unserem Fall entschieden wir uns für eine 32-Bit-Architektur. Nach der Erstellung der ISO-Datei musste diese noch so angepasst werden, dass sie die automatische Installation durch VBoxManage unterstützt. Hierzu war es notwendig, sich mit dem Windows Assessment and Deployment Kit (ADK) vertraut zu machen. Das Windows ADK ermöglicht es, ISO-Dateien zu konfigurieren [28]. Im Anhang finden sich eine detaillierte Dokumentation zur Erstellung der ISO-Datei.

Für die zweite VM, auf der der FTP-Server betrieben werden sollte, entschieden wir uns für eine Ubuntu-Maschine. Die Entscheidung und Konfiguration für diese Maschine werde ich im Abschnitt 3.2.7 detaillierter beschreiben. Die ISO-Datei für die Ubuntu-Maschine musste ebenfalls entsprechend vorbereitet werden, was sich jedoch als einfacher herausstellte als bei der Windows-Maschine. Dies lag an der Möglichkeit, die ISO-Datei einfach über das Windows Subsystem for Linux (WSL) zu bearbeiten und so anzupassen, dass sie mit VBoxManage ausgeführt werden konnte.

### 3.2.4 Intel Pin Tool

Das Intel Pin-Tool ist ein dynamisches Binär-Instrumentierungs-Framework für die IA-32-, x86-64- und MIC-Befehlssatzarchitekturen. Es ermöglicht die Entwicklung dynamischer Programm-analysewerkzeuge. Einige der mit Pin erstellten Werkzeuge sind beispielsweise Intel® VTune™ Amplifier, Intel® Inspector, Intel® Advisor und der Intel® Software Development Emulator (Intel® SDE). Diese so genannten Pintools können zur Programmanalyse von Benutzeranwendungen unter Linux, Windows und macOS eingesetzt werden. Da es sich bei Pin um ein dynamisches Binär-Instrumentierungstool handelt, erfolgt die Instrumentierung an den kompilierten Binär-dateien zur Laufzeit. Daher ist es nicht notwendig, den Quellcode neu zu kompilieren, und es können auch Programme instrumentiert werden, die dynamischen Code erzeugen [20].

Pin bietet eine umfangreiche API, die die spezifischen Merkmale des zugrundeliegenden Befehls-satzes abstrahiert und die Weitergabe von Kontextinformationen, wie beispielsweise Registerinhalten, als Parameter an den injizierten Code ermöglicht. Pin speichert automatisch die durch den injizierten Code überschriebenen Register und stellt sie wieder her, wodurch die korrekte Funktionsweise der Anwendung gewährleistet bleibt. Eingeschränkter Zugriff auf Symbol- und Debug-Informationen ist ebenfalls möglich [20].

Obwohl das Pin-Tool ursprünglich als Instrument zur Analyse von Computerarchitekturen entwickelt wurde, hat seine flexible API in Verbindung mit einer aktiven Community, den sogenannten "Pinheads", zur Entwicklung einer Vielzahl von Werkzeugen für Sicherheit, Emulation und parallele Programmanalyse geführt [20].

In diesem Projekt wurde mir das Pin-Tool von meinem Betreuer zur Verfügung gestellt, und wir haben uns entschieden, es aufgrund seiner Eigenschaft als Open-Source-Software weiterhin zu nutzen. Dies war das einzige Mal in diesem Projekt, dass eine Software verwendet wurde, die keine weiteren Konfigurationsarbeiten meinerseits benötigte. Obwohl die Software bereits für den Einsatz vorbereitet war, lag die Verantwortung für die Integration und Implementierung des Tools in den bestehenden Automatisierungsprozess bei mir. Dies war ein komplexer Prozess, den ich vollständig selbst durchgeführt habe und der in Abschnitt 4 genauer beschrieben wird.



Intel liefert zusammen mit dem Download des Pin-Tools eine Vielzahl von vorbereiteten Versionen des Tools aus. Es besteht die Möglichkeit, eines der vielen vorab konfigurierten Pintools zu verwenden oder eine individuelle Konfiguration vorzunehmen, abhängig davon, welche Informationen man erfassen und beobachten möchte. In unserem Fall war unser Hauptinteresse die Auflistung und Beobachtung der aufgerufenen Systemaufrufe. Unser Ziel war es, die Spur dieser Systemaufrufe, wie in Abschnitt 2.6 dargestellt, zu erfassen und zu analysieren.

### 3.2.5 Automatisierungsprozess

Der Automatisierungsprozess konfrontiert uns mit zwei wesentlichen Herausforderungen: Zum einen die Automatisierung der Erstellung der Testumgebung, zum anderen die Automatisierung des Malware-Analyseprozesses mithilfe des Intel Pin Tools. Außerdem müssen die erstellten Skripte sowohl auf Windows als auch auf macOS funktionieren.

Die erste Herausforderung bestand darin, die Testumgebung automatisiert zu erstellen. Dies wurde mithilfe eines Skripts realisiert, das die VirtualBox-CLI nutzt. Unser Ziel war es, die Anzahl der benötigten Benutzerinteraktionen auf ein Minimum zu reduzieren. Dies ist von entscheidender Bedeutung, um die Einrichtung der virtuellen Maschinen so schnell und ohne weiteren Konfigurationsaufwand wie möglich durch die Ausführung eines Skripts abzuschließen. Zwar ist die Einrichtung auch über die VirtualBox-GUI möglich, dieser Weg wäre jedoch zeitaufwändiger, was durch den Einsatz des Skripts vermieden wird. Im Skript selber, können jedoch weitere Konfigurationen vorgenommen werden, um die Maschinen an die eigenen Anforderungen anzupassen.

Sobald die Maschinen installiert und betriebsbereit sind, müssen sie für die Malware-Analyse vorbereitet werden. Um Zeit zu sparen, wurden jeweils separate Skripte zur Einrichtung der Ubuntu- und der Windows-Maschine erstellt. Trotz dieser Automatisierung sind einige manuelle Schritte erforderlich, da nicht alle Aspekte durch Skripte automatisiert werden können. Dies wird im nächsten Kapitel 4 detailliert erläutert.

Die Automatisierung der Malware-Analyse erfordert sowohl die Nutzung der CLI als auch der API, die Implementierung erfolgte in einem Python-Skript. Bei der Erstellung war es von zentraler Bedeutung, das Skript modular zu gestalten, sodass das Intel Pin Tool mit minimalem Aufwand durch ein anderes Analyse-Tool der Wahl ersetzt werden kann.

Dieser Abschnitt der Arbeit war der zeitintensivste. Im Laufe der Zeit durchlief das Skript verschiedene Iterationen, bis es schließlich die von uns gestellten Anforderungen erfüllte. Eine der Herausforderungen bestand darin, die Windows-Maschine von außen steuern zu können, ohne das Verhalten der Malware zu beeinflussen. Hierfür wurden verschiedene Methoden ausprobiert. Die Tatsache, dass viele Funktionen der API und der CLI nicht mehr zur Verfügung stehen, wenn die Gast-Editionen, die zu Beginn auf der Windows-Maschine installiert sind, deinstalliert werden, erschwerte den Zugriff. Diese Probleme wurden nacheinander abgearbeitet, bis wir schließlich zur aktuellen, finalen Version gelangten, die im Kapitel 4 erläutert wird.

Eine weitere wichtige Anforderung bei der Automatisierung der Malware-Analyse ist die Rückkehr der Maschine in einen neutralen Zustand nach der Ausführung der Malware, ähnlich dem Zustand vor der Ausführung der Malware. Dies ist wichtig, da verschiedene Arten von Malware das System so beeinträchtigen können, dass die Ergebnisse sonst nur begrenzt verwertbar wären.



Um dieses Ziel zu erreichen, haben wir mit Snapshots gearbeitet. Nach jedem Durchlauf des Skripts muss die Maschine auf den Zustand des vorher erstellten Snapshots zurückgesetzt werden. Dies gewährleistet, dass die Ausgangsbedingungen für jede Analyse gleich sind und ermöglicht eine zuverlässige und reproduzierbare Untersuchung.

Ein weiteres Problem betrifft die Ausführung der Malware zusammen mit dem Intel Pin Tool. Das Pin Tool führt die Anwendungen aus und überwacht dabei die Systemaufrufe. Die Ausführung von Malware gestaltet sich jedoch als Herausforderung, da der Windows Defender sehr aggressiv ist und sich nur schwer deaktivieren lässt. Um die Malware dennoch ausführen zu können, mussten wir zusätzliche Maßnahmen ergreifen. Diese werden wir im Abschnitt 3.2.9 ausführlich erläutern.

Schließlich mussten wir eine Strategie entwickeln, um die Daten vom Windows-System sicher auf ein anderes System zu übertragen, ohne das Host-System dem Risiko einer Malware-Infektion auszusetzen. Unsere Lösung bestand darin, dass jedes Mal, wenn das Skript für die Malware-Analyse durchgeführt wird, der Output direkt an die Ubuntu-Maschine via FTP gesendet wird. Dies gewährleistet eine sichere Datenübertragung, während die Integrität der Host-Maschine gewahrt bleibt. Von dieser können dann die Daten weiter an das Host-System transferiert werden.

### 3.2.6 Isolierungsprozess und Hardening

Es ist zwingend notwendig, sicherzustellen, dass die VMs ausreichend isoliert sind, um eine potenzielle Malware-Infektion der Host-Maschine oder des FTP-Servers zu verhindern.

In VirtualBox lassen sich die Netzwerkeinstellungen der VMs nach Belieben konfigurieren. Die VMs wurden in ein eigenes internes Netzwerk versetzt, um eine mögliche Malware-Infektion der Host-Maschine zu verhindern. Abbildung 3.3 zeigt einen Überblick aus der Dokumentation, welche Netzwerktypen welche Kommunikationseigenschaften besitzen. Da die virtuellen Maschinen nach der Einrichtung keine Kommunikation mit dem Internet oder der Host-Maschine benötigen, stellt ein internes Netzwerk die sicherste Option dar. Einige empfehlen auch ein Bridged-Network, allerdings wären dafür zusätzliche Sicherheitsmaßnahmen erforderlich.

Mode	VM→Host	VM←Host	VM1↔VM2	VM→Net/LAN	VM←Net/LAN
Host-only	+	+	+	–	–
Internal	–	–	+	–	–
Bridged	+	+	+	+	+
NAT	+	<a href="#">Port forward</a>	–	+	<a href="#">Port forward</a>
NATservice	+	<a href="#">Port forward</a>	+	+	<a href="#">Port forward</a>

Abb. 3.3: Übersicht der Netzwerkmodi [49]

Nach dem vollständigen Aufsetzen der Windows-Maschine und ihrer Vorbereitung für die Malware-Analyse ist es notwendig, die VirtualBox Guest Additions zu deinstallieren. Zwar erleichtern sie den Betrieb der virtuellen Maschine und die Kommunikation zwischen Host und VM, jedoch prüfen einige Malware-Varianten nach dieser Datei. Sie führen sich dann nicht aus, da die Malware-Entwickler wissen, dass ihre Malware in virtuellen Maschinen untersucht wird. Darüber hinaus stellt dies ein Sicherheitsrisiko dar, da die Malware auf diese Weise leichter auf das Host-System

gelangen kann. Auch Funktionen wie “Drag and Drop” sowie “Shared Clipboard”, die nur durch die Guest Additions ermöglicht werden, sollten deaktiviert sein.

Es ist auch wichtig, eine Internetverbindung zu simulieren, was mit verschiedenen Open-Source-Softwarelösungen erreicht werden kann. Wie in den vorherigen Kapiteln beschrieben, versucht sich Malware oft der Analyse zu entziehen. Daher ist es von entscheidender Bedeutung, das System so nah wie möglich an einem realen System zu gestalten.

Darüber hinaus ist es empfehlenswert, mit Snapshots die verschiedenen Zustände der VM zu dokumentieren - sowohl vor dem Hardening- und Isolationsprozess als auch danach. Dies ermöglicht es, bei Problemen oder Änderungen diese ohne großen Aufwand rückgängig zu machen.

### 3.2.7 FTP-Server

Zur Datensicherung setzen wir einen FTP-Server ein. Die grundlegende Idee besteht darin, dass nach der Analyse von Malware oder Goodware durch das Pintool der resultierende Output verarbeitet und an den FTP-Server gesendet wird.

Für den FTP-Server habe ich mich für eine Ubuntu-Maschine entschieden. Diese Entscheidung beruht auf mehreren Gründen. Da wir auf der Windows-Maschine Malware für Windows 10 32Bit testen, ist es unwahrscheinlich, dass diese Malware die Linux-Maschine infiziert. Zudem ist die Konfiguration und Bearbeitung des FTP-Servers auf Linux deutlich einfacher. Darüber hinaus bietet Ubuntu eine Vielzahl von vorgefertigten Bibliotheken und Optionen, die bei Verwendung von Windows deutlich umständlicher wären und mehr Aufwand erfordern würden.

Als FTP-Server haben wir den Standard-Linux-FTP-Server vsftpd gewählt, der in vielen Linux-Distributionen weit verbreitet ist und alle notwendigen Funktionen bereitstellt. Die Konfiguration ist ebenfalls sehr einfach und kann in der Datei `vsftpd.conf` vorgenommen werden.

### 3.2.8 Aufbereitung von Goodware und Malware

Für die Überprüfung des Systems ist es unerlässlich, diverse Testdaten zu verwenden. Selbstverständlich hängt die Qualität der erzielten Ergebnisse stark von den eingesetzten Daten ab.

Im Kontext der Goodware kann prinzipiell jede legitime Software als Testobjekt genutzt werden. Es zeigen sich allerdings Unterschiede in der Qualität der Ergebnisse, weshalb es nicht ausreichend ist, willkürlich ausgewählte Software zu testen. Es sollte vielmehr darauf abgezielt werden, speziell solche Software einzusetzen, die eine Vielzahl von Systemaufrufen im Kernel-space anfordert und diese Informationen benötigt. Dies ist von besonderer Bedeutung, da Malware oftmals das Ziel verfolgt, in die Kernel-Ebene einzudringen. Ein Vergleich mit Goodware, die ebenfalls auf der Kernel-Ebene agiert, ist daher sinnvoll. Auch die Art und Menge der genutzten Systemaufrufe, ihre Intensität und die Reihenfolge ihrer Ausführung spielen eine entscheidende Rolle.

Bei der Auswahl der Goodware wurde daher darauf geachtet, insbesondere viele interne Microsoft-Programme zu testen. Als Auswahl wurden zum Beispiel die Sysinternals herangezogen, die nahezu das gesamte Spektrum der Systemaufrufe abdecken. Ein Vorteil der Sysinternals ist, dass sie von der Hardware- bis zur Software-Ebene fast alle Bereiche abdecken, die für Malware-Entwickler von Interesse sind und als Angriffsziel ihrer Software dienen könnten.

Die Suche nach geeigneter Malware erwies sich als eine größere Herausforderung. Zwar ist Malware weit verbreitet, jedoch ist es schwierig, eine umfangreiche Sammlung zu finden, die ein breites Spektrum verschiedener Malware-Typen abdeckt. Es gibt zwar viele Anbieter, die Malware für Forschungszwecke anbieten, diese sind aber häufig kostenpflichtig. Nach umfangreicher Suche stellten wir schließlich eine Anfrage bei [Virusshare.com](http://Virusshare.com), um Zugang zu deren Malware-Sammlungen zu erhalten. VirusShare bietet eine große Menge an Malware für Forschungszwecke an, allerdings musste dafür eine Anfrage gestellt werden. Daraufhin wurden verschiedene Archive mit Malware-Proben aus unterschiedlichen Zeiträumen heruntergeladen.

Nach dem Herunterladen musste die Malware mit zusätzlicher Software identifiziert werden, da die Dateien nach dem Entpacken aus dem Archiv keinen Dateityp aufwiesen. Das Pintool kann jedoch nur Anwendungen ausführen und muss diese daher testen können. Zur Analyse der Dateitypen wurde TrID verwendet. Da wir nur an Anwendungen interessiert waren, die auf Win32 laufen, wurden nur diese Dateien in Anwendungen umgewandelt, die dann für die Malware-Analyse verwendet wurden. Je moderner die Malware war, desto schwieriger gestaltete sich die Analyse, da fortschrittliche Malware eine Vielzahl von Verschleierungsfunktionen nutzt. Trotzdem stellte dies kein größeres Problem dar, da wir mehrere tausend Malware-Proben zum Testen zur Verfügung hatten.

### 3.2.9 Herausforderungen mit dem Windows Defender

Die frühen Phasen des Projekts, die sich auf das Testen von Goodware konzentrierten, gaben wenig Anlass zur Annahme, dass der Windows Defender zu einem signifikanten Problem werden könnte. Als wir jedoch zur Phase der Malware-Tests übergingen, stießen wir auf einige unerwartete Herausforderungen.

Die Windows-Maschine war in einem Zustand, der den Test von Malware erlaubte und dieser Zustand wurde durch einen Snapshot festgehalten. Der Windows Defender wurde vor der Erstellung dieses Snapshots manuell deaktiviert, um die Ausführung von Malware zu ermöglichen. Überraschenderweise stellten wir jedoch fest, dass der Windows Defender dazu neigt, sich nach einem Systemneustart oder nach einer gewissen Zeitspanne selbst wieder zu aktivieren. Obwohl die meisten Funktionen des Defenders deaktiviert blieben, wurde die Funktion "Echtzeit-schutz" regelmäßig wieder aktiviert. Dies führte entweder dazu, dass die Ausführung von Malware vollständig blockiert wurde oder dass die resultierenden Daten von geringer Bedeutung waren.

Eine intensivere Untersuchung ergab, dass es möglich ist, den Windows Defender direkt im Windows-Register für den spezifischen Benutzer zu deaktivieren. Dieser Ansatz führte zu einer längeren Periode der Deaktivierung des Defenders, bis der Dienst den Registereintrag selbständig löschte und sich wieder aktivierte.

Die abschließende Lösung für dieses Problem bestand darin, den Defender durch das Skript vor jeder Ausführung des Pintools zu deaktivieren. Dies ermöglichte dem Pintool eine ausreichende Zeitspanne für die Analyse der Malware, ohne durch den Windows Defender gestört zu werden.

Es ist wichtig zu beachten, dass dies ein erwünschtes Verhalten einer Antivirensoftware ist und den normalen Betrieb eines Computers schützt. In einem Kontext der Malware-Analyse kann es jedoch als kontraproduktiv angesehen werden, da es das Sammeln von relevanten Daten behindert.

### 3.2.10 Erhebung und Weiterleitung der Daten

Das Pintool generiert nach jeder Anwendung eine Ausgabedatei, die die erfassten Ergebnisse beinhaltet. Diese Ergebnisse variieren je nach Konfiguration des Pintools. Da die Ausgabedatei durch jede neue Ausführung des Pintools überschrieben wird, ist es unerlässlich, dass die Daten unmittelbar nach ihrer Generierung gespeichert und umbenannt werden. Anschließend sollten diese Resultate umgehend an den FTP-Server weitergeleitet werden, wo sie für die nachfolgende Verarbeitung bereitstehen.

Es bestehen verschiedene Optionen zur weiteren Verarbeitung dieser Daten. Eine Möglichkeit besteht darin, nach Abschluss der Analyse die Ubuntu-**VM** wieder mit dem Internet zu verbinden, über die Bridged-Network Option und die Daten in eine Cloud hochzuladen. Alternativ kann auch die Funktion "SShared Folder" von VirtualBox selbst genutzt werden. Dies erfordert jedoch die Installation der Guest Additions, was auf der Ubuntu-Maschine problemlos umsetzbar sein sollte. Durch diese Verfahren wird gewährleistet, dass die erfassten Daten für nachfolgende Analysen und Vergleiche zur Verfügung stehen und ihre Kontinuität sichergestellt wird.

### 3.2.11 Dokumentation

Die Dokumentation der durchgeführten Arbeit ist ein unerlässlicher Bestandteil des Projekts. Insbesondere mit dem Ziel, die Ergebnisse zu veröffentlichen und anderen Forschern die Möglichkeit zu geben, das System in ihrer eigenen Umgebung aufzubauen und Malware-Analysen durchzuführen, kommt der Dokumentation eine besondere Bedeutung zu.

Zu diesem Zweck wurde der Quellcode zusammen mit detaillierten Installationsanweisungen in einem Github-Repository bereitgestellt. Dies ermöglicht einen einfachen Zugang und Download für alle Interessierten und gewährleistet die Nachvollziehbarkeit und Reproduzierbarkeit der Arbeit. Das Repository ist unter dem folgenden Link erreichbar: [github.com/Hamudah/dynamic-malware-analysis](https://github.com/Hamudah/dynamic-malware-analysis).

Alle für die Implementierung erforderlichen Skripte und Konfigurationsdateien sind im Github-Repository vorhanden. Die Struktur des Repository ist so angelegt, dass sich alle Konfigurationsdateien im Ordner `config_files` befinden. Im Ordner `executables` ist eine `Ninite.exe` zu finden, welche die Installation auf der Windows-Maschine vereinfacht. Eine detaillierte Erläuterung hierzu findet sich in Kapitel 4. Die im Ordner `Scripts` enthaltenen Skripte sind weiter unterteilt in Shell-, Bat- und Python-Skripte.

## 4 Implementierung

Nachdem wir uns ausführlich mit den Anforderungen und den Entwurfsentscheidungen in Kapitel 3 auseinandergesetzt haben, wenden wir uns nun der Implementierung der automatisierten Testumgebung zu. Dieser Prozess wird in mehreren Schritten genauer erläutert.

Zunächst werden wir detailliert auf den Aufbau der virtuellen Umgebung eingehen. Anschließend wird die Vorbereitung der Maschinen für die Malware-Erkennung detailliert beschrieben. Schließlich wird auf die Anpassung und Anwendung des Pintools für die spezifischen Anforderungen der Malware-Erkennung eingegangen. Durch diese systematische Herangehensweise wird sichergestellt, dass die Implementierung effektiv und präzise erfolgt, um optimale Ergebnisse in der Malware-Erkennung zu erzielen.

### 4.1 Erstellung der Testumgebung

Die Erstellung der Testumgebung erfordert bestimmte Voraussetzungen. VirtualBox 7 oder eine neuere Version muss auf dem System installiert sein, auf dem die Umgebung erstellt werden soll. Darüber hinaus muss das System über ausreichende Ressourcen verfügen, um mindestens zwei VMn gleichzeitig betreiben zu können.

Die Ubuntu-Virtual-Machine benötigt relativ wenige Ressourcen, da der FTP-Server nur einen geringen Teil davon in Anspruch nimmt. Die Anforderungen an die Windows-Virtual-Machine sind jedoch höher, da diese einem realen System so nah wie möglich kommen sollte. Dies bedeutet, dass sie mindestens 2-4 Kerne an Leistung, mindestens 2-4GB RAM und etwa 100GB an Festplattenspeicher benötigt.

Wie in den vorherigen Kapiteln bereits diskutiert wurde, ist Malware in der Lage, sich intelligent der Analyse zu entziehen. Daher ist es von äußerster Wichtigkeit, die Windows-Maschine so realistisch wie möglich zu gestalten und zu konfigurieren.

Die ISO-Dateien sollten, wie in Abschnitt 3.2.3 beschrieben, für die automatisierte Installation durch VBoxManage vorbereitet sein. Eine detaillierte Anleitung zur Bearbeitung der ISO-Dateien finden sich im Anhang A.

Sobald diese Voraussetzungen erfüllt sind, können beide virtuellen Maschinen durch Ausführung des Skripts **complete\_setup.bat** erstellt werden. Alle im Skript verwendeten Befehle können kopiert und in ein Skript eingefügt werden, das auf macOS läuft.

Das Skript definiert mehrere Umgebungsvariablen, die Namen, Betriebssystemtypen und Speicherorte der Festplattendateien für die beiden virtuellen Maschinen festlegen. Darüber hinaus werden in diesem Skript alle Spezifikationen für die beiden Maschinen definiert, die je nach

Bedarf angepasst werden können. Zusätzlich wird ein internes Netzwerk mit dem Namen "MalwareAnalysis" automatisch erstellt, in welchem beide virtuellen Maschinen eingebunden sind. Diese Struktur gewährleistet die Isolation und gleichzeitig die Interaktion zwischen den beiden Maschinen für die Zwecke der Malware-Analyse.

Schließlich erfolgt nach der Einrichtung der VMs und ihrer Konfiguration die automatisierte Installation der beiden VMs. Diese befindet sich in den Zeilen 81 und 82 des Skriptes. Dort können Benutzername, Passwort, Sprache und weitere Eigenschaften der virtuellen Maschinen definiert werden.

Nach Ausführung des Skripts sollten sich zwei GUIs öffnen, in denen der Installationsfortschritt beobachtet werden kann. Sobald der Vorgang abgeschlossen ist, stehen die beiden für die Malware-Analyse benötigten virtuellen Maschinen zur Verfügung.

## 4.2 Vorbereitung der Maschinen zur Malware-Analyse

Nachdem die virtuellen Maschinen erstellt wurden, ist es nun notwendig, diese so zu konfigurieren, dass sie zur Malware-Analyse eingesetzt werden können. Initial ist eine aktive Internetverbindung für beide Maschinen erforderlich. Die Netzwerkeinstellungen lassen sich dabei jederzeit über die VirtualBox GUI oder via der Kommandozeile anpassen. Der entsprechende Befehl für die Kommandozeilenschnittstelle lautet: `VBoxManage modifyVM <VM_name> -nic1 nat`. Dieser Befehl erlaubt es, die Netzwerkeigenschaften der virtuellen Maschinen anzupassen und sicherzustellen, dass sie in der Lage sind, eine Internetverbindung zu etablieren, was für die anfängliche Konfiguration und Einrichtung notwendig ist.

Das Github-Repository sollte auf beiden Maschinen sowie auf dem Host-System heruntergeladen werden. Dies kann entweder über die Shared Folder Funktion von VirtualBox erreicht werden, um das Repository mit den Maschinen zu teilen, oder indem es direkt aus dem Internet heruntergeladen wird.

### 4.2.1 Host-Maschine

Um das Skript für die Analyse ausführen zu können, muss noch eine Voraussetzung erfüllt sein. Auf dem System, auf dem man es ausführen will, in unserem Fall das Windows Host-System, sollte eine virtuelle Umgebung (VENV) für Python erstellt werden. Dies wurde in unserem Fall mit Visual Studio Code (VS Code) realisiert. VS Code bietet eine detaillierte Anleitung zur Erstellung einer VENV für Python. Die Anleitung findet sich unter [VS Code Python Environment](#).

Sobald die VENV eingerichtet ist, muss nun die VBoxApi für Python installiert werden. In einem Terminal, in dem die VENV aktiv ist, kann die SDK durch den Befehl `python sdk\installer\vboxapisetup.py install` installiert werden. Zusätzlich müssen noch die `pywin32` und `virtualbox` Bibliotheken heruntergeladen werden. Dies kann durch den Befehl `pip install <library>` durchgeführt werden.

### 4.2.2 Ubuntu-Maschine

Auf der Ubuntu-Maschine kann, sobald das Repository verfügbar ist, das Shell-Skript `ftp_setup.sh` ausgeführt werden. Dieses Skript lädt den FTP-Server herunter und passt die Konfiguration entsprechend an. Darüber hinaus erstellt es einen zusätzlichen Benutzer namens “ftpuser”, der ausschließlich Zugriff auf das FTP-Verzeichnis hat.

Es wird auch ein SSH-Schlüssel für den Hauptbenutzer benötigt, der dazu dient, per SSH auf die Windows-Maschine zuzugreifen und dort remote ein Skript auszuführen. Details dazu werden im Abschnitt 4.3 beschrieben.

Nach Abschluss aller vorbereitenden Maßnahmen sollte die Verbindung der Ubuntu-VM zum Internet gekappt und die Netzwerkeinstellungen auf ein internes Netzwerk umgestellt werden. Diese Maßnahme dient dazu, die Testumgebung während der Malware-Analyse sicher und isoliert zu halten. Zusätzlich sollte der Ubuntu-VM eine manuelle IP-Adresse zugewiesen werden. In unserem Fall haben wir die Adresse 192.178.162.1 verwendet. Diese Zuweisung kann jedoch nach Belieben angepasst werden. Es ist jedoch wichtig zu beachten, dass entsprechende Änderungen auch im Skript `pin_tool_execution_arg.py` vorgenommen werden müssen. Weitere Details zu diesem Skript werden in Abschnitt 4.3 erläutert.

### 4.2.3 Windows-Maschine

Die Komplexität und der Umfang der Vorbereitungen auf der Windows-Maschine erfordern eine umfassendere und strukturierte Herangehensweise. Daher wird dieser Teil in drei spezifische Unterabschnitte unterteilt:

1. **Softwarevoraussetzungen und -installationen:** In diesem Abschnitt wird die benötigte Software und deren Installation diskutiert. Hier werden die spezifischen Werkzeuge und Anwendungen vorgestellt, die zur Durchführung der Malware-Analyse benötigt werden.
2. **System- und Netzwerkkonfiguration:** In diesem Abschnitt werden die erforderlichen Anpassungen und Konfigurationen am System und Netzwerk vorgestellt, die zur ordnungsgemäßen Durchführung der Malware-Analyse und zur Gewährleistung der Sicherheit und Isolation der Testumgebung notwendig sind.
3. **Datenaufbereitung:** Der letzte Abschnitt befasst sich mit der Aufbereitung und Organisation der Daten, insbesondere der Malware- und Goodware-Proben, die analysiert werden sollen. Hier wird auch die Methode zur Übertragung dieser Daten auf die VM beschrieben.

Jeder dieser Abschnitte stellt eine wesentliche Phase in der Vorbereitung der Windows-Maschine für die Malware-Analyse dar und wird detailliert dargestellt, um ein klares Verständnis der durchzuführenden Schritte zu gewährleisten.



### Softwarevoraussetzungen und -installationen

Zum Beginn ist eine aktive Internetverbindung vonnöten, um die erforderliche Software herunterzuladen und zu installieren.

Sobald das Repository auf der Windows-Maschine bereitsteht, sollte die Datei "**Ninite.exe**" ausgeführt werden, die sich im Ordner **Executables** befindet. Ninite ist ein kostenfreier Service, der eine auswahlgesteuerte und gebündelte Installation vieler Anwendungen und Laufzeitumgebungen erlaubt. Dieses Tool vereinfacht den Installationsprozess erheblich und spart Zeit, da es den Einzelinstallationsbedarf für jede Anwendung beseitigt. Zwar sind viele der durch Ninite bereitgestellten Anwendungen nicht direkt für die Malware-Analyse notwendig, dennoch tragen sie zur Authentizität des simulierten Systems bei, was entscheidend ist, um zuverlässige Analyseergebnisse zu erzielen.

Trotz der breiten Auswahl an Anwendungen, die Ninite bietet, sind einige zusätzliche Softwareinstallationen notwendig. Hierzu zählt ein SSH-Server, der eine sichere Verbindung zwischen den beiden virtuellen Maschinen ermöglicht. In diesem Projekt wurde der Bitvise SSH-Server verwendet, allerdings wäre jeder andere SSH-Server ebenso geeignet. Im SSH-Server wird der öffentliche SSH-Schlüssel, der zuvor auf der Ubuntu-Maschine erstellt wurde, eingetragen.

Weiterhin ist es erforderlich, **Microsoft Visual Studio Community 2019** herunterzuladen und zu installieren, da die **x86 Native Tools Command Prompt for VS 2019** für die Ausführung des Pintools benötigt wird. Da das Testsystem auf einer 32-Bit-Architektur basiert, ist die Version 2019 von Microsoft Visual Studio Community die neueste verwendbare Version.

Die Installation des Pintools steht als nächstes an. In diesem Projekt wurde das Pintool bereits vorkompiliert und vom Betreuer zur Verfügung gestellt. Es kann jedoch auch selbständig aus dem Internet heruntergeladen und mit der **x86 Native Tools Command Prompt for VS 2019** kompiliert werden. Dieser Punkt bietet Spielraum für die Integration anderer Analysetools, die dann an die spezifischen Anforderungen angepasst und entweder in das vorhandene Skript **pin\_tool\_execution\_arg.py** eingebettet oder auf dessen Grundlage ein neues Skript erstellt werden können. Das hier verwendete Pintool befindet sich im Anhang [A](#).

Darüber hinaus wird eine Anwendung wie **TrID** benötigt, um den Dateityp zu ermitteln. Neben **TrID** muss das **TrIDDefs.TRD package** heruntergeladen und im gleichen Verzeichnis wie die **TrID.exe** platziert werden.

Schließlich ist die Installation von **FakeNet** notwendig. Diese Anwendung simuliert eine Internetverbindung, um die Malware zu täuschen und ihr vorzuspiegeln, dass sie sich auf einem realen System mit aktiver Internetverbindung befindet.

Nachdem die erforderliche Software installiert wurde, ist es ratsam, einen Snapshot des Systems zu erstellen. Ein solcher Snapshot fungiert als Wiederherstellungspunkt, zu dem man das System zurücksetzen kann, falls etwas schiefgeht oder bestimmte Änderungen rückgängig gemacht werden müssen. Da die folgenden Schritte erhebliche Modifikationen am System vornehmen und die Malware zur Analyse vorbereiten, ist der Einsatz eines Snapshots besonders wertvoll. Er gewährleistet, dass man stets zu einem stabilen und funktionierenden Zustand zurückkehren kann. Deshalb sollte das Erstellen eines Snapshots vor der Fortsetzung nicht außer Acht gelassen werden.

## Datenaufbereitung

Nachdem die erforderliche Software auf dem System installiert wurde, muss als nächstes die Good- und Malware, die getestet werden soll, auf das Windows-System übertragen werden. Hierfür gibt es verschiedene Möglichkeiten, etwa das Herunterladen aus dem Internet oder das Übertragen der Daten über die Shared Folder-Funktion von VirtualBox. Wie bereits erwähnt, bietet VirusShare Zugang zu einer umfangreichen Sammlung von Malware-Proben, während die SysInternals-Suite eine gute Quelle für Goodware-Proben darstellt.

Der nächste Schritt hängt von den spezifischen Malware-Proben ab, die zur Verfügung stehen. In unserem Fall wurden die Malware-Proben von VirusShare bezogen, was bedeutet, dass sie zuerst aus dem Archiv extrahiert und ihr Dateityp bestimmt werden muss, um sie ausführen zu können. Oftmals lässt sich der Typ eines Programms anhand seiner Kopfzeilen identifizieren. Aus diesem Grund wurde ein Skript namens `find_file_type_and_create_exe.py` entwickelt, das mithilfe von TrID alle Malware-Proben, die für Windows 32Bit geschrieben wurden, identifiziert und in ausführbare Anwendungen umwandelt. So können sie im weiteren Verlauf problemlos ausgeführt werden.

Jetzt sollten wir über ausreichend Good- und Malware-Proben verfügen, die wir für eine automatisierte Analyse nutzen können.

## System- und Netzwerkkonfiguration

Schließlich sind einige System- und Netzwerkeinstellungen erforderlich. Zu diesem Zeitpunkt empfiehlt es sich erneut, einen Snapshot der Windows-Maschine anzufertigen. Durch die bisherige Vorbereitung von Software und Daten hat man einen stabilen Zustand erreicht, zu dem man bei Bedarf zurückkehren kann.

Nach Abschluss aller vorherigen Schritte sollte das System wieder auf das interne Netzwerk umgestellt werden. Es ist jetzt auch an der Zeit, die IP-Adresse der Maschine zu ändern. Diese wurde manuell in den Adapteroptionen auf 192.178.162.2 geändert. Obwohl die Auswahl der IP-Adresse flexibel ist, sollte sie bei Verwendung des Shell-Skripts `execute_pintool.sh` entsprechend angepasst werden.

Nach der Änderung der IP-Adresse muss der Port für den FTP-Server freigegeben werden. Standardmäßig verwendet FTP die Ports 20 und 21, aber es kann manuell eingestellt werden, welcher Port verwendet werden soll. Dazu muss in der Firewall unter Eingehende Regeln eine neue Regel erstellt werden, um eine FTP-Verbindung zu ermöglichen. In der Regel ist dies für SSH nicht erforderlich, aber nach einem Test kann auch dies angepasst werden. Unter den Python-Skripten befindet sich das Skript `test_ftp_connection.py`, mit dem überprüft werden kann, ob die Verbindung zwischen den Maschinen erfolgreich ist.

Schließlich sind wir nun in der Lage, eine SSH- und eine FTP-Verbindung zwischen beiden Maschinen herzustellen. Als nächstes müssen wir die **VirtualBox Guest Editions** deinstallieren. Dieser Schritt ist entscheidend, da nahezu jede moderne Malware überprüft, ob sie sich auf einer virtuellen Maschine oder einem physischen System befindet. Nach der Deinstallation der **VirtualBox Guest Editions** sollte das System neu gestartet werden, um alle Änderungen zu

übernehmen. Es sollte auch sichergestellt werden, dass die Guest Editions aus den Laufwerken ausgeworfen wurden.

Schließlich müssen wir noch den Windows Defender deaktivieren. Wie bereits in Abschnitt 3.2.9 erläutert, kann der Windows Defender nicht vollständig ausgeschaltet werden. Wir sollten jedoch in der Lage sein, die meisten seiner Funktionen zu deaktivieren. Am Ende sollten alle Optionen deaktiviert sein, mit Ausnahme der Echtzeiterkennung. Obwohl alle Windows Defender-Optionen durch das Python-Skript, das das Pintool ausführt, deaktiviert werden, ist das Verhalten des Defenders nicht immer vorhersehbar. Daher ist es ratsam, doppelt sicher zu gehen und diese Optionen manuell zu deaktivieren.

### 4.3 Automatisierung der Malware-Analyse

Nachdem alle vorherigen Schritte abgeschlossen sind, können wir nun zur automatisierten Analyse übergehen. Dabei wird die Analyse in zwei Teile geteilt: die für Goodware und die für Malware.

#### 4.3.1 Automatisierte Analyse von Goodware

Die Goodware-Analyse ist im Vergleich zur Malware-Analyse verhältnismäßig einfach gestaltet. Wie bei der Malware-Analyse wird auch hier die zu untersuchende Software in einer CSV-Datei gespeichert. In Abbildung 4.1 ist ein Ausschnitt einer solchen CSV-Datei dargestellt, die die Software im Format [Name, Pfad zur Software, Argumente] speichert. Das Python-Skript `add_goodware_exe_to_csv.py` ermöglicht das einfache Hinzufügen der zu testenden Software zu einer CSV-Datei. Dieses Skript durchsucht alle Anwendungen im Zielverzeichnis und erstellt eine CSV-Datei in dem vorgegebenen Format.

Beim ersten Start von SysInternals ist jedoch noch die Zustimmung zur Endbenutzer-Lizenzvereinbarung (EULA) erforderlich. Deshalb kann bei der ersten Verwendung des Pintools das Skript `eula.py` genutzt werden, das als Argument `-accepteula` anhängt und somit die Ausführung ermöglicht.

```
1 name,exe,args
2 accesschk64,D:\BA\SysInternals\accesschk64.exe,-accepteula
3 ADEplorer64,D:\BA\SysInternals\ADEplorer64.exe,-accepteula
4 ADInsight64,D:\BA\SysInternals\ADInsight64.exe,-accepteula
5 adrestore64,D:\BA\SysInternals\adrestore64.exe,-accepteula
6 Autologon64,D:\BA\SysInternals\Autologon64.exe,-accepteula
7 Autoruns64,D:\BA\SysInternals\Autoruns64.exe,-accepteula
```

Abb. 4.1: CSV Datei

Ist eine CSV-Datei mit der zu testenden Software vorhanden, kann die Analyse beginnen. Hierbei muss die Ubuntu-Maschine aktiv sein, um die Ergebnisse zu empfangen. Durch Ausführung des Skriptes `pin_tool_execution.py` werden die in der CSV-Datei gelisteten Anwendungen nacheinander ausgeführt.

Der Prozess der Ausführung und Analyse wird in Abbildung 4.2 als Flussdiagramm dargestellt. Zunächst führt das Pintool die Anwendungen nacheinander aus und wartet jeweils, bis eine Ausgabedatei erstellt wurde. Diese Ausgabedatei erhält den Namen der jeweiligen Software.

Anschließend wird geprüft, ob bereits eine Datei mit ähnlichem Namen existiert. Falls ja, wird die Dateigröße der neuen Datei mit der existierenden verglichen. Da in unserem speziellen Fall eine größere Datei mit mehr aufgezeichneten Systemaufrufen wertvoller ist, wird stets die größere Datei lokal gespeichert und danach an den FTP-Server weitergeleitet.

In Abbildung 4.3 wird der Output dargestellt, der bei erfolgreicher Ausführung entsteht..

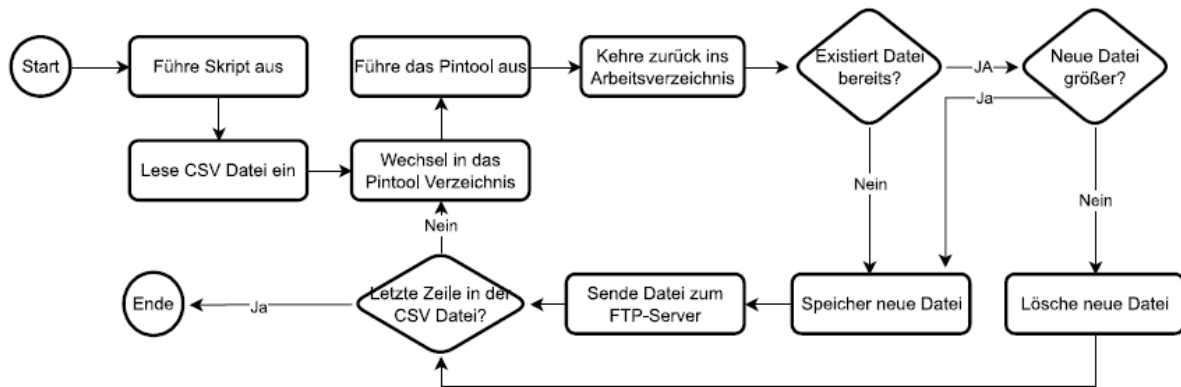


Abb. 4.2: Flussdiagramm der Prozessabläufe

```

Running for accesschk.exe...
Existing output file for accesschk.exe is larger. Keeping existing file.
Completed for accesschk.exe
Running for AccessEnum.exe...
Completed for AccessEnum.exe
Running for ADEplorer.exe...

```

Abb. 4.3: Ausgabe der Goodware-Analyse im Terminal

Das Skript kann nun für jegliche Goodware-Analyse, die durchgeführt werden soll, verwendet werden. Im zugehörigen Github-Repository finden sich im Verzeichnis `config_files` weitere CSV-Dateien, die Goodware enthalten, die auf Windows getestet werden könnte. Viele dieser Programme sind standardmäßig auf Windows installiert und müssen nicht zusätzlich heruntergeladen werden.

Im Gegensatz zur Malware-Analyse ist es hier nicht notwendig, die Maschine nach jeder Ausführung neuzustarten und auf den Snapshot zurückzusetzen, da das Verhalten von Goodware in der Regel nicht destruktiv ist und die Ergebnisse der anderen zu analysierenden Goodware nicht beeinträchtigt. Sollte dennoch ein Neustart und eine Rücksetzung auf den Snapshot gewünscht sein, kann die CSV-Datei einfach dem Skript, das die Malware ausführt, zur Verfügung gestellt werden.

#### 4.3.2 Automatisierte Analyse von Malware

Die Malware-Analyse ist im Vergleich zur Goodware-Analyse etwas komplexer.

Wie bei der Goodware-Analyse wird auch hier eine CSV-Datei benötigt, in der die zu analysierende Malware aufgelistet ist. Jedoch wird hier das spezifische Skript `add_malware_exe_to_csv.py`

eingesetzt, um alle Malware-Anwendungen aus einem bestimmten Ordner zur CSV-Datei hinzuzufügen. Nach Erstellung der CSV-Datei sind alle Vorbereitungen für die Analyse abgeschlossen.

Falls noch kein Snapshot der Windows-Maschine erstellt wurde, ist dies jetzt für die automatisierte Analyse unerlässlich. Der Snapshot wird genutzt, um das System nach jeder Ausführung des Pintools wieder in den Ursprungszustand zurückzusetzen. Der Snapshot wurde "Äfter Hardening" genannt, kann aber nach Belieben umbenannt werden. Eine solche Umbenennung erfordert eine Anpassung im Skript `start_malware_analysis.py` in Zeile 62.

Im Sequenzdiagramm der Abbildung 4.4 wird der Ablauf der Ausführung des Skripts `start_malware_analysis.py` dargestellt. Dieses Skript erhält ein Argument, welches definiert, wie oft das Skript durchlaufen soll.

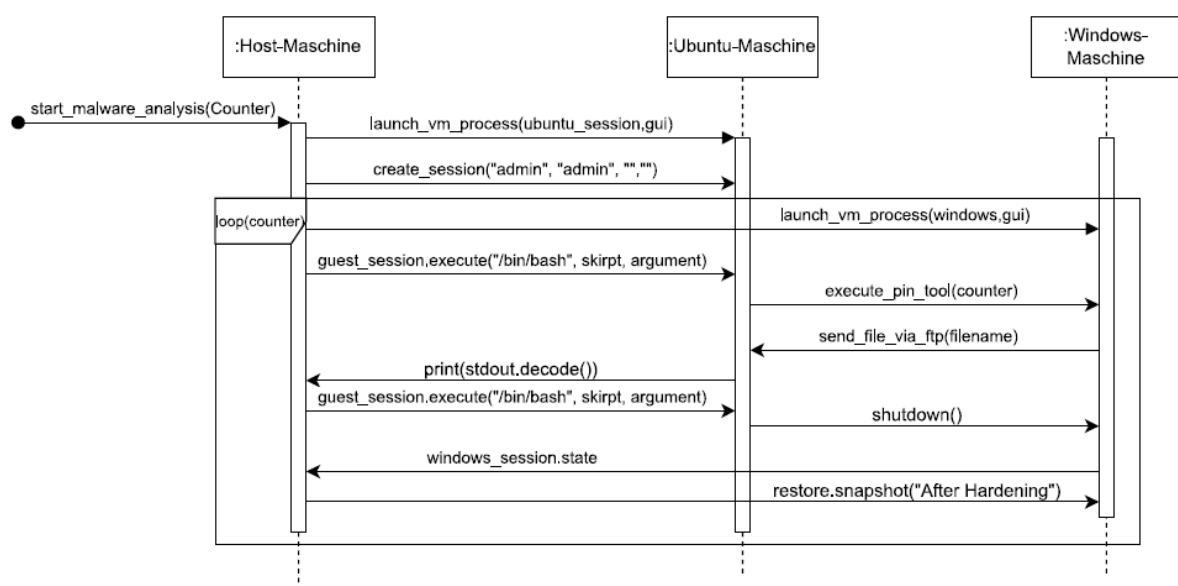


Abb. 4.4: Sequenzdiagramm

Der Prozess startet mit der Signalübertragung von dem Host-System an die beiden VMs über die VirtualBox API. Dabei wird einmalig eine Gästesitzung auf der Ubuntu VM mit dem Administratorenkonto erstellt oder für den Benutzer, für den zuvor der SSH-Schlüssel erstellt wurde, wie bereits in Abschnitt 4.2.2 erläutert.

Nun beginnt die Schleife ab diesem Punkt und interiert so oft durch, wie als Argument bei der Ausführung des Skriptes definiert wurde. Dabei wird identisch wie bei der Ubuntu-Maschine, die Windows-Maschine gestartet. Über die VirtualBox API kann nun mit dem Gastzugriff ein CLI-Befehl ausgeführt werden, der das Shell-Skript `execute_pintool.sh` startet. Dieses Skript erhält als Argument die aktuelle Durchlaufnummer.

Das Shell-Skript `execute_pintool.sh` stellt dann eine SSH-Verbindung zur Windows-Maschine her und führt dort das Skript aus, das das Pintool zur Malware-Analyse startet. Diese Herangehensweise ist aus zwei Gründen gewählt: Erstens wurden die Guest Additions auf der Windows-Maschine deinstalliert, sodass nicht alle API-Befehle möglich sind. Zweitens könnte fortgeschrittene Malware die VirtualBox API erkennen und sich der Ausführung verweigern, wenn sie direkt

aufgerufen wird. Bei vorherigen Tests hat sich gezeigt, dass die Anzahl der ausführbaren Malware deutlich geringer war, wenn der direkte Zugriff genutzt wurde, daher erfolgt der Zugriff über SSH von der Ubuntu-Maschine aus.

Infolgedessen wird das Skript `pin_tool_execution_arg.py` ausgeführt. Dieses Skript bietet weitere Funktionen im Vergleich zu `pin_tool_execution.py`, das für die Goodware-Analyse verwendet wurde. Es erhält das Argument von `execute_pintool.sh` und weiß dadurch, welche Malware in der CSV-Datei nun getestet werden soll.

Im Gegensatz zur Goodware-Analyse werden hier zusätzliche Funktionen aufgerufen, um die Ausführung der Malware zu gewährleisten. Zunächst wird die Funktion `disable_windows_defender` aufgerufen, wie in Abbildung 4.5 dargestellt. Dies ist notwendig, da sonst einige Malware durch den Windows Defender an der Ausführung gehindert werden könnte.

```

9  def disable_windows_defender():
10     # Disable Windows Defender Real-time Protection
11     disable_realtime_protection_cmd = r'reg add "HKLM\SOFTWARE\Policies\Microsoft\Windows
12     subprocess.run(["powershell.exe", "-Command", disable_realtime_protection_cmd], check
13
14     # Disable Windows Defender AntiSpyware
15     disable_defender_cmd = r'reg add "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender"
16     subprocess.run(["powershell.exe", "-Command", disable_defender_cmd], check=True)
17
18     # Stop Windows Defender services
19     stop_defender_services = [
20         "Set-Service -Name WinDefend -StartupType Disabled",
21         "Stop-Service -Name WinDefend",
22         "Set-Service -Name WdNisSvc -StartupType Disabled",
23         "Stop-Service -Name WdNisSvc",
24         "Set-Service -Name WdBoot -StartupType Disabled",
25         "Stop-Service -Name WdBoot",
26     ]
27     for cmd in stop_defender_services:
28         cmd = f'Start-Process powershell.exe -ArgumentList "{cmd}" -Verb RunAs'
29         subprocess.run(["powershell.exe", "-Command", cmd], check=True)

```

Abb. 4.5: `disable_windows_defender`

Eine weitere zusätzliche Funktion ist das Fakenet vor der Ausführung des Pintools noch gestartet wird. Dies ist notwendig, da einige Malware, nach einer aktiven Internetverbindung suchen, wie bereits in Abschnitt 4.2.3 erklärt wurde. Die Anwendung wird durch das Skript als Administrator vor der Ausführung des Pintools ausgeführt.

Nachdem diese beiden Funktionen ausgeführt wurden, tritt das Pintool in Aktion, wobei es die anstehende Malware ausführt. Dabei erhält das Skript die aktuelle Durchlaufnummer als Argument und führt die Malware aus, die in der CSV-Datei an dieser Position steht.

Nachdem das Pintool seinen Lauf beendet hat und die Ausgabedatei an den FTP-Server gesendet wurde, führt die Ubuntu-Maschine über SSH ein zusätzliches Skript aus, welches die Windows-Maschine herunterfährt. Im Anschluss daran wird der Zustand der Maschine auf den zuvor erstellten Snapshot zurückgesetzt. Dann startet der nächste Durchlauf, indem die Windows-Maschine erneut eingeschaltet wird und der Prozess sich wiederholt.

Im Unterschied zur Goodware-Analyse erfolgt bei dieser Methode keine Überprüfung, ob eine Datei mit ähnlichem Namen bereits existiert und analysiert wurde. Dieser Unterschied ist auf die erhöhte Komplexität zurückzuführen, die eine solche Überprüfung in diesem Automatisierungsprozess mit sich gebracht hätte, ohne dabei einen signifikanten Nutzen zu bieten.

Bei der Nutzung von VirtualBox können gelegentlich Probleme auftreten, bei denen die Windows-Maschine hängen bleibt und nicht ordnungsgemäß herunterfährt. Wenn das System dann versucht, sich selbst zu reparieren, kann es in diesem Zustand hängen bleiben. In solchen Fällen muss die Windows-Maschine manuell über die VirtualBox-GUI geschlossen werden. Nachdem dieses Problem behoben wurde, sollte das Skript jedoch problemlos weiterlaufen.

Während des gesamten Prozesses, der hier beschrieben wurde, werden alle Schritte im Terminal protokolliert und können dort verfolgt werden. Wenn der Prozess ohne Probleme durchläuft, erhält man eine Ausgabe wie in Abbildung 4.6. Wenn keine Ausgabe erfolgt, wird dies ebenfalls im Terminal angezeigt, wie in Abbildung 4.7 zu sehen ist.

```
Current state: Paused
Current state: Paused
Start restoring Snapshot from the Windows machine
Windows machine restored to snapshot
Closed existing sessions on Windows machine
Started Windows virtual machine
Running the script!
Executed the Script 26 times!
The operation completed successfully.
The operation completed successfully.
Running for VirusShare_059bfdc6ba7a4d64364e9c35ec455e75...
Completed for VirusShare_059bfdc6ba7a4d64364e9c35ec455e75
```

**Abb. 4.6:** Ausgabe bei Erfolgreicher Analyse

```
Start restoring Snapshot from the Windows machine
Windows machine restored to snapshot
Closed existing sessions on Windows machine
Started Windows virtual machine
Running the script!
Executed the Script 10 times!
The operation completed successfully.
The operation completed successfully.
Running for VirusShare_0597c408db9c43a4eb4efa3a9dbcc249...
Error: 'test.csv' not generated for VirusShare_0597c408db9
and the software command.
Completed for VirusShare_0597c408db9c43a4eb4efa3a9dbcc249
```

**Abb. 4.7:** Ausgabe bei fehlgeschlagener Analyse

Um zu vermeiden, dass nach einem Durchlauf des Skripts oder einer manuellen Unterbrechung die gleiche Malware erneut getestet wird, muss die CSV-Datei manuell aktualisiert werden. Anschließend wird der Snapshot vom aktuellen Zustand erneut erstellt. So wird sichergestellt, dass bei einem Neustart des Skripts die Malware-Analyse nicht von vorne beginnt, sondern an der Stelle fortgesetzt wird, wo sie unterbrochen wurde. Dieser Schritt der manuellen Aktualisierung der CSV-Datei ist ein wichtiger Aspekt, um die Effizienz der Malware-Analyse zu gewährleisten.



## 5 Auswertung und Ausblick

Das Ergebnis dieser Arbeit wird hauptsächlich anhand der Implementierung und des Designs gemessen, die in Kapitel 3 und 4 erläutert wurden.

### 5.1 Bewertung des Systems

Die Leistung des erstellten Systems wird anhand der in Kapitel 3.2 beschriebenen Anforderungen gemessen. Zu Beginn wurden klare Vorgaben definiert, was das System können sollte. Beim Abgleichen des Systems mit diesen Anforderungen wird deutlich, dass alle gestellten Anforderungen weitgehend erfüllt wurden.

Das System funktioniert wie geplant und befindet sich in einem einwandfreien Zustand. In der Vergangenheit mussten alle Tests manuell durchgeführt werden, ein Prozess, der im Durchschnitt zwischen 3-5 Minuten dauerte. Im Vergleich dazu benötigt der automatisierte Prozess nur 1-3 Minuten. Dies ist abhängig von der zu testenden Malware, da das Pintool bei komplexen Anwendungen länger arbeitet. In Abbildung 5.1 kann man anhand der Zeitstempel erkennen, wann die Anwendungen an den FTP-Server gesendet wurden und kann daher abschätzen, wie lange eine solche Ausführung dauert. Zwischen dem untersten Beispiel in der Abbildung und dem nächsten lagen zwar 5 Minuten, dies lag jedoch daran, dass es für eine Anwendung kein Ergebnis gab, wie bereits in Abbildung 4.7 dargestellt wurde. Bei den anderen Ergebnissen kann man jedoch einen deutlichen Trend von 2 Minuten erkennen. Im Durchschnitt können damit etwa 30 Malware-Proben pro Stunde getestet werden.








Name	Größe	^ Geändert	
 VirusShare_059dafd0142751eb70f36c55...	8,8 kB	20:19	☆
 VirusShare_059d949f71802a435f5c92d4...	4,9 kB	20:17	☆
 VirusShare_059d73e8339f7d959a6d80a7...	10,5 kB	20:15	☆
 VirusShare_059d5e2a9fbb13fb259030f7...	5,7 kB	20:12	☆
 VirusShare_059d483a1cb46f85ccbcc40b...	25,4 kB	20:10	☆
 VirusShare_059d35352863741e62b5f6b2...	26,7 kB	20:08	☆
 VirusShare_059bfdc6ba7a4d64364e9c35...	3,2 kB	20:03	☆

Abb. 5.1: Ausgabe auf dem FTP-Server

Darüber hinaus hat die Automatisierung den Vorteil, dass sie keine Fehler macht. Als Menschen neigen wir dazu, Schritte zu vergessen und zu übersehen, die zum einen das Ergebnis verfälschen und zum anderen das Host-System gefährden könnten.

Zudem kann die durch die Automatisierung gewonnene Zeit für andere Aufgaben verwendet werden, wie z.B. die Auswertung der Ergebnisse, das Finden neuer Malware-Proben usw. Durch

die Automatisierung gewinnt man wertvolle Zeit, die sonst für zeitraubende Wiederholung der gleichen Befehle aufgewendet würde.

Auch die Tatsache, dass man nun in der Lage ist, eine solche Testumgebung innerhalb weniger Minuten lokal aufzubauen und Malware nun von nahezu jedem System analysieren zu können, ist im Kampf gegen Malware von großem Wert. Malware-Analyse ist extrem kostspielig, da solche Anwendungen oft nur für eine große Summe Geld zur Verfügung stehen, wie z.B. Aviras Cloud Sandbox [API](#) [5].

## 5.2 Ausblick

Nach Abschluss der Arbeit und nach einer längeren Beobachtungsphase des Systems sind natürlich neue Ideen und Verbesserungsvorschläge entstanden.

Der wohl wichtigste Aspekt ist die Skalierung dieses Systems. Es ist relativ einfach, im Malware-Analyse-Skript eine zusätzliche Windows-Maschine einzurichten, die parallel zur anderen Windows-Maschine läuft und auf der ebenfalls Malware getestet wird. Die Herausforderung besteht hier jedoch in den begrenzten Ressourcen, da ein gewöhnlicher PC oder Laptop nur über eine bestimmte Anzahl von RAM und CPUs verfügt, die den Windows-Systemen zugewiesen werden können. Daher wäre die Überlegung, das Ganze auf einem Server einzurichten, der skaliert werden kann. Zwar würde ein Server Kosten verursachen, es gibt jedoch viele Online-Anbieter, bei denen man relativ günstig an einen Server mit umfangreicher Hardware gelangen kann. Wie bereits in Abschnitt 5.1 erwähnt, kann man auf einem System durchschnittlich 30 Malware-Proben pro Stunde analysieren, aber wir hatten in Abschnitt 1.1 erwähnt, dass täglich fast 400.000 neue Malware-Arten auftreten. Daher wäre dies ein wichtiger Punkt, der angegangen werden könnte.

Darüber hinaus könnte man neben dem Intel Pintool weitere Analysetools auf dem System zur Verfügung stellen. Zwar kann man in dem bereitgestellten Skript das Pintool einfach durch ein anderes Tool ersetzen, aber es wäre von Vorteil, wenn von Anfang an mehrere solche Analysetools zur Verfügung gestellt werden.

Ein weiterer Verbesserungsvorschlag wäre die Entwicklung einer ([GUI](#)) für diese Anwendung, da die Nutzung der Kommandozeile für viele ohne Erfahrung eine Herausforderung darstellen könnte. Es könnte eine [GUI](#) entwickelt werden, über die man die zu testenden Daten hochladen kann und die diese dann automatisiert an die Maschine weiterleitet. Auch die Möglichkeit, die Analyse darüber zu starten und zu stoppen, wurde diskutiert, jedoch war die Zeit für die Erstellung einer solchen [GUI](#) nicht vorhanden.

## 6 Fazit

Die Schlussbetrachtung dieser Arbeit offenbart, dass das konzipierte System ein essenzielles Werkzeug für Forschende darstellt, die eine lokale Malware-Analyse durchführen wollen, ohne dafür finanzielle Mittel aufwenden zu müssen.

Die Konfiguration einer solchen Umgebung stellt eine beträchtliche Forschungsaufgabe dar und beinhaltet ein hohes Maß an Komplexität, das viele Forschende vor bedeutende Hürden stellt und ihnen die Möglichkeit entzieht, Malware-Analysen durchzuführen. Diese Arbeit und die im Github-Repository zur Verfügung gestellte Dokumentation haben jedoch den Arbeitsaufwand und die Komplexität für den Aufbau einer solchen Umgebung drastisch reduziert. Selbst eine Person ohne umfangreiche Erfahrung sollte nun in der Lage sein, Malware-Analysen durchzuführen.

Neben der erheblichen Vereinfachung und Effizienzsteigerung bei der Einrichtung einer Malware-Analyseumgebung, stellt dieses System einen wertvollen Beitrag zur Sicherheitsforschung dar. Durch die Bereitstellung eines Zugangs zu solchen Werkzeugen und Ressourcen fördert es die Demokratisierung der Malware-Forschung und ermöglicht es einem breiteren Spektrum an Fachleuten, sich an der Identifizierung und Bekämpfung von Malware zu beteiligen.

Darüber hinaus führt die Automatisierung des Prozesses zu einer erheblichen Verbesserung der Analyseleistung. Indem sie Arbeitszeit, die zuvor für routinemäßige und sich wiederholende Aufgaben aufgewendet wurde, freisetzt, können Forschende ihre Anstrengungen auf komplexere und wertvollere Aspekte der Malware-Analyse fokussieren.

Letztlich bietet das System eine robuste Basis für zukünftige Verbesserungen und Erweiterungen. Es wurde bereits über potenzielle Optimierungen wie die Skalierung auf Serverebene, die Integration zusätzlicher Analyse-Tools und die Entwicklung einer grafischen Benutzeroberfläche diskutiert. All diese Potenziale unterstreichen die fortwährende Relevanz und den Wert dieses Systems als Beitrag zur Malware-Forschung.

Das entwickelte System erfüllt die ursprünglich gesteckten Ziele und bietet darüber hinaus Potenzial für weitere Forschung und Entwicklung in diesem Bereich. Es demonstriert die Realisierbarkeit und den Nutzen der Automatisierung von Malware-Analysen und stellt einen wertvollen Ausgangspunkt für zukünftige Arbeiten in diesem Bereich dar.

# A Anhänge

Alle hier aufgelisteten Skripte, mit Ausnahme von `MyPinTool.cpp`, sind im GitHub-Repository unter folgendem Link zu finden: <https://github.com/Hamudah/dynamic-malware-analysis>.

## A.1 complete\_setup.bat

```
1 @echo off
2
3 REM Name of the virtual machines
4 set VM_NAME1=FTP_SERVER2
5 set VM_NAME2=WindowsMalware2
6
7 REM Name of the distros
8 set VM_OSTYPE1=Ubuntu22_LTS_64
9 set VM_OSTYPE2=Windows10
10
11 REM Location of the virtual machine disk in MB
12 set VM_DISK1=D:/VM_HDD/%VM_NAME1%.vdi
13 set VM_DISK2=D:/VM_HDD/%VM_NAME2%.vdi
14
15 REM VBoxManage Path
16 set VBOXMANAGE_EXECUTABLE="C:\Program_
    Files\Oracle\VirtualBox\VBoxManage.exe"
17
18 REM Windows
19 #####
20 REM Windows
21
22 REM Delete VM with same name if it exists, otherwise create new one
23 %VBOXMANAGE_EXECUTABLE% unregistervm %VM_NAME2% --delete
24 %VBOXMANAGE_EXECUTABLE% createvm --name %VM_NAME2% --register
25
26 REM define OS TYPE
27 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME2% --ostype %VM_OSTYPE2%
28
29 REM define vm specs
30 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME2% --cpus 4 --memory 4096
    --vram 128
31
32 REM define network settings; Here internal network call MalwareAnalysis
33 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME2% --nic1 intnet --intnet1
    MalwareAnalysis
```

```

34
35 REM Change graphic Controll, since standard is VboxVGA which flickers
36 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME2% --graphicscontroller vmsvga
37
38 REM create hdd and define path
39 %VBOXMANAGE_EXECUTABLE% createhd --filename %VM_DISK2% --size 100000
    --variant Standard
40 %VBOXMANAGE_EXECUTABLE% storagectl %VM_NAME2% --name "SATA_Controller"
    --add sata --controller IntelAHCI
41 %VBOXMANAGE_EXECUTABLE% storageattach %VM_NAME2% --storagectl "SATA_
    Controller" --port 0 --device 0 --type hdd --medium %VM_DISK2%
42
43 REM Attach ISO file
44 %VBOXMANAGE_EXECUTABLE% storageattach %VM_NAME2% --storagectl "SATA_
    Controller" --port 1 --device 0 --type dvddrive --medium
    "D:\VM_ISO\Win10Unattended.iso"
45
46 REM Define Boot Order
47 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME2% --boot1 dvd --boot2 disk
    --boot3 none --boot4 none
48
49 REM Ubuntu FTP
    #####
50
51 REM Delete VM with same name if it exists, otherwise create new one
52 %VBOXMANAGE_EXECUTABLE% unregistervm %VM_NAME1% --delete
53 %VBOXMANAGE_EXECUTABLE% createvm --name %VM_NAME1% --register
54
55 REM define OS TYPE
56 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME1% --ostype %VM_OSTYPE1%
57
58 REM define vm specs
59 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME1% --cpus 4 --memory 4096
    --vram 128
60
61 REM define network settings; Here internal network call MalwareAnalysis
62 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME1% --nic1 intnet --intnet1
    MalwareAnalysis
63
64 REM Change graphic Controll, since standard is VboxVGA which flickers
65 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME1% --graphicscontroller vmsvga
66
67 REM create hdd and define path
68 %VBOXMANAGE_EXECUTABLE% createhd --filename %VM_DISK1% --size 32768
69 %VBOXMANAGE_EXECUTABLE% storagectl %VM_NAME1% --name "SATA_Controller"
    --add sata --controller IntelAHCI
70 %VBOXMANAGE_EXECUTABLE% storageattach %VM_NAME1% --storagectl "SATA_
    Controller" --port 0 --device 0 --type hdd --medium %VM_DISK1%
71
72 REM Attach ISO file
73 %VBOXMANAGE_EXECUTABLE% storageattach %VM_NAME1% --storagectl "SATA_
    Controller" --port 1 --device 0 --type dvddrive --medium
    "D:\VM_ISO\ubuntu-custom\ubuntu-22.04.1-desktop-amd64.iso"

```

```

74
75 REM Define Boot Order
76 %VBOXMANAGE_EXECUTABLE% modifyvm %VM_NAME1% --boot1 dvd --boot2 disk
    --boot3 none --boot4 none
77
78 REM #####
79
80 REM Perform unattended installation
81 %VBOXMANAGE_EXECUTABLE% unattended install %VM_NAME2%
    --iso="D:\VM_ISO\Win10Unattended.iso" --start-vm=gui --user=hamud
    --password=Malware --full-user-name="Malware" --install-additions
    --locale=de_DE --time-zone=CET --language=de --key=""
82 %VBOXMANAGE_EXECUTABLE% unattended install %VM_NAME1%
    --iso="D:\VM_ISO\ubuntu-custom\ubuntu-22.04.1-desktop-amd64.iso"
    --start-vm=gui --user=FTP-Server --password=FTP-Server
    --full-user-name="Hamud" --install-additions --locale=de_DE
    --time-zone=CET --language=de
83
84 REM Start both VMs
85 %VBOXMANAGE_EXECUTABLE% startvm %VM_NAME1% --type headless
86 %VBOXMANAGE_EXECUTABLE% startvm %VM_NAME2% --type headless

```

## A.2 start\_malware\_analysis.py

```

1 import sys
2 import time
3 import virtualbox
4 from virtualbox.library import LockType, MachineState, SessionState
5
6
7 def close_existing_sessions(machine, session):
8     machine.lock_machine(session, LockType.shared)
9     if session.state == SessionState.locked:
10         session.unlock_machine()
11
12
13 def main(num_files):
14     vbox = virtualbox.VirtualBox()
15     windows_machine = vbox.find_machine("WindowsMalware32bit")
16     ubuntu_machine = vbox.find_machine("FTP_SERVER2")
17
18     ubuntu_session = virtualbox.Session()
19     close_existing_sessions(ubuntu_machine, ubuntu_session)
20     print("Closed_existing_sessions_on_Ubuntu_machine")
21
22     ubuntu_machine.launch_vm_process(ubuntu_session, "gui", [])
23     print("Started_Ubuntu_virtual_machine")
24     time.sleep(60)
25

```

```

26     guest_session =
        ubuntu_session.console.guest.create_session("hamud", "hamud",
            "", "")
27
28     for i in range(num_files):
29         windows_session = virtualbox.Session()
30         close_existing_sessions(windows_machine, windows_session)
31         print("Closed_existing_sessions_on_Windows_machine")
32         time.sleep(3)
33
34         windows_machine.launch_vm_process(windows_session, "gui", [])
35         print("Started_Windows_virtual_machine")
36         time.sleep(45)
37
38         argument = str(i)
39         print(f"Running_the_script!")
40         process, stdout, stderr = guest_session.execute("/bin/bash", [
41             "/home/hamud/Schreibtisch/BA/dynamic-malware-analysis-main_
                /Scripts/shell/execute_pintool.sh",
42             argument
43         ])
44         process.wait_for(8, 120000)
45         print(f"Executed_the_Script_{i+1}_times!")
46         print(stdout.decode())
47         print("Shutting_down_the_machine!")
48         process, stdout, stderr = guest_session.execute("/bin/bash", [
49             "/home/hamud/Schreibtisch/BA/dynamic-malware-analysis-main_
                /Scripts/shell/shutdown.sh"
50         ])
51         time.sleep(30)
52
53         while windows_machine.state != MachineState.powered_off:
54             print("Current_state:", windows_machine.state)
55             time.sleep(5)
56
57         if windows_session.state == SessionState.locked:
58             windows_session.unlock_machine()
59
60         windows_machine.lock_machine(windows_session, LockType.shared)
61         print("Start_restoring_Snapshot_from_the_Windows_machine")
62         snapshot = windows_machine.find_snapshot("After_Hardening")
63         progress = windows_session.machine.restore_snapshot(snapshot)
64         progress.wait_for_completion()
65         windows_session.unlock_machine()
66         print("Windows_machine_restored_to_snapshot")
67
68
69 if __name__ == "__main__":
70     if len(sys.argv) != 2:
71         print("Usage: python power_on_vms_loop.py <num_files>")
72         sys.exit(1)
73
74     num_files = int(sys.argv[1])

```



```
75     main(num_files)
```

### A.3 pin\_tool\_execution.py

```
1  import subprocess
2  import os
3  import time
4  import shutil
5  import csv
6  import ftplib
7
8  def execute_pin_command(software_name, command):
9      pin_tool_dir =
10         r'C:\Users\hamud\Desktop\pin-3.18-98332-gaebd7b1e6-msvc-windows
11         \source\tools\MyPinTool'
12         vcvarsall_path = r'C:\Program Files\Microsoft Visual
13         Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat'
14         vcvarsall_cmd = f'"{vcvarsall_path}" x86 > nul &&'
15         pin_cmd = f'{vcvarsall_cmd} ..\..\..\pin-t_obj-ia32\MyPinTool.dll
16         -tsc 1 -os test.csv -to 40 --{command}'
17
18     output_dir = os.path.join(os.getcwd(), 'Windows_System_Call_API',
19                               'Goodware')
20     if not os.path.exists(output_dir):
21         os.makedirs(output_dir)
22
23     original_dir = os.getcwd()
24     os.chdir(pin_tool_dir)
25
26     with open(os.devnull, 'w') as devnull:
27         subprocess.run(pin_cmd, shell=True, stderr=devnull,
28                        stdout=devnull)
29
30     os.chdir(original_dir)
31
32     test_csv_path = os.path.join(pin_tool_dir, 'test.csv')
33     if os.path.exists(test_csv_path):
34         output_file = os.path.join(output_dir, f'{software_name}.csv')
35         test_csv_size = os.path.getsize(test_csv_path)
36
37         if os.path.exists(output_file):
38             output_file_size = os.path.getsize(output_file)
39             if test_csv_size > output_file_size:
40                 print(f'New output file for {software_name} is larger.
41                       Replacing existing file.')
42                 shutil.move(test_csv_path, output_file)
43             else:
44                 print(f'Existing output file for {software_name} is
45                       larger. Keeping existing file.')
46                 os.remove(test_csv_path)
47         else:
```

```

41         shutil.move(test_csv_path, output_file)
42     else:
43         print(f"Error: 'test.csv' not generated for {software_name}.
44             Check your pin command and the software command.")
45
46     # Remove the test.csv file after processing
47     test_csv_path = os.path.join(pin_tool_dir, 'test.csv')
48     if os.path.exists(test_csv_path):
49         os.remove(test_csv_path)
50
51 def send_file_via_ftp(filename):
52     ftp = ftplib.FTP("192.178.162.1")
53     ftp.login(user="ftpuser", passwd="ftpuser")
54     ftp.cwd("upload")
55     base_filename = os.path.basename(filename)
56     with open(filename, "rb") as f:
57         ftp.storbinary(f'STOR {base_filename}', f)
58     ftp.quit()
59
60 def main():
61     goodwill_list_file =
62         r'C:\Users\hamud\Desktop\dynamic-malware-analysis-main\config_files
63         \goodware_list4.csv'
64
65     # Read the goodwill list
66     with open(goodwill_list_file, 'r') as f:
67         reader = csv.reader(f)
68         next(reader) # Skip the header row
69
70     # Execute the pin command for each goodwill and save the output
71     for row in reader:
72         if not row:
73             continue
74
75         software_name, *command_args = row
76         command = ' '.join([arg.strip('"') for arg in command_args
77                             if arg])
78         print(f'Running for {software_name}...')
79         execute_pin_command(software_name, command)
80         print(f'Completed for {software_name}')
81
82     # Send the output file via FTP if the new output file was
83     # saved
84     output_filename = os.path.join(os.getcwd(), 'Windows\System_
85         Call_API', 'Goodware', f"{software_name}.csv")
86     if os.path.exists(output_filename):
87         send_file_via_ftp(output_filename)
88
89     time.sleep(1)
90
91 if __name__ == '__main__':
92     main()

```

## A.4 pin\_tool\_execution\_arg.py

```

1  import subprocess
2  import os
3  import time
4  import shutil
5  import csv
6  import ftplib
7  import sys
8
9  def disable_windows_defender():
10     # Disable Windows Defender Real-time Protection
11     disable_realtime_protection_cmd = r'reg add \
        "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time\
        Protection" /v DisableRealtimeMonitoring /t REG_DWORD /d 1 /f '
12     subprocess.run(["powershell.exe", "-Command",
        disable_realtime_protection_cmd], check=True)
13
14     # Disable Windows Defender AntiSpyware
15     disable_defender_cmd = r'reg add \
        "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender" /v \
        DisableAntiSpyware /t REG_DWORD /d 1 /f '
16     subprocess.run(["powershell.exe", "-Command",
        disable_defender_cmd], check=True)
17
18     # Stop Windows Defender services
19     stop_defender_services = [
20         "Set-Service -Name WinDefend -StartupType Disabled",
21         "Stop-Service -Name WinDefend",
22         "Set-Service -Name WdNisSvc -StartupType Disabled",
23         "Stop-Service -Name WdNisSvc",
24         "Set-Service -Name WdBoot -StartupType Disabled",
25         "Stop-Service -Name WdBoot",
26     ]
27     for cmd in stop_defender_services:
28         cmd = f'Start-Process powershell.exe -ArgumentList "{cmd}" \
            -Verb RunAs '
29         subprocess.run(["powershell.exe", "-Command", cmd], check=True)
30
31 def stop_fakenet():
32     global fakenet_process
33     if fakenet_process:
34         fakenet_process.terminate()
35
36 def run_fakenet(fakenet_path):
37     global fakenet_process
38     with open(os.devnull, 'w') as devnull:
39         fakenet_process = subprocess.Popen(["powershell.exe",
            "-Command", fakenet_path, "-Verb", "runAs"], stdout=devnull,
            stderr=devnull)
40
41

```

```

42 def execute_pin_command(software_name, command):
43     pin_tool_dir =
44         r'C:\Users\hamud\Desktop\pin-3.18-98332-gaebd7b1e6-msvc-
45         \windows\source\tools\MyPinTool'
46     vcvarsall_path = r'C:\Program Files\Microsoft Visual
47         Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat'
48     vcvarsall_cmd = f'"{vcvarsall_path}"x86>nul&&'
49     pin_cmd = f'{vcvarsall_cmd}..\..\pin-tobj-ia32\MyPinTool.dll
50         -tsc1-os test.csv-to40--{command}'
51
52     output_dir = os.path.join(os.getcwd(), 'Windows\System\Call\API',
53         'Goodware')
54     if not os.path.exists(output_dir):
55         os.makedirs(output_dir)
56
57     original_dir = os.getcwd()
58     os.chdir(pin_tool_dir)
59
60     with open(os.devnull, 'w') as devnull:
61         subprocess.run(pin_cmd, shell=True, stderr=devnull,
62             stdout=devnull)
63
64     os.chdir(original_dir)
65
66     test_csv_path = os.path.join(pin_tool_dir, 'test.csv')
67     if os.path.exists(test_csv_path):
68         output_file = os.path.join(output_dir, f'{software_name}.csv')
69         test_csv_size = os.path.getsize(test_csv_path)
70
71         if os.path.exists(output_file):
72             output_file_size = os.path.getsize(output_file)
73             if test_csv_size > output_file_size:
74                 print(f'New output file for {software_name} is larger.
75                     Replacing existing file.')
76                 shutil.move(test_csv_path, output_file)
77             else:
78                 print(f'Existing output file for {software_name} is
79                     larger. Keeping existing file.')
80                 os.remove(test_csv_path)
81         else:
82             shutil.move(test_csv_path, output_file)
83     else:
84         print(f"Error: 'test.csv' not generated for {software_name}.
85             Check your pin command and the software command.")
86
87     # Remove the test.csv file after processing
88     test_csv_path = os.path.join(pin_tool_dir, 'test.csv')
89     if os.path.exists(test_csv_path):
90         os.remove(test_csv_path)
91
92 def send_file_via_ftp(filename):
93     ftp = ftplib.FTP("192.178.162.1")
94     ftp.login(user="ftpuser", passwd="ftpuser")

```

```

87     ftp.cwd("upload")
88     base_filename = os.path.basename(filename)
89     with open(filename, "rb") as f:
90         ftp.storbinary(f'STOR_{base_filename}', f)
91     ftp.quit()
92
93 def main():
94     if len(sys.argv) < 2:
95         print("Usage: python script.py <line_number>")
96         sys.exit(1)
97
98     try:
99         line_number = int(sys.argv[1])
100     except ValueError:
101         print("Error: line_number must be an integer")
102         sys.exit(1)
103
104     goodwill_list_file =
105         r'C:\Users\hamud\Desktop\dynamic-malware-analysis-main\config_files
106         \malware_list.csv'
107
108     # Read the goodwill list
109     with open(goodwill_list_file, 'r') as f:
110         reader = csv.reader(f)
111         next(reader) # Skip the header row
112
113     for index, row in enumerate(reader, start=1):
114         if not row or index != line_number:
115             continue
116
117         software_name, *command_args = row
118         command = '_'.join([arg.strip('"') for arg in command_args
119                             if arg])
120         print(f'Running for {software_name}...')
121         execute_pin_command(software_name, command)
122         print(f'Completed for {software_name}')
123
124         # Send the output file via FTP if the new output file was
125         # saved
126         output_filename = os.path.join(os.getcwd(), 'Windows\System_
127         Call_API', 'Goodware', f"{software_name}.csv")
128         if os.path.exists(output_filename):
129             send_file_via_ftp(output_filename)
130
131         time.sleep(1)
132         break
133
134     sys.exit(0) # Add this line to exit the script after sending the
135                 # file via FTP
136
137 if __name__ == '__main__':
138     disable_windows_defender()
139     fakenet_path = r'C:\Users\hamud\Desktop\Fakenet1.0b\FakeNet'

```

```

135     run_fakenet(fakenet_path)
136     try:
137         main()
138     finally:
139         stop_fakenet()

```

## A.5 add\_goodware\_exe\_to\_csv.py

```

1  import os
2  import csv
3
4  input_directory = r"C:\Users\hamud\Desktop\SysInternals" # Replace
    with the path to the directory containing the malware files
5  output_csv = r"C:\Users\hamud\Desktop\dynamic-malware-analysis-
6  main\dynamic-malware-analysis-main\config_files\goodware_list4.csv"
7
8  with open(output_csv, mode='w', newline='') as csv_file:
9      csv_writer = csv.writer(csv_file)
10     csv_writer.writerow(['name', 'path', 'arg'])
11
12     for filename in os.listdir(input_directory):
13         if filename.endswith(".exe") and '64' not in filename:
14             file_path = os.path.join(input_directory, filename)
15             csv_writer.writerow([filename, file_path])
16             print(f"Added_{filename}_to_the_CSV_list")

```

## A.6 add\_malware\_exe\_to\_csv.py

```

1  import os
2  import csv
3
4  exe_files_directory = r'C:\Users\hamud\Desktop\Malware'
5  output_csv_file =
    r'C:\Users\hamud\Desktop\dynamic-malware-analysis-main\config_files
6  \malware_list.csv'
7
8  def main():
9      exe_files = []
10
11     for exe_file in os.listdir(exe_files_directory):
12         if exe_file.endswith('.exe'):
13             software_name = os.path.splitext(exe_file)[0]
14             command = os.path.join(exe_files_directory, exe_file)
15             exe_files.append([software_name, command])
16
17     with open(output_csv_file, 'w', newline='') as csvfile:
18         csv_writer = csv.writer(csvfile)
19         csv_writer.writerow(['name', 'exe'])

```

```

20         for exe_file in exe_files:
21             csv_writer.writerow(exe_file)
22
23 if __name__ == '__main__':
24     main()

```

## A.7 eula.py

```

1  import csv
2  import subprocess
3  import time
4
5  def main():
6      csv_file =
7          r'C:\Users\hamud\Desktop\dynamic-malware-analysis-main\config_files
8          \goodware_list4.csv'
9
10     with open(csv_file, 'r') as f:
11         reader = csv.reader(f)
12
13     for row in reader:
14         if not row:
15             continue
16
17         software_name, software_path, *args = row
18         command = f'"{software_path}" -accepteula'
19
20         print(f"Running this program {software_name}")
21
22         try:
23             process = subprocess.Popen(command, shell=True)
24             time.sleep(5) # Wait for 5 seconds
25             process.terminate()
26         except Exception as e:
27             print(f"Error executing {software_name}: {e}")
28
29 if __name__ == '__main__':
30     main()

```

## A.8 find\_file\_type\_and\_create\_exe.py

```

1  import subprocess
2  import os
3
4  def identify_and_rename_exe_files(folder_path, trid_path):
5      exe_files = []
6
7      for file in os.listdir(folder_path):

```



```

8         file_path = os.path.join(folder_path, file)
9         if os.path.isfile(file_path):
10             result = subprocess.run([trid_path, '-ns', file_path],
11                                     capture_output=True, text=True)
12             if 'Win32_EXE' in result.stdout:
13                 exe_files.append(file)
14                 new_file_path = os.path.splitext(file_path)[0] + ".exe"
15                 os.rename(file_path, new_file_path)
16
17     return exe_files
18
19 if __name__ == "__main__":
20     folder_path = "path/to/your/folder"
21     trid_path = "path/to/trid.exe"
22
23     exe_files = identify_and_rename_exe_files(folder_path, trid_path)
24     print("Identified_and_renamed_exe_files:", exe_files)

```

## A.9 execute\_pintool.sh

```

1  #!/bin/bash
2
3  if [ $# -eq 0 ]; then
4      echo "Usage: _$0_<argument>"
5      exit 1
6  fi
7
8  argument=$1
9
10 ssh hamud@192.178.162.2 "Path\to\python.py_$argument"

```

## A.10 ftp\_setup.sh

```

1  #!/bin/bash
2
3  #Update the os
4  sudo apt update && sudo apt upgrade
5
6  #Install FTP-Server (VSFTPD)
7  sudo apt install vsftpd
8
9  #Check vsftpd
10 systemctl status vsftpd --no-pager -l
11
12 #Add a user for FTP to use the FTP Server. Using is created via another
13 #script.
14 sudo adduser ftpuser

```

```
15 #Directory where the FTP server saves data
16 sudo mkdir /home/ftpuser/ftp
17 sudo chown nobody:nogroup /home/ftpuser/ftp
18 sudo chmod a-w /home/ftpuser/ftp
19 sudo mkdir /home/ftpuser/ftp/upload
20
21 #Give ownership to the user created before
22 sudo chown ftpuser:ftpuser /home/ftpuser/ftp/upload
23
24 #Small Demo file to test if everything worked
25 echo "My_own_FTP_Server" | sudo tee /home/ftpuser/ftp/upload/demo.txt
26
27 #Create userlist
28 echo "ftpuser" | sudo tee -a /etc/vsftpd.userlist
29 sudo ls -la /home/ftpuser/ftp
30
31 sudo cp config_files/vsftpd.conf /etc/vsftpd.conf
32
33 #Restart ftp server
34 sudo systemctl restart vsftpd
```

# Literaturverzeichnis

- [1] Mohammed N Alenezi, Haneen Alabdulrazzaq, Abdullah A Alshafer, and Mubarak M Alkharang. Evolution of malware threats and techniques: A review. *International journal of communication networks and information security*, 12(3):326–337, 2020.
- [2] Apricorn. Apricorn 2022 Global IT Security Survey. Technical report, Apricorn, 2022.
- [3] Ömer Aslan Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271, 2020.
- [4] AV-Atlas. Total Amount of Malware and PUA. Available at: <https://portal.av-atlas.org/malware> (Accessed: 31.05.2023), 2023.
- [5] Avira. Threat Analysis. Accessed: 2023-05-16, 2023.
- [6] Marie Baezner and Patrice Robin. Stuxnet. Technical report, ETH Zurich, 2017.
- [7] Arini Balakrishnan and Chloe Schulze. Code obfuscation literature survey. *CS701 Construction of Compilers*, 19:31, 2005.
- [8] Bitkom e. V. Wirtschaftsschutz 2022. Technical report, Bitkom e. V., Albrechtstraße 10, 10117 Berlin, 2022.
- [9] Pablo Bravo and Daniel F Garcia. Proactive detection of kernel-mode rootkits. In *2011 Sixth International Conference on Availability, Reliability and Security*, pages 515–520. IEEE, 2011.
- [10] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In *Proceedings of the 15th IFIP TC 6/TC 11 International Conference on Communications and Multimedia Security*, pages 63–72. Springer, 2014.
- [11] Fred Cohen. Computer viruses: theory and experiments. *Computers & security*, 6(1):22–35, 1987.
- [12] Michael Dorman. virtualbox 2.1.1. Accessed: 2023-02-01, 2020.
- [13] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):1–42, 2008.
- [14] Eleazar Eskin, Wenke Lee, and Salvatore J Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings DARPA Information Survivability Conference and Exposition II*, volume 1, pages 165–175. IEEE, 2001.
- [15] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Malware analysis and classification: A survey. *Journal of Information Security*, 2014, 2014.

- [16] Lee Garber. Melissa virus creates a new type of threat. *Computer*, 32(06):16–19, 1999.
- [17] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.
- [18] Rashid Husain and Salihu Umar Suru. An advance study on computer viruses as computer architecture.
- [19] Daisuke Inoue, Katsunari Yoshioka, Masashi Eto, Yuji Hoshizawa, and Koji Nakao. Automated malware analysis system and its sandbox for revealing malware’s internal and external activities. *IEICE transactions on information and systems*, 92(5):945–954, 2009.
- [20] Intel. Pin - A Dynamic Binary Instrumentation Tool, 2023.
- [21] Nivedita James. 81 Phishing Attack Statistics 2023: The Ultimate Insight, 2023.
- [22] Javatpoint. System Calls in Operating System (OS). Accessed: 2023-05-15, 2023.
- [23] Mateusz Jurczyk. Windows WIN32K.SYS System Call Table (NT/2000/XP/2003/Vista/2008/7/8/10). Accessed: 2023-05-15, 2023.
- [24] Kaspersky. Cybercriminals attack users with 400,000 new malicious files daily – that is 5% more than in 2021. Accessed: 2023-05-16, 2022.
- [25] E Konstantinou and S Wolthusen. Metamorphic Virus: Analysis and Detection Technical Report. Technical report, RHUL-MA-2008-02 Department of Mathematics Royal Holloway, University of London, 2008.
- [26] Vadim Kotov and Fabio Massacci. Anatomy of exploit kits: Preliminary analysis of exploit kits as software artefacts. In *Engineering Secure Software and Systems (ESSoS 2013)*, pages 181–196. Springer, 2013.
- [27] William Merrin. *Digital war: A critical introduction*. Routledge, 2018.
- [28] Microsoft. Download and install the Windows ADK. Accessed: 2023-02-04, 2022.
- [29] David Moore, Colleen Shannon, and K. Claffy. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 273–284, 2002.
- [30] Philip O’Kane, Sakir Sezer, and Domhnall Carlin. Evolution of ransomware. *Iet Networks*, 7(5):321–327, 2018.
- [31] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. Dynamic malware analysis in the modern era—A state of the art survey. *ACM Computing Surveys (CSUR)*, 52(5):1–48, 2019.
- [32] Hilarie Orman. The morris worm: A fifteen-year perspective. *IEEE Security & Privacy*, 1(5):35–43, 2003.
- [33] K Philip, S Sakir, and C Domhnall. Evolution of ransomware. *IET Netw*, 7(5):321–327, 2018.
- [34] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim. Camouflage in malware: from encryption to metamorphism. *International Journal of Computer Science and Network Security*, 12(8):74–83, 2012.

- [35] Katie Rees. The Top 4 Ways That Malware Spreads, 2022.
- [36] CQ Researcher. *Issues in terrorism and Homeland Security: Selections from CQ researcher*. Sage, 2010.
- [37] Ronny Richardson and Max M North. Ransomware: Evolution, mitigation and prevention. *International Management Review*, 13(1):10, 2017.
- [38] Sanjay K Sahay, Ashu Sharma, and Hemant Rathore. Evolution of malware and its detection techniques. In *Information and Communication Technology for Sustainable Development: Proceedings of ICT4SD 2018*, pages 139–150. Springer, 2020.
- [39] Nadeem Shah and Mohammed Farik. Ransomware-threats vulnerabilities and recommendations. *International Journal of Scientific & Technology Research*, 6(06):307–309, 2017.
- [40] Ashu Sharma and Sanjay Kumar Sahay. Evolution and detection of polymorphic and metamorphic malwares: A survey. arXiv preprint arXiv:1406.7061, 2014.
- [41] Laura Sheldon. Implementing Information Security Architecture and Governance: A Big Framework for Small Business, 2016.
- [42] SonicWall Inc. SonicWall Cyber Threat Report 2023. Technical report, SonicWall Inc., 1033 McCarthy Boulevard, Milpitas, CA 95035, 2023.
- [43] Statcounter. Operating System Market Share Worldwide. Accessed: 2023-05-17, 2023.
- [44] Clare Stouffer. What is malware + how to prevent malware attacks in 2022. Available at: <https://us.norton.com/blog/emerging-threats/malware>, 2022.
- [45] Swimlane. Automated Malware Analysis with Low-Code Security Automation. Accessed: 2023-05-16, 2023.
- [46] Rabia Tahir. A study on malware and malware detection techniques. *International Journal of Education and Management Engineering*, 8(2):20, 2018.
- [47] Viridya Tasril, Meiliyani Br Ginting, APUS Mardiana, and APU Siahaan. Threats of computer system and its prevention. *International Journal of Scientific Research in Science and Technology*, 3(6):448–451, 2017.
- [48] Yuli Vasiliev. Controlling VirtualBox from the Command Line, 2014.
- [49] VirtualBox. 6.2. Introduction to Networking Modes, 2023.
- [50] Andrew Walenstein, Rachit Mathur, Mohamed R Chouchane, and Arun Lakhotia. The design space of metamorphic malware. In *2nd International Conference on i-Warfare and Security (ICIW 2007)*, pages 241–248, 2007.
- [51] Wing Wong and Mark Stamp. Hunting for metamorphic engines. *Journal in Computer Virology*, 2:211–229, 2006.
- [52] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010.

- [53] Junjie Zhang, Christian Seifert, Jack W Stokes, and Wenke Lee. Arrow: Generating signatures to detect drive-by downloads. In *Proceedings of the 20th International Conference on World Wide Web*, pages 187–196, 2011.