Technische
Hochschule Ulm
University of
Applied Sciences

Machine Learning im Kontext von Cyber Security

Masterarbeit
zur Erlangung des Grades eines Master of Science (M.Sc.) im Studiengang Informationssysteme

vorgelegt von
Kathrin Rodi

Matrikelnummer: 3129378

3. Februar 2020

Erstgutachter: Prof. Dr. Reinhold von Schwerin

Zweitgutachter: Prof. Dr. Markus Schäffter

Betreuer: Hans-Martin Münch

Eigenständigkeitserklärung

Diese Abschlussarbeit wurde von mir selbständig verfasst. Es wurden nur die angegebenen Quellen und Hilfsmittel verwendet. Alle wörtlichen und sinngemäßen Zitate sind in dieser Arbeit als solche kenntlich gemacht.

Kathrin Rodi, 3. Februar 2020

Abstract

Machine Learning Ansätze sind im Kontext von Cyber Security essenziell, da es durch immer anspruchsvoller werdende Sicherheitsbedrohungen nicht mehr möglich ist deren Indikatoren manuell zu ermitteln und zu klassifizieren. Diese Aufgabe von Menschen bearbeiten zu lassen wäre deutlich zu kostenintensiv und ineffizient. Anhand einer Literaturrecherche nach Webster und Watson (2002) wird überprüft welchen Mehrwert Machine Learning in Bezug auf Informationssicherheit bieten kann. Dazu werden bestehende Ansätze sowohl aus der Industrie als auch aus der Wissenschaft klassifiziert, wobei die jeweils verwendeten Algorithmen, Features, Evaluationskriterien sowie die durchgeführte Evaluation der jeweiligen Ergebnisse untersucht werden.
Des Weiteren wird der Begriff Indicator of Compromise (IoC) geklärt und besonders auf dessen Bedeutung, in Bezug auf Malware Erkennung, eingegangen.
Zusätzlich wird untersucht welche Datensätze, in Bezug auf Cyber Security, bestehen und welche Qualität diese aufweisen.
Ergänzend werden Ansätze aus der Industrie überprüft welche bereits wissenschaftlich untersucht wurden um herauszufinden, welche Ansätze in der Industrie momentan besonders gefragt sind.
Ziel der Arbeit ist es eine umfassende Übersicht über bestehende Machine Learning Ansätze im Bereich Informationssicherheit zu gewinnen. Zudem wird einer der gefundenen, qualifizierten Datensätze wissenschaftlich validiert.

Inhaltsverzeichnis

Inhaltsverzeichnis I

Abkürzungsverzeichnis IV

Abbildungsverzeichnis XI

Tabellenverzeichnis XIII

1. Einleitung 1

1.1. Motivation 1

1.2. Ziele der Arbeit 3

1.3. Aufbau der Arbeit 4

2. Forschungsmethoden 5

2.1. Literaturrecherche 5

2.2. CRISP-DM 6

3. Erkennung von Schadcode/IOCs 9

4. Analyseverfahren 14

4.1. Grundbegriffe aus der Cyber Security 14

4.2. Grundbegriffe des maschinellen Lernens 18

4.3. Angewandte Ansätze 27

4.3.1. Erkennung von Malware - Hybride Analyse (2015) 27

4.3.2. Erkennung von Malware - Statische Analyse (2016) 28

4.3.3. Erkennung bösartiger XML-basierter Office Dokumente(2016) 28
4.3.4. Erkennung bösartiger HTTP-Anfragen (2016) 30
4.3.5. Klassifizierung von Netzwerkattacken (2017) 30
4.3.6. Erkennung von Ransomware - Dynamische Analyse (2017) . 31
4.3.7. Erkennung von Malware - Imageanalyse (2017) 31
4.3.8. Erkennung böswilliger MS Office Dateien (2017) 32
4.3.9. Klassifizierung von DDoS Attacken (2017) 32
4.3.10. Schwachstellen Scanner für Web Applikationen (2017) 33
4.3.11. Erkennung von Malware anhand von PE-Header (2017) . . . 34
4.3.12. Erkennung von Malware anhand von PE-Header mit erweitertem Feature-Set (2017) 34
4.3.13. Erkennung von Exfiltration und C&C Tunnels (2017) 35
4.3.14. Erkennung bösartiger PowerShell-Befehle (2018) 35
4.3.15. Klassifizierung von Netzwerkverkehr in 5 Klassen (2018) . . 36
4.3.16. Anomalieerkennung anhand von Systemprotokollen (2018) . 36
4.3.17. Erkennung von bösartigem Netzwerkverkehr (2018) 37
4.3.18. Erkennung von Botnetzen (2018) 38
4.3.19. Klassifizierung von Microsoft Malware (2018) 38
4.3.20. Klassifizierung von Malware anhand von Datenpaketen (2018) 39
4.3.21. Erkennung von Port-Scans (2018) 39
4.3.22. Erkennung von Netzwerkverkehr (2018) 40
4.3.23. Erkennung bösartiger SQL-Abfragen (2018) 40
4.3.24. Erkennung von LDDoS Attacken (2018) 41
4.3.25. Klassifizierung von Wi-Fi Netzwerkdaten (2018) 41
4.3.26. Klassifizierung von verschleierter Malware (2019) 42
4.3.27. Klassifizierung von Malware - Imageanalyse (2019) 43
4.3.28. Erkennung von FF Netzwerken (2019) 43
4.3.29. Erkennung von drive-by Download-Attacken bei Twitter (2019) 44

4.3.30. Erkennung von DGA Domains (2019) 45
4.3.31. Erkennung von Phishing Websites (2019) 46
4.3.32. Erkennung von Insider Bedrohungen (2019) 47
4.3.33. Erkennung von bösartigen PDFs (2019) 48
4.4. Ergebnisse der Untersuchung der Analyseverfahren 49

5. Datensätze **51**
5.1. HTTP Datenset CSIC 2010 52
5.2. NSL-KDD 52
5.3. LANL 53
5.4. ISCX 53
5.5. CTU-13 54
5.6. Microsoft Malware Classification Challenge (BIG 2015) 55
5.7. CICIDS2017 55
5.8. CIC 56
5.9. ASNM-NPBO 57
5.10. CERT 58
5.11. Ember 59
5.12. Evaluation der Datensätze 61

6. Prototypische Implementierung **65**

7. Zusammenfassung und Ausblick **75**

8. Literatur **I**

A. Anhang **IX**

Abkürzungsverzeichnis

AB AdaBoost 29

ANN Artificial Neural Network 30

ANNs Artificial Neural Networks 37

ACK acknowledge 17

APTs Advanced Persistent Threats 3

ARFF Attribute-Relation File Format 28

ASNM-NPBO Advanced Security Network Metrics & Non-Payload-Based Obfuscations 40

AUC Area Under the Curve 34

AWID Aegean WiFi Intrusion Dataset 41

BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V 2

BKA Bundeskriminalamt 1

BN Bayesian Network 29

BTB Bug Terminating Bot 33

C&C Command & Control 35

CDNs Content Distribution Networks 43

CNN Convolutional Neural Network 31

CNNs Convolutional Neural Networks 24

CRISP-DM Cross-Industry Standard Process for Data Mining 5

CSV Comma-Separated Values 65

C2 Command & Control 12

DBN Deep Belief Network 36

DPI Deep Packet Inspection 39

DoS Denial of Service 17

DDoS Distributed Denial of Service 17

DGA Domain Generation Algorithm 45

DLL Dynamic Link Library 42

DLLs Dynamic Link Libraries 10

DNN Deep Neural Network 41

DNNs Deep Neural Networks 43

DNS Domain Name System 32

DRBMs Deep Restricted Boltzmann Machines 37

DT Decision Tree 19

ERP Enterprise-Resource-Planning 47

ET Extra Random Trees 34

FC Fuzzy Classifier 38

FC Neural Network Fully Connected Neural Network 34

FF Fast-Flux 43

FNR False Negative Rate 37

FPR False Positive Rate 25

FTP File Transfer Protocol 53

GBT Gradient Boosting Tree 46

GMM-EM Gaussian-Mixture Model for Expectation-Maximization 33

GUI Graphical User Interface 11

GPU Graphics Processing Unit 31

GPUs Graphics Processing Units

GRU Gated Recurrent Units 76

HITB Hack In The Box 48

HTTP Hypertext Transfer Protocol 2

HTTPS Hypertext Transfer Protocol Secure 53

IBk Instance-Based k 28

ICMP Internet Control Message Protocol 32

IDS Intrusion Detection System 30

IMAP Internet Message Access Protocol 53

IoC Indicator of Compromise C

IoCs Indicators of Compromise 6

IoT Internet of Things 5

IRC Internet Relay Chat 18

JSON JavaScript Object Notation 59

k-NN k-Nearest Neighbors 19

LANL Los Alamos National Laboratory 37

LB LogitBoost 29

LDAP Lightweight Directory Access Protocol 37

LDDoS Low-rate DDoS 17

LDA Linear Discriminant Analysis 35

LightGBM Light Gradient Boosting Machine 43

LIEF Library to Instrument Executable Formats 59

LR Logistic Regression 19

LSTM Long-Short Term Memory XI

MLAs Machine Learning Algorithmen 3

MLP Multi-Layer Perceptron 30

MMCC Microsoft Malware Classification Challenge 63

NB Naïve Bayes 29

NBTree Naïve Bayesian Tree 30

NLP Natural Language Processing 24

Opcode Operation Code 28

PE-Datei Portable Executable Datei 59

PE-Dateien Portable Executable Dateien 10

POP3 Post Office Protocol v. 3 53

RAM Random-Access Memory 12

RBM Restricted Boltzmann Machine 37

RCE Remote Code Execution 18

RF Random Forest 19

RNN Recurrent Neural Network 24

RNNs Recurrent Neural Networks 30

ROC AUC Receiver Operator Characteristic Area Under Curve 25

R2L Remote to Local 16

SGDClassifier Stochastic Gradient Descent Classifier 30

SMO Sequential Minimal Optimization 29

SMTP Simple Mail Transfer Protocol 53

SSH Secure Shell 32

SVM Support Vector Machine XI

SVMs Support Vector Machines 42

SYN synchronize 17

TCP Transmission Control Protocol 17

TF-IDF Term Frequency - Inverse Document Frequency 29

TPR True Positive Rate 25

TPU Tensor Processing Unit 66

UDP User Datagram Protocol 38

U2R User to Root 17

VBA Visual Basic for Applications 17

XGBoost Extreme Gradient Boosting 42

XSS Cross-Site Scripting 16

Abbildungsverzeichnis

2.2.1. CRISP-DM Phasen (SmartVisionEurope 2015) 7

3.0.1. Aufbau einer PE-Datei (eigene Darstellung) 11

4.1.1. Schutzziele der Informationssicherheit (eigene Darstellung) 15

4.2.1. Entscheidungsgrenze (eigene Darstellung) 20

4.2.2. Decision Tree (Joshua Saxe 2018) 21

4.2.3. Entscheidungsgrenze und Support Vektoren der Support Vector Machine (SVM)
(C. N. Nguyen und Zeigermann 2018) 22

4.2.4. Aufbau eines Neurons nach Joshua Saxe (2018) (eigene Darstellung) 23

4.2.5. Beispielhafte ROC Kurven (Molin 2019) 25

4.3.1. Features als Pfade dargestellt (Cohen u. a. 2016) 29

4.4.1. Verteilung der Analyseverfahren nach Themen (eigene Darstellung) 49

5.9.1. Experimentelle Verschleierungstechniken mit Parametern und IDs (Homoliak u. a. 2019) 57

5.11.1 Rohe Features, die aus einer einzelnen PE-Datei extrahiert wurden (Anderson und Roth 2018) 61

6.0.1. Struktur einer Long-Short Term Memory (LSTM) Zelle (eigene Darstellung in Anlehnung an Raschka und Mirjalili (2019)) 68

6.0.2. Loss nach Epochen für Trainings- und Testdaten (eigene Darstellung) 70

6.0.3. Genauigkeit und loss `modelRnn1` (eigene Darstellung) 71

6.0.4. Genauigkeit und loss `modelRnn2` (eigene Darstellung) 72

6.0.5. Genauigkeit und loss `modelRnn3` (eigene Darstellung) 73

6.0.6.Klassifizierung von unbekannten Testdaten (eigene Darstellung) . . 74

Tabellenverzeichnis

3.1. Auszug von Indikatoren für einen Malware Angriff (in Anlehnung an Sikorski (2012)) 13

5.1. Evaluierung ausgewählter Datensätze 63

6.1. Ergebnisse der getesteten Modelle 73

1. Einleitung

Bereits im 19. Jahrhundert träumt der Polymath Charles Babbage vom mechanisierten Rechnen. Dieser Wunsch basierte hauptsächlich auf dem Zorn über die Unzulänglichkeit der damaligen analogen, mathematischen Anwendungen. Babbage entwickelte ein Konzept für analytische Maschinen, also einen programmierbaren Allzweckrechner. Seine Kollegin, die britische Mathematikerin, Ada Lovelace lieferte die entsprechenden Ideen zur Programmierung seiner Maschine. Allerdings konnte das Konzept der *Analytical Engine* niemals umgesetzt werden und besteht seither, rein als Entwurf. Dennoch macht diese Forschung die beiden bis heute zu Pionieren des modernen Computers und dessen Programmierung (Wilkes und Spatz 1995). Babbages Wunschtraum von damals ist nicht nur längst Wirklichkeit geworden, er hat sich in rasendem Tempo weiterentwickelt. Heute können Computer nicht nur fehlerfrei Logarithmen berechnen, sie sind bereits in der Lage einen Großteil unseres Lebens zu digitalisieren. Bankgeschäfte, Einkäufe, die Steuererklärung und bald auch Arztbesuche sind nur ein kleiner Teil dessen, was wir online erledigen. Dabei produzieren wir eine enorme Masse an persönlichen Daten, welche in falschen Händen eine Gefahr für uns darstellt. Durch den Diebstahl unserer Kreditkartendaten können Angreifer beispielsweise auf unsere Finanzen zugreifen. Gerade deshalb gilt es diesen Teil unseres Lebens zu schützen. Wie wir unsere physischen Habseligkeiten schützen in dem wir beispielsweise Schlösser verwenden, gilt es ebenso unsere digitalen Artefakte zu schützen um finanziellen, reputativen sowie physischen Schaden zu verhindern. Studien zeigen allerdings, dass wir der nötigen Sicherheit weit hinterherhinken (Microsoft 2019).

1.1. Motivation

> *Cybercrime umfasst die Straftaten, die sich gegen Datennetze, informationstechnische Systeme oder deren Daten richten [...] oder die mittels Informationstechnik begangen werden. (Bundeskriminalamt 2018)*

Das Bundeskriminalamt (BKA) verzeichnete allein im Jahr 2017 knapp 86.000 Fälle von Cybercrime. Davon waren über 1.400 Phishing Angriffe im Onlinebanking bei

denen ein durchschnittlicher Schaden von 4000€ pro Fall entstand (Bundeskriminalamt 2018). Laut einer Studie des Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM) ist bereits, jeder zweite Deutsche Opfer eines Cyberangriffs geworden. Lediglich 18% hätten diesbezüglich angegeben Anzeige bei der Polizei erstattet zu haben (e.V. 2017). Dies lässt vermuten, dass die Dunkelziffer der tatsächlichen Cybercrime Straftaten weit über den 86.000 gemeldeten Fällen liegt. AVTest registriert täglich bis zu 350.000 neue schadhafte Programme (AV-TEST 2019). McAfee (2019) verzeichnete im 1. Quartal 2019 die höchste Anzahl an neuer Malware seit Jahren. Das Problem hierbei ist nicht allein die Quantität der Software, sondern auch die Qualität. Immer bessere Verschleierungstaktiken sorgen dafür, dass Malware schwerer identifiziert werden kann (P. He u. a. 2017). Sicherheitsüberprüfungen die auf Signaturabgleichen beruhen, funktionieren beispielsweise nur bei bereits bekannten Signaturen, neuartige Malware kann von ihnen nicht erkannt werden. Diese Komplexität und Fülle an Malware überfordert nicht nur Intrusion Detection Systeme, sondern auch Sicherheitsexperten. Wie schon von Evans und Reeder (2010) vorhergesagt, fehlt es an Expertise für diese Flut an Angriffen. Da der Mangel an Fachkräften erst in Jahren ausgeglichen werden kann, bedarf es alternativer Lösungen für die Sicherheit von Heute. Zudem sind herkömmliche Lösungen wie das manuelle Überprüfen von verdächtigem Code, Reverse Engineering oder das manuelle Erkennen von Sicherheitslücken zu langsam, um mit Konflikten adäquat umzugehen (Singla und Bertino 2019).

Machine Learning kann der Schlüssel hierfür sein. Diese Technologie kann für die automatische Verarbeitung von Sicherheitsereignissen genutzt werden (Singla und Bertino 2019). Gängige Warnungen können leicht von Machine Learning Verfahren überprüft werden. Dadurch haben Sicherheitsexperten mehr Kapazität sich um besondere Warnungen zu kümmern. Des Weiteren ist es schwierig Warnsignale zusehends zu priorisieren und zu kategorisieren (Joshua Saxe 2018). Auch hierbei können Algorithmen helfen. Beispielsweise lässt sich ein System implementieren, welches eine Klassifizierung in gutartig oder bösartig durchführt. Dabei spricht man von einer *binären* Klassifikation. Gleichzeitig ist es möglich, die als bösartig gelabelten Daten in diverse Kategorien einzustufen. Beispielsweise kann Malware, durch *Multi-Klassen Klassifikation*, in Subklassen wie Viren, Würmer, Trojaner und Ransomware aufgeteilt werden. Dadurch kann die spezifische Untersuchung und Bekämpfung effizienter gestaltet werden. Eine weitere Fähigkeit von Machine Learning ist das *Clustering*. Diese Technik fasst grundsätzlich ähnliche Inhalte zusammen. Dabei entstehen Gruppen mit Daten die eine hohe interne Homogenität, verglichen mit anderen Gruppen jedoch eine hohe Heterogenität aufweisen. Clustering kann unter anderem dazu genutzt werden, Hypertext Transfer Protocol (HTTP) Verkehr zu analysieren und herauszufinden, um welche Art von Anfragen es sich

handelt. Die Requests können beispielsweise zu Botnet-, Mobiltelefon- oder gängigen Benutzeranfragen geclustert werden. Dies stellt eine immense Erleichterung für Sicherheitsexperten dar, da sich diese unmittelbar dem potenziell gefährlichen Cluster widmen können (Joshua Saxe 2018).
Machine Learning Algorithmen (MLAs) besitzen die Fähigkeit des eigenständigen Lernens. Da sie so zu neuen Erkenntnissen gelangen und nicht auf bereits bekannte Warnungen, wie beispielsweise bösartige Signaturen, angewiesen sind, können mit ihrer Hilfe sowohl Advanced Persistent Threats (APTs), als auch Zero Days erkannt werden. Anhand der dadurch verringerten Antwortzeit auf Attacken, kann nicht nur ein Verlust von Daten, sondern auch ein finanzieller Schaden abgemildert werden (Hu u. a. 2019).

Forschungen belegen die Wirksamkeit von Machine Learning Ansätzen im Bereich Cyber Security und somit die hier aufgeführten Thesen von beispielsweise Homoliak u. a. (2019),Jeong, Woo und Kang (2019), Sabar, Yi und Song (2018), Brown u. a. (2018) und Yin u. a. (2017). Um dies zu verdeutlichen werden adäquate Untersuchungen in Kapitel 4 beschrieben.

1.2. Ziele der Arbeit

Die Ziele dieser Arbeit belaufen sich auf die folgenden vier Punkte:

1. Zunächst soll der Begriff *Indicator of Compromise* geklärt und in Bezug auf Malware untersucht werden.
2. Des Weiteren soll der momentane Stand der Forschung im Bereich Cyber Security, mit Hilfe von Machine Learning Verfahren, erörtert und durch etwaige Ansätze aus der Industrie erweitert werden.
3. Um eine aussagekräftige Analyse zu tätigen bedarf es qualitativ hochwertiger Datensätze. Um einen solchen Datensatz zu ermitteln, gilt es zu evaluieren welche Datensätze einer Analyse dienlich sind.
4. Ferner soll eine prototypische Umsetzung einer Analyse mit einem der evaluierten Datensätze durchgeführt werden. Dadurch soll getestet werden, ob Machine Learnning einen tatsächlichen Mehrwert im Bereich Cyber Security bieten kann.

Die Arbeit soll somit sowohl Sicherheitsexperten als auch Data Scientists, einen Überblick über den momentanen Stand der Forschung liefern. Zudem soll es diesem Publikum durch die Evaluierung der Datensätze erleichtert werden eigenständige,

neue Analysen durchzuführen oder bestehende Analyseverfahren zu optimieren. Die prototypische Implementierung soll diesbezüglich als Beispiel dienen.

1.3. Aufbau der Arbeit

Das erste Kapitel dient der Einführung in das Thema, wobei zusätzlich die Relevanz der Forschung erläutert wird. Ferner werden die Ziele der Arbeit abgesteckt. Das folgende Kapitel 2 erläutert das fundierte Vorgehen, durch welches Informationen generiert und Erkenntnisse erlangt werden. Der Hauptteil besteht aus drei Teilen: der Untersuchung der bestehenden Analyseverfahren, der Evaluierung der Datensätze, sowie der prototypischen Implementierung eines Analyseverfahrens anhand eines der evaluierten Datensätze. Im Anschluss wird zu den Ergebnissen kritisch Stellung genommen, sowie ein Fazit gezogen. Abschließend wird ein Ausblick für zukünftige Forschungen gegeben.

2. Forschungsmethoden

In diesem Kapitel werden die Forschungsmethoden erläutert, auf welcher der Informationsgewinn basiert. Die Literaturrecherche wurde zu Beginn der Arbeit durchgeführt, um Informationen bezüglich des Themas zu sammeln, sowie den momentanen Stand der Forschung zu identifizieren.
Der Cross-Industry Standard Process for Data Mining (CRISP-DM) wird als Vorgehensmodell ausgewählt, da hierdurch eine strukturierte Vorgehensweise sichergestellt werden kann. Der genaue Aufbau dieses Prozesses wird in Kapitel 2.2 beschrieben.

2.1. Literaturrecherche

Im Rahmen einer Literaturrecherche nach Webster und Watson (2002) wurden die wissenschaftlichen Datenbanken ACM Digital Library, ScienceDirect und IEEE sowie die akademische Suchmaschine Google Scholar nach relevanten Inhalten durchsucht. Hierbei wurde darauf geachtet, dass es sich bei den Ergebnissen um peer-reviewed Journals sowie peer-reviewed Konferenzen handelt, um eine bestmögliche Qualität der zu verwendenden Quellen zu garantieren. Ferner wurde lediglich nach Publikationen ab 2015 gesucht, um die Aktualität der Ansätze zu gewährleisten. Da sich besonders im Bereich Cyber Security binnen eines Jahres enorme Entwicklungen zeigen, wäre durch das Hinzuziehen älterer Publikationen kein Mehrwert entstanden. Als Suchstring wurde die logische Kombination aus den Begriffen „Machine Learning“ OR „Deep Learning“ AND „Cyber Security“ OR „Information Security“ NOT „Android“ NOT „IoT“ NOT „Mobile“ verwendet. Dies beruht darauf, dass Machine Learning und Deep Learning, sowie Information Security und Cyber Security oftmals synonym verwendet werden. Da sich die Arbeit nicht mit dem Thema mobil- oder Internet of Things (IoT)- basierter Applikationen beschäftigt, wurden diese Keywords bei der Suche ausgegrenzt. Eine Forschung in diesem Bereich ist gleichermaßen umfangreich und bedarf einer eigenständigen Arbeit.
Die Suche ergab insgesamt 308 Treffer. Zusätzlich wurde sowohl eine Vorwärts - als auch eine Rückwärtssuche durchgeführt, welche zu weiteren 24 Treffern führte. Durch die Rückwärtssuche konnten weitere relevante Ansätze von Machine Learning im Bereich Cyber Security, sowie hilfreiche Informationen zu bestehenden Datensets

ausfindig gemacht werden. Auch die Vorwärtssuche, welche mit Google Scholar umgesetzt wurde, führte zu hochaktuellen Beiträgen zum Thema.
Insgesamt wurden 332 Quellen ausfindig gemacht. Anhand Titel, Abstract, Einleitung und Schluss wurden 266 Quellen in Ermangelung von Relevanz oder wegen Überschneidungen mit bereits gefundenen Ansätzen aussortiert. Hingegen wurden 66 Quellen für die hier vorliegende Arbeit verwendet. Eine Übersicht über den Prozess der Literaturrecherche kann in Anlage 1 eingesehen werden. Wie von Webster und Watson (2002) empfohlen, wurden die gefundenen Quellen anschließend akribisch in einer Liste nach Inhalt und Relevanz gefiltert. Zunächst wurden die ausgewählten Quellen in vier Themenblöcke aufgeteilt:

- Ansatz inklusive Datenset
- Ansatz ohne Datenset
- Indicators of Compromise (IoCs)
- Datensatz

Die Quellen wurden anschließend den einzelnen Blocks zugewiesen. Zudem wurden weitere Blocks erstellt, die jedoch keinen Einfluss auf die Relevanz der Quelle hatten und somit hier nicht gelistet sind. Zu jeder Quelle wurde die Kernaussage, sowie inhaltsrelevante Punkte, wie verwendete Machine Learning Verfahren, Namen von Datensets oder interessante Ergebnisse notiert. Anschließend wurde die Relevanz der Quellen untersucht. Um diesbezüglich ein systematisches Vorgehen zu garantieren wurden folgende Relevanzkriterien erstellt:

1. Hoher Themenbezug zu mindestens einem der Themen: IoCs oder Datensets
2. Ausführung eines Ansatzes
3. Ausführung eines Ansatzes inklusive verfügbarem Datenset

Die Quellen wurden anhand dieser Skala bewertet, wobei 1 für eine geringe Relevanz und 3 für eine hohe Relevanz steht. Zusätzlich wurde die Anzahl der Zitationen festgehalten, um die wissenschaftliche Relevanz innerhalb der Forschungsgemeinde zu evaluieren. Einen Ausschnitt der daraus resultierenden Literaturliste kann in Anlage 2 eingesehen werden.

2.2. CRISP-DM

Bereits im 18. Jahrhundert legte Thomas Bayes mit seinem *Satz von Bayes*, der die Berechnung bedingter Wahrscheinlichkeiten beschreibt, den Grundstein dafür was

wir heute *Data Mining*, also den Erkenntnisgewinn aus Daten, nennen. Als in den 1950er Jahren die Produktion kommerzieller Seriencomputer startete, konnte die Datenanalyse automatisiert werden. Daraus entwickelten sich die ersten neuronalen Netze und Cluster Analysen, wie wir sie heute kennen. Einen weiteren Aufschwung erlebte Data Mining in den 1990ern, wo auch der CRISP-DM von DaimlerChrysler, SPSS und NCR entwickelt wurde (SmartVisionEurope 2015). Dieser Prozess beschreibt eine Methodik für Data Scientists, um eine effiziente, robuste und universelle Vorgehensweise zu garantieren (Chapman u. a. 1999). Wie in Abbildung 2.2.1 dargestellt, besteht dieses Vorgehensmodell aus sechs Phasen.

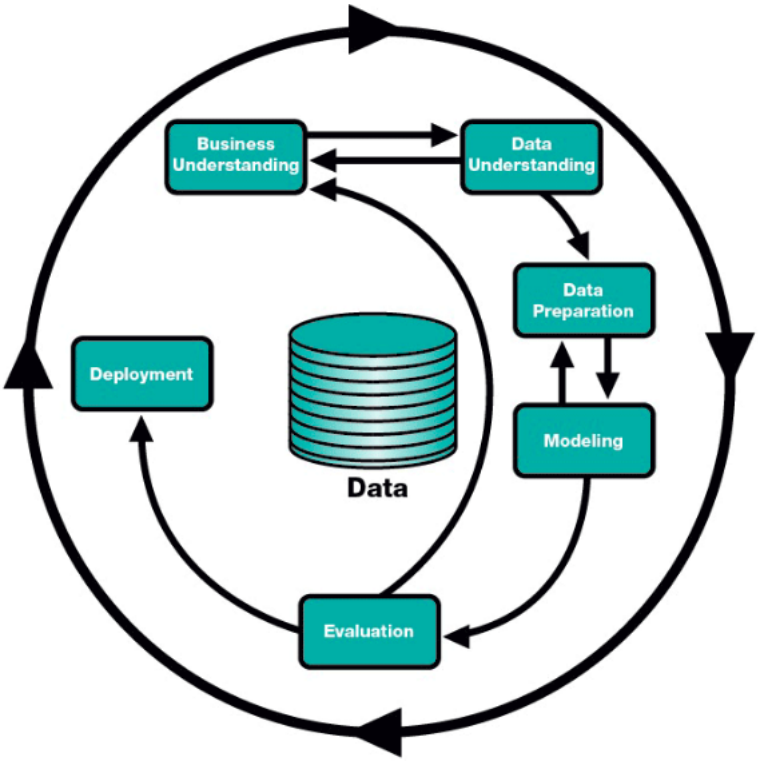


Abbildung 2.2.1.: CRISP-DM Phasen (SmartVisionEurope 2015)

Business Understanding: diese Phase beschäftigt sich mit der Frage nach dem Ziel der Analyse. Dementsprechend, werden die Aufgaben erstellt und ein Plan festgelegt.
Data Understanding: die zweite Phase zielt darauf ab Daten zu sammeln und durch ein erstes Screening, deren Qualität festzustellen. Wie die Grafik 2.2.1 zeigt, kann dies dazu führen, dass die Ergebnisse aus Phase eins, nochmals angepasst werden müssen.
Data Preparation: nachdem Daten gesammelt wurden, gilt es anschließend diese

für Analysen aufzubereiten. Hierbei liegt der Fokus darauf, die bestmögliche Konstruktion des finalen Datensatzes für die anschließende Modellierung zu gewinnen. Dazu ist es nötig, relevante Daten auszuwählen und die Daten zu bereinigen. Dazu gehört sowohl das Entfernen und Korrigieren von Datenfehlern, als auch das Schätzen fehlender Daten durch Interpolation.
Modeling: diese Phase beschäftigt sich zunächst mit der Erstellung verschiedener Modelle, wie zum Beispiel eines Decision Trees oder eines neuronalen Netzes und der anschließenden Auswahl der adäquatesten Modellierungstechnik. Dazu gehört das Kreieren eines Test- und eines Trainingsdatensets, womit verschiedene Modelle getestet werden können. Gegebenenfalls bedarf dies dem Wiederholen der dritten Phase, um ein Datenset nochmals zu justieren.
Evaluation: während dieser Phase wird das Modell, welches die in Phase eins definierten Ziele am besten erfüllt, ausgewählt.
Deployment: in der letzten Phase werden die Ergebnisse aufbereitet, präsentiert und zusätzlich in einem Dokument festgehalten (SmartVisionEurope 2015).

Dieses Vorgehensmodell wurde ausgewählt, da es ein strukturiertes Vorgehen ermöglicht und dadurch die Qualität der Ergebnisse gesteigert werden kann. Das *Business Understanding* besteht in dieser Arbeit darin, herauszufinden, was der momentane Stand der Forschung bezüglich Machine Learnning im Bereich Cyber Security ist. Anschließend werden bestehende Datensets untersucht, die der späteren Analyse dienen. Um diese Daten anwenden zu können werden diese zunächst in der *Data Preparation* Phase entsprechend aufbereitet. In der nächsten Phase, dem *Modeling* werden diverse Modelle auf deren Passgenauigkeit überprüft. Anschließend wird das Modell, welches die Anforderungen am besten erfüllt, implementiert. Darüber hinaus wird das Vorgehen ausführlich dokumentiert.

3. Erkennung von Schadcode/IOCs

Die permanente Steigerung in Größe und Komplexität von Computersystemen, bietet nicht nur einen höheren Nutzen für Kunden, sondern auch mehr Angriffsfläche für Hacker. Dies erschwert die Arbeit von Sicherheitsexperten. Da es darum geht die Kompromittierung eines Systems so früh wie möglich zu erkennen, um potenziellen Schaden zu verhindern, beziehungsweise diesen so gering wie möglich zu halten, arbeiten Experten gegen die Zeit.
Wurde ein System Opfer eines Angriffs, gilt es dieses forensisch zu untersuchen. Normalerweise hinterlässt ein Angreifer Spuren seines Einbruchs. Die Aufgabe der IT Security ist es, diese zu finden. Diese Hinterlassenschaften werden als *Indicator of Compromise (IoC)* bezeichnet, also Indikatoren, welche darauf hindeuten, dass ein System kompromittiert wurde.
IoCs müssen jedoch differenziert betrachtet werden. Es gibt eindeutige Indikatoren, welche kaum einen Zweifel daran lassen, dass ein System kompromittiert wurde. Wenn ein Sicherheitsexperte beispielsweise Schadsoftware auf einem System findet und feststellt, dass es zu einem Datenupload auf einen nicht identifizierbaren Server kam, kann davon ausgegangen werden, dass das System tatsächlich kompromittiert wurde. Der Indikator bildet sich hierbei aus den beiden Indizien: Malware und unautorisierter Upload.
Des Weiteren gibt es Indikatoren welche nicht eindeutig sind. Angenommen auf einem Computer werden Prozesse erkannt, welche nicht von diesem selbst gestartet wurden, sondern durch remote gesendete Befehle, könnten diese durch das Windows Tool `PsExec` übermittelt worden sein. Mit Hilfe dessen, lassen sich administrative Tätigkeiten, wie beispielsweise Systemupdates oder Passwort Änderungen, anhand von remote gesendeten Befehlen, durchführen. So vorteilhaft dieses Tool in den richtigen Händen erscheint, so gefährlich ist es in den falschen. Angreifer können `PsExec` für bösartige Zwecke missbrauchen. Zwar verlangt der remote Zugriff eine IP-Adresse mit korrespondierenden Benutzerinformationen, diese können jedoch durch andere Arten von Angriffen beschafft werden. Da es sich bei `PsExec` um ein legitimiertes Tool zur Systemkoordination handelt, wird es von Anti-Viren Programmen nicht erkannt. Dadurch wird die Entdeckung eines Missbrauchs deutlich erschwert. Da die Benutzerinformationen allerdings unverschlüsselt übertragen werden, können immerhin diese über Tools wie `Wireshark` oder `Tcpdump` abgefangen werden. Der Nachweis über die Nutzung dieses Tools allein reicht also nicht aus,

um eine Kompromittierung annehmen zu können.

Bei der Malware spezifischen Analyse gilt es zunächst herauszufinden, was genau passiert ist und welches Schadprogramm für den Angriff verantwortlich ist. Traditionelle Anti-Virus Programme arbeiten basierend auf Datenbanken, in welchen sie bereits bekannte Signaturen und Heuristiken anwenden. Das Problem hierbei ist, dass es für Angreifer ein Leichtes ist, ihren Code zu modifizieren, um die Signatur zu verändern, wodurch das Schadprogramm nicht mehr als solches erkannt wird. Verschleierungstaktiken wie diese, lassen sich in drei Gruppen einteilen (P. He u. a. 2017):

- **Packing**
  Dies bezeichnet die Technik exekutierbare Dateien zu komprimieren. Um die komprimierte Malware zu erkennen muss diese zunächst entpackt werden. Gleichzeitig ist dies aber auch ein guter IoC, da ausführbare Dateien im Regelfall nicht komprimiert vorliegen.
- **Metamorphismus**
  Hierbei wird die Erkennung erschwert in dem der Binärcode mutiert wird. Das bedeutet, die Sequenz der Opcodes wird bei jeder Ausführung geändert.
- **Polymorphismus**
  Eine polymorphe Schadsoftware generiert bei jeder Ausführung eine weitere Version der Malware, sodass eine große Anzahl an divergierender Signaturen für dasselbe Programm entstehen.

Diese Techniken erschweren das Erkennen von Malware anhand gängiger Anti-Virus Programme deutlich. Zukünftig kann Machine Learning hierbei eine große Rolle spielen. Denn wie Han u. a. (2019) bereits erfolgreich untersuchten, ist auf MLAs basierende Erkennungssoftware in der Lage, Malware trotz dieser Verschleierungstechniken zu erkennen.

Das Erkennen von Malware basiert im Regelfall auf der Untersuchung von Portable Executable Dateien (PE-Dateien). Diese beinhalten ausführbare Daten im Binärformat. Dazu gehören Windows `.exe` Dateien, Objektcode und Dynamic Link Libraries (DLLs). Eine PE-Datei, ist wie in Abbildung 4.3.1 ersichtlich, aufgebaut:

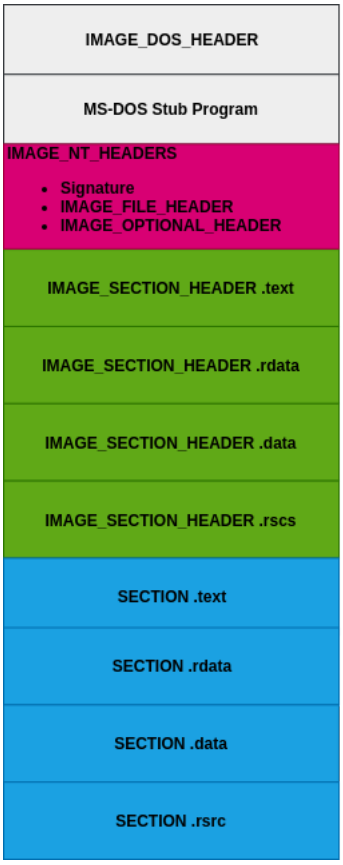


Abbildung 3.0.1.: Aufbau einer PE-Datei (eigene Darstellung)

Die in Abbildung 4.3.1 grau hinterlegten Bereiche sind für die Analyse von PE-Dateien irrelevant, sie dienen unter anderem lediglich dazu, eine Fehlermeldung auszugeben falls eine `.exe` Datei in einem Betriebssystem ausgeführt werden soll, mit welchem diese nicht kompatibel ist.
Der Bereich `IMAGE_NT_HEADERS` bietet bereits Informationen für eine Analyse. `IMAGE_FILE_HEADER` beinhaltet grundlegende Informationen bezüglich der Datei. Beispielsweise wann diese ausgeführt wurde, was für eine Analyse sehr nützlich sein kann. Der Sektor `IMAGE_OPTIONAL_HEADER` ist entgegen dem was der Name vermuten lässt nicht optional. Hier werden wichtige Informationen wie der Programmeinstiegspunkt, die Stackgröße zu Beginn sowie die Verwendung eines Graphical User Interface (GUI) oder einer Konsole definiert.
Die grün hinterlegten `IMAGE_SECTION_HEADER` bieten die interessantesten Informationen für eine Analyse. Diese *Header* werden vom Compiler generiert und benannt, sodass der Benutzer wenig Kontrolle über die Namen hat. Dementsprechend konsistent ist die Benennung im Regelfall. Im PE-Header finden sich also relevante Informationen wie Imports, Exports, die Namen der verschiedenen Bereiche (blau

hinterlegt), sowie deren Speichergröße auf der Festplatte und im Random-Access Memory (RAM), sowie die Ressourcen welche von einem Programm benötigt werden.

Grundsätzlich gibt es zwei Methoden um eine Malware Analyse durchzuführen: eine dynamische und eine statische.
Die dynamische Analyse beinhaltet das Ausführen schadhafter Programme. Dabei wird Malware in einer sicheren Umgebung ausgeführt und so deren Verhalten analysiert. Dadurch kann im Gegensatz zur statischen Analyse die tatsächliche Verhaltensweise einer Datei untersucht werden, denn nicht jede Zeichenkette die in einer Binärdatei gefunden wird, muss zwangsläufig ausgeführt werden. Zudem können Logdateien analysiert werden, welche erst durch das Ausführen eines Programms entstehen. Dynamische Analysen werden im Regelfall in einer *Sandbox*, also in einem isolierten Bereich durchgeführt, wodurch kein Schaden am System genommen wird. Der Nachteil dieser Analyse besteht darin, dass die Malware die virtuelle Umgebung erkennen kann und sich somit stoppt. Des Weiteren können von der Schadsoftware benötigte Registry Keys oder Dateien in der virtuellen Umgebung fehlen, sodass deren Verhalten nicht korrekt aufgezeichnet werden kann (Sikorski 2012).

Bei der statischen Analyse werden hingegen PE-Dateien erforscht ohne die Datei tatsächlich auszuführen. Zunächst durchläuft potenzielle Malware diverse Virenscanner, um die Entdeckung einer bösartigen Signatur zu erhöhen.
Zusätzlich kann Hashing zum Einsatz kommen. Dabei wird ein eindeutiger Hash generiert, welcher verwendet werden kann, um zu recherchieren, ob dieser bereits von anderen Antivirus-Dienstleistern analysiert wurde. Hashing bietet zudem den Vorteil, dass die Datei selbst noch nicht geteilt werden muss. Des Weiteren ist der Austausch eines Hashes auch um einiges schneller als der Upload einer Datei.
Programme verwenden Zeichenketten, beispielsweise um sich mit einer URL verbinden zu können oder um Textausgaben zu drucken. Diese können einen guten Einblick über das Verhalten einzelner Programme liefern. Beispielsweise können dadurch IP Adressen für Command & Control (C2)-Systeme identifiziert werden, welche von Angreifern zum Verwalten von remote Sitzungen von infizierten Hosts verwendet werden.
Handelt es sich bei Dateien um sogenannte *Packer*, also komprimierte Programme, müssen diese zunächst entpackt werden, um eine erfolgreiche Analyse durchführen zu können. Bei Dateien die relativ wenige Zeichenketten enthalten, handelt es sich meist um Packer. Wie die nachfolgende Tabelle 3.1 zeigt, bietet die statische Analyse eine Vielzahl an Indikatoren, welche darauf hinweisen können, dass es sich bei der untersuchten Datei um Schadsoftware handelt (Sikorski 2012).

Ort	Indikator	Verhalten
System	neue/modifizierte Dateien	Veränderung des Dateisystems durch Malware
System	Registry Einträge	Veränderung/Erstellung von Registry Keys
PE Header	wenige Imports	durch Packer komprimierte Dateien, um die Erkennung und Analyse zu erschweren
PE Header - Imports	`SetWindowsHookEx`	empfängt Tastatureingaben (Keylogger)
PE Header - Imports	`RegisterHotKey`	bestimmte Tastenkombination startet Anwendung (Keylogger)
IMAGE_FILE_HEADER	Kompilierungszeit	verdächtige Kompilierungszeit
Sections	Abweichende Namen	z.B. .srtsa anstatt .data
SECTION .text	divergierende Speichergröße von Virtual Size und Raw Size	Packer extrahiert Code nach .text
SECTION .rsrc	eingebettetes Programm, Treiber	weitere durch Malware gestartete Aktionen

Tabelle 3.1.: Auszug von Indikatoren für einen Malware Angriff (in Anlehnung an Sikorski (2012))

Diese Indikatoren aus Tabelle 3.1, sind nur ein kleiner Teil dessen, was bei einer statischen Analyse entdeckt werden kann. Dennoch wird dadurch deutlich, wie hilfreich PE-Dateien im Erkennen von Malware sein können.
Welche Features aus diesen Dateien generiert werden können, um eine aussagekräftige Untersuchung mit Hilfe von MLAs zu generieren, wird in diversen Ansätzen beschrieben, mit welchen sich das folgende Kapitel beschäftigt.

4. Analyseverfahren

Dieses Kapitel beschäftigt sich mit Machine Learning Verfahren, welche für die Erkennung von Cyber Security Angriffen verwendet werden. Diese Verfahren wurden anhand einer umfangreichen Literaturrecherche ermittelt. Jedes Vorgehen wird auf dessen verwendete Algorithmen, sowie der ausgewählten Features untersucht. Zudem wird analysiert welche Evaluationskriterien verwendet wurden um die Effektivität zu messen und zu welchem Ergebnis der jeweilige Ansatz führte.
Das Vorgehen orientiert sich an einem problembezogenen Ansatz. Dazu wird im Folgenden eine Übersicht darüber generiert, welche Probleme aus der Cyber Security bereits von Machine Learning Algorithmen in Angriff genommen wurden. Dies dient einer übersichtlichen Darstellung der Möglichkeiten und zeigt die Diversität auf, in welcher MLAs die Arbeit von Cyber Security Spezialisten unterstützen und verbessern können.

Zunächst werden Verfahren erläutert, welche bereits wissenschaftlich belegt wurden. Konnten, dem Ansatz entsprechende Adaptionen aus der Industrie identifiziert werden, wurde der jeweilige Ansatz um den industriellen erweitert.

Vorab werden elementare Grundbegriffe sowohl aus dem Bereich Cyber Security als auch aus dem des maschinellen Lernens erläutert, um eine homogene Diskussionsbasis zu schaffen.

4.1. Grundbegriffe aus der Cyber Security

Um Probleme, die IT Security betreffend, zu besprechen, ist es hilfreich ein Modell zu haben, welches der Diskussion als Grundlage dient. Dies führt zu einem besseren Verständnis des Problems. Um solch eine Grundlage zu schaffen wird im Folgenden das sogenannte *CIA Triad*, im deutschen auch als *CIA Prinzipien* bekannt, erläutert. Zu den drei Schutzzielen der Informationssicherheit zählen Vertraulichkeit (engl. Confidentiality), Integrität (engl. Integrity) und Verfügbarkeit (engl. Availability). Aus diesen drei Prinzipien (siehe Abb. 4.1.1) lassen sich die Hauptbedrohungen der IT Security ableiten, dem Verlust der Vertraulichkeit, der Integrität und der

Verfügbarkeit (Andress 2019). Um Informationssicherheit zu erlangen, gilt es diese Bedrohungen abzuwehren.

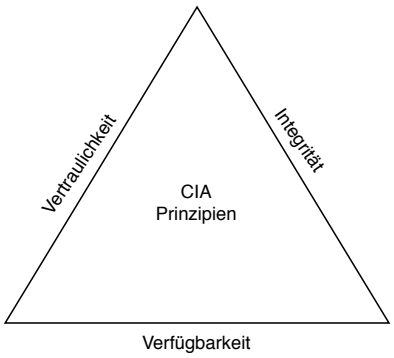


Abbildung 4.1.1.: Schutzziele der Informationssicherheit (eigene Darstellung)

Unter Vertraulichkeit versteht man den Schutz von Daten vor unautorisiertem Zugriff (Ingeno 2018). Daten dürfen also nur von dafür berechtigten Personen gelesen werden. Vor allem wenn es sich um private, personenbezogene Daten handelt gilt es diese besonders zu schützen, da andernfalls reputativer bis hin zu finanziellem Schaden für Einzelpersonen oder Unternehmen auftreten kann.
Integrität stammt von dem lateinischen Wort *integritas* ab, und bedeutet *Unversehrtheit*. Die Informationssicherheit meint mit der Unversehrtheit der Daten, dass deren Modifikation ausschließlich dafür berechtigten Personen zu steht. Bei Betriebssystemen können Zugriffsrechte für die Integrität sorgen. Beispielsweise kann der Besitzer einer Datei diese Lesen und Schreiben, andere Benutzer dürfen diese jedoch ausschließlich lesen. Dieselben Mechanismen finden sich auch in Datenbanken.
Ein weiteres zu schützendes Gut der Informationssicherheit ist die Verfügbarkeit von Daten. Informationen und Daten sollten Benutzern jederzeit zur Verfügung stehen. Beispielsweise der Moodle Server der THU sollte für Studierende permanent erreichbar sein, da darüber Lernmaterial bezogen und Vorprüfungsleistungen abgegeben werden. Eine Einschränkung der Verfügbarkeit dieses Services könnte für viele Studierende einen großen Nachteil bedeuten.

Die Verletzung mindestens eines dieser Schutzziele wird in der Informationssicherheit als Sicherheitsvorfall bewertet. Solche Vorfälle können unterschiedliche Ursachen haben. Probleme mit der Infrastruktur wie beispielsweise Hardware oder Software Fehler können dafür verantwortlich sein. Des Weiteren können durch Zwischenfälle wie Gewitter oder Feuer oder durch Katastrophen wie Kriege oder

Flugzeugabstürze, Sicherheitsvorfälle ausgelöst werden. In dieser Arbeit werden jedoch ausschließlich durch Menschen verursachte Sicherheitsvorfälle analysiert. Attackiert ein Angreifer ein System erfolgreich, ist immer mindestens ein Prinzip des CIA Triads verletzt (Ingeno 2018). Beispielsweise kann durch mangelnde Zugangskontrolle in Web Applikationen auf Seiten zugegriffen und Daten eingesehen werden, die ausschließlich für autorisierte Benutzer reserviert sind. Dadurch wäre die Vertraulichkeit verletzt.

Um eine Schwachstelle auszunutzen, muss sich der Angreifer zunächst einen *Angriffsvektor* zurecht legen. Unter Angriffsvektor versteht man die Strategie mit der eine Sicherheitslücke ausgenutzt werden kann (Rahalkar 2018). Das bedeutet, ein Angreifer muss sich vorab überlegen wie er mit einem potenziellen Opfer interagieren kann beziehungsweise will. Nachfolgend werden jeweils vier Angriffsvektoren pro CIA Prinzip kurz erläutert. Alle der beschriebenen Angriffsvektoren finden sich in den untersuchten Ansätzen wieder. Dadurch soll ein besseres Verständnis der Analyseverfahren gewährleistet werden.

- **Vertraulichkeit**
  - Twitter drive-by Downloads:
    Mittels verschleierter Links können Angreifer Zugriff auf die Systeme der Opfer und somit auf deren Daten erlangen (Javed, Burnap und Rana 2019).
  - Probe:
    Untersuchung von Netzwerken, um Informationen von Systemen zu sammeln und potenzielle Sicherheitslücken zu ermitteln.
  - SQL Injections:
    Bösartige SQL Abfragen können dem Angreifer Zugriff auf Benutzerdaten bis hin zu Datenbanken geben.
  - Remote to Local (R2L):
    Unautorisierter Zugriff auf ein entfernt liegendes System.
- **Integrität**
  - Cross-Site Scripting (XSS):
    Durch die Verwendung von JavaScript in Web Anwendungen, kann unter anderem auf einen Cookie zugegriffen und somit eine Benutzer Session gestohlen werden. Dadurch kann der Angreifer Zugriffs- und Modifizierungsrechte auf Benutzerdaten erhalten.

  - User to Root (U2R):
    Unbefugter Zugriff auf lokale Superuser Berechtigungen, wodurch Daten gelesen, modifiziert und gelöscht werden können.
  - Macrobasierte Attacken:
    Macros bestehen aus eingebettetem Visual Basic for Applications (VBA) Code, welcher für bösartige Zwecke wie dem Ausführen von Befehlen über den VBA-Shell-Befehl missbraucht werden können. Zudem können bösartige Dateien ausgeführt oder aus dem Internet heruntergeladen werden (Cohen u. a. 2016).
  - Phishing:
    Benutzer werden über E-Mails oder Links zu bösartigen Seiten geleitet und dahin gehend manipuliert, persönliche Informationen wie Benutzernamen, Passwörter oder Kreditkartennummern preis zu geben. Dadurch bekommt der Angreifer Zugang zu Benutzerkonten und kann somit finanziellen Schaden anrichten (Alswailem u. a. 2019).

- **Verfügbarkeit**
  - Denial of Service (DoS), Distributed Denial of Service (DDoS), Low-rate DDoS (LDDoS):
    Eine DoS Attacke führt zu einer Nichterreichbarkeit eines Internetservices durch Datenüberlastung. Um dies zu erreichen kann ein Client halboffene Transmission Control Protocol (TCP) Verbindungen mit einem Server herstellen. Das bedeutet, der Client sendet ein synchronize (SYN) Flag, der Server antwortet mit einem SYN acknowledge (ACK) Flag, bekommt nun aber kein ACK Flag vom Client zurück und reserviert so unnötig Ressourcen. Durch Flutung von SYN Flags können also Ressourcen eines Servers aufgebraucht werden. Bei einer DDoS Attacke handelt es sich um dasselbe Problem, allerdings ist der Verursacher kein alleiniger Client sondern besteht aus einem ganzen Netzwerk an Clients. Die LDDoS Attacke zielt nicht darauf ab einen Server zu fluten, sondern Anfragen bruchstückhaft und so langsam zu senden, dass dadurch ebenfalls alle Ressourcen aufgebraucht werden (Siracusano, Shiaeles und Ghita 2018).
  - Buffer Overflow:
    Ein Buffer Overflow entsteht wenn ein Programm mehr Daten aufnimmt, als vorgesehen war. Diese Daten laufen in benachbarte Puffer über und können dort Daten überschreiben, was zum Verlust der Systemintegrität führen kann.

- Ransomware:
  Hierbei handelt es sich um eine spezielle Schadsoftware, welche Ressourcen des Benutzer verschlüsselt und ein Lösegeld verlangt um ihm diese wieder zur Verfügung zu stellen (Maniath u. a. 2017).
- Botnetze:
  DDoS Attacken werden oftmals mit Hilfe von kompromittierten Systemen, beispielsweise durch Trojaner, durchgeführt. In diesem Fall spricht man von einem Botnetz, welches durch einen Master über beispielsweise Internet Relay Chat (IRC) oder HTTP gesteuert wird (Mathur, Raheja und Ahlawat 2018).

Allerdings muss nicht nur eines der CIA Prinzipien verletzt werden. Es gibt Angriffe, bei welchen alle drei Prinzipien betroffen sind, wie beispielsweise bei einer Remote Code Execution (RCE). Dabei wird zunächst Zugang zu einem Opfersystem erlangt auf welchem Daten unautorisierterweise gelesen werden können (Vertraulichkeit). Zusätzlich können Daten verändert werden (Integrität). Diese Veränderungen können beispielsweise die Zugangsdaten des Opfers betreffen, wodurch dieses keine Möglichkeit mehr hat auf dessen Daten zuzugreifen (Verfügbarkeit). Ein Angriffsvektor für dieses *worst case* Szenario bietet Malware, weshalb es besonders wichtig ist Systeme dahingehend zu schützen.

4.2. Grundbegriffe des maschinellen Lernens

Täglich werden 2,5 Trillionen Bytes digitale Daten erzeugt, in welchen immenses Wissenspotenzial steckt (MerlinOne 2019). Da kein Mensch Muster aus dieser Masse an Daten herauslesen kann, bedarf es maschinelles Lernen. Da es sich dabei um ein äußerst umfangreiches Themengebiet handelt, würde eine detaillierte Erläuterung dessen, den Umfang dieser Arbeit übersteigen. Daher werden im Folgenden lediglich ausgewählte Teilbereiche, sowie Begriffe, welche im weiteren Verlauf der Arbeit verwendet werden, erläutert. Dadurch wird ein besseres Verständnis der Probleme sowie der Lösungsansätze gewährleistet.

Im Bereich der IT Sicherheit wird im Falle eines Sicherheitsvorfalls, anomales Verhalten gesucht, um so den Angriff zu rekonstruieren. Beispielsweise kann ungewöhnlich hoher Netzwerkverkehr auf eine DoS Attacke hindeuten. Mit dem Erkennen solcher Ausreißer beschäftigt sich die *Anomalie Erkennung*. Gerade im Bereich Malware Erkennung bietet sich dies an da gutartige, sowie bösartige Programme jeweils ähnlichen Code besitzen, wodurch sich diese gut voneinander abheben. Um dies zu analysieren bietet Machine Learning zwei Ansätze: *supervised* und *unsupervised*

learning. Ersteres beschreibt ein Verfahren, bei welchem den zu untersuchenden Daten sogenannte *Labels* angefügt wurden. Diese kennzeichnen die Zugehörigkeit der Daten zu bestimmten Klassen. Beispielsweise kann ein zu untersuchendes Programm das Label *bösartig* oder *gutartig* besitzen, je nachdem ob es Malware oder gängiger Software angehört. Dabei wird eine zu untersuchende Probe *Sample* genannt.
Unter *unsupervised learning* versteht man das Lernen eines Algorithmus durch ein ungelabeltes Datenset. Diesen Daten fehlt also das Klassenattribut. Diese Methode, oft auch *Clustering* genannt führt zu einer explorativen Datenanalyse, mit Hilfe dieser, Daten ohne Vorwissen in Untergruppen zusammen gefasst werden können (Raschka und Mirjalili 2017). Zu den gängigsten Algorithmen im Bereich Anomalie Erkennung gehören Random Forest (RF), Decision Tree (DT), Logistic Regression (LR), k-Nearest Neighbors (k-NN) und SVM, weshalb diese im Folgenden kurz erklärt werden.
Um Dateien nach gut- und bösartig zu klassifizieren, werden zunächst MLAs trainiert. Dabei werden diesen, Dateien mit bestimmten *Features* gezeigt. Features sind bestimmte Eigenschaften einer Datei. Im Bereich der Malware Analyse können das beispielsweise die Anzahl komprimierter Daten oder die Anzahl suspekter Funktionsaufrufe sein. Anhand der Features wird der *Feature Space* definiert. Dabei handelt es sich um einen n-dimensionalen Raum, wobei n der Anzahl an Features entspricht. In diesem Raum befindet sich eine *Entscheidungsgrenze* (siehe Abb. 4.2.1), dies ist eine geometrische Struktur die durch den Feature Space verläuft und dabei Daten auf der einen Seite der Grenze als Schadsoftware und die auf der anderen Seite nach normaler Software klassifiziert.

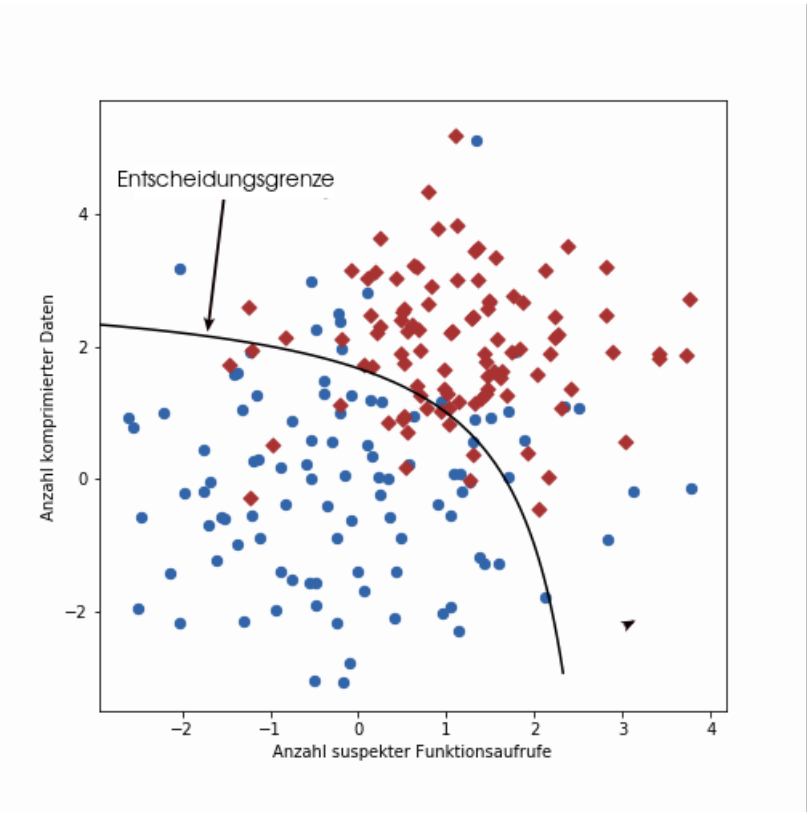


Abbildung 4.2.1.: Entscheidungsgrenze (eigene Darstellung)

Im 2-dimensionalen Raum ist die Entscheidungsgrenze eine Linie. Wird der Feature Space zu einem mehrdimensionalen Raum, ändert sich auch die Struktur der Entscheidungsgrenze von einer Linie zu einer Hyperebene, welche die Klassen voneinander trennt.

Logistic Regression produziert Linien, Ebenen oder Hyperebenen als Entscheidungsgrenzen. Bei diesem Algorithmus werden zunächst alle Features mit einem gewissen Gewicht multipliziert. Die Gewichtung skaliert oder verringert das Feature je nachdem wie indikativ für Malware dieses bewertet wird. Anschließend wird auf die Gesamtsumme aller Features und deren Gewichtung ein *bias*, also eine mögliche Verzerrung dazu addiert. Das Ergebnis wird durch die logistische Funktion in einen Wahrscheinlichkeitswert zwischen 0 und 1 umgewandelt. Durch Training wird die Entscheidungsgrenze dahingehend justiert, dass sich die Samples auf der jeweils richtigen Seite der Grenze befinden.

Im Gegensatz zu dieser einfach gehaltenen Entscheidungsgrenze, steht die von **k-Nearest Neighbors**. Dieser Algorithmus basiert auf der Idee, dass wenn die

Mehrheit der k nächsten Samples einer Datei, Malware angehören, die Datei selbst auch Malware ist. K impliziert hierbei die Anzahl der umliegenden Nachbarn die bei der Klassifizierung berücksichtigt werden. Die Distanz zu den umliegenden Nachbarn wird durch eine Abstandsfunktion wie der euklidischen Entfernung gemessen. Zunächst wird die quadratische Differenz aller Features für jedes Sample berechnet. Anschließend wird die euklidische Entfernung zwischen den zuvor entstandenen Feature Vektoren berechnet. Da die Entscheidungsgrenze von k-NN nicht durch lineare Strukturen beschränkt ist, lässt dieser Algorithmus komplexere Modellierungen zu (Joshua Saxe 2018).

Ein weiterer oft genutzter Algorithmus zur Anomalie Erkennung ist der **Decision Tree**. Dieser generiert während des Trainings automatisch eine Reihe von Fragen, wie beispielsweise in Abbildung 4.2.2 ersichtlich, um zu entscheiden welcher Klasse ein Sample angehört.

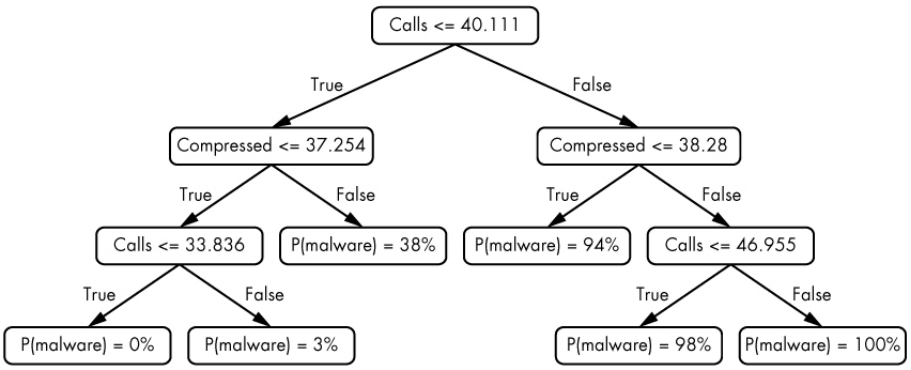


Abbildung 4.2.2.: Decision Tree (Joshua Saxe 2018)

Als Wurzelknoten sollte eine Frage bezüglich eines aussagekräftigen Features gewählt werden. Anschließend wird die Reduzierung der Unsicherheit für jede Frage berechnet und diejenige gewählt die diese am besten reduziert. Wie lange ein Baum Fragen stellt kommt darauf an, welche Anzahl an Fragen oder welche Tiefe des Baumes angegeben wird. Da Decision Trees gezackte Entscheidungsgrenzen generieren, führen sie manchmal zu inakkuraten Modellen (Joshua Saxe 2018).

Dieser Mangel an Präzision, kann durch eine Fülle an Bäumen kompensiert werden, wie es der **Random Forest** Algorithmus bietet. Dabei handelt es sich um die Verwendung von hunderten oder tausenden Decision Trees gleichzeitig. Anstatt einen Baum zu trainieren werden eine Vielzahl von Bäumen in unterschiedlicherweise trainiert, sodass der Algorithmus eine breit gefächerte Perspektive der Daten bekommt. Anschließend wird die Wahrscheinlichkeit bestimmt mit welcher eine

Datei, im Falle von Malware Erkennung, bösartig ist. Dafür wird die Anzahl positiv bewertender Bäume durch die Gesamtanzahl der Bäume geteilt. Um zu vermeiden, dass alle Bäume gleich sind und somit zu demselben Ergebnis gelangen, bedarf es unterschiedlicher Perspektiven der Bäume (Joshua Saxe 2018). Um dies zu gewährleisten wird für jeden Baum ein zufällig gewähltes Trainingsset verwendet und an einer unbekannten Datei getestet. Dadurch kann, im Vergleich zu einzelnen Decision Trees, eine viel weichere Entscheidungsgrenze gezogen werden.

Die **Support Vector Machine** sucht nach einer Gerade oder Hyperebene die den größtmöglichen Abstand zu den jeweils nächstgelegenen Punkten (siehe Abbildung 4.2.3) der verschiedenen Klassen besitzt. Der Abstand der Vektoren zur Trennlinie wird dabei maximiert, sodass eine zuverlässige Klassifikation entsteht.

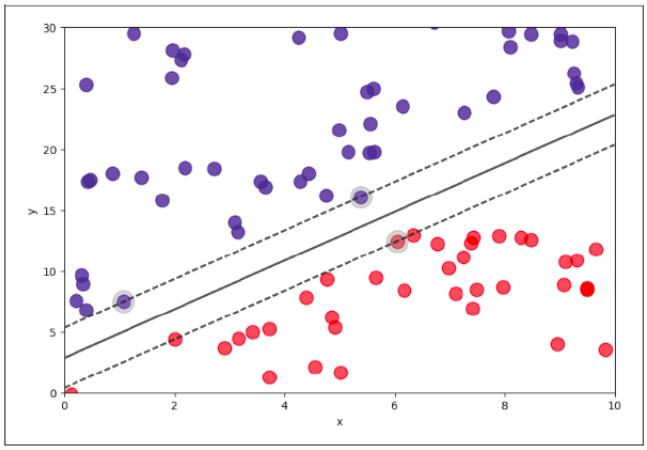


Abbildung 4.2.3.: Entscheidungsgrenze und Support Vektoren der SVM (C. N. Nguyen und Zeigermann 2018)

Wird die SVM auf ein nicht-lineares Klassifizierungsproblem angewandt, werden die Daten in einen Raum höherer Dimension abgebildet. Dadurch wird die Anzahl möglicher linearer Trennungen erhöht. Diese Methode wird als *Kernel-Trick* bezeichnet. Weiterführende Informationen bezüglich des mathematischen Hintergrunds können beispielsweise in C. N. Nguyen und Zeigermann (2018) eingesehen werden.

Eine besondere Art von Machine Learning ist **Deep Learning**. ”Deep” bezieht sich auf die Architektur die dabei genutzt wird. Diese besteht aus mehreren Schichten von Verarbeitungseinheiten, wobei jede die Ausgabe der vorherigen Schicht als Eingabe verwendet. Jede dieser Verarbeitungseinheiten wird Neuron genannt. Daher auch der Name **neuronales Netz**. Ein Neuron (siehe Abb. 4.2.4) besteht dabei aus einer Eingabe aus welcher eine gewichtete Summe kalkuliert wird. Diese

Summen werden aufaddiert und um einen bias ergänzt. Die Gewichte sowie der bias sind die Parameter eines Neurons, welche sich im Laufe des Trainings ändern und dadurch das Modell optimieren. Anschließend wird auf die so entstandene Summe eine Aktivierungsfunktion angewendet. Aufgabe dieser ist es eine nichtlineare Transformation auf die gewichtete Summe anzuwenden.

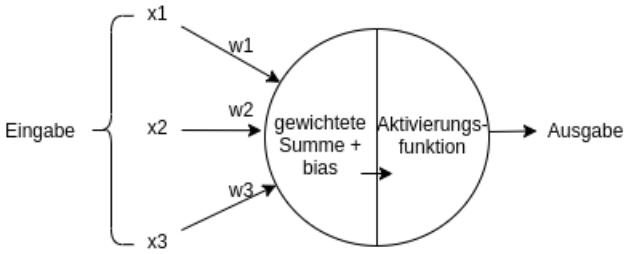


Abbildung 4.2.4.: Aufbau eines Neurons nach Joshua Saxe (2018) (eigene Darstellung)

Es gibt eine Vielzahl von Aktivierungsfunktionen, deren Funktionsweisen welche Joshua Saxe (2018) ausführlich beschreibt. Um ein neuronales Netz zu erstellen, werden Neuronen zu einem Graph, mit einer gewissen Anzahl an Schichten angeordnet. Besteht das Netz aus einem gerichteten Graph spricht man von einem **Feed Forward** Netz, da Daten nur in eine Richtung (von links nach rechts) fließen. Neuronale Netze verfügen über die Fähigkeit, Features automatisch zu extrahieren. Stellt man einem Netz beispielsweise eine HTML Datei zur Verfügung, kann jede Schicht lernen diese rohen Daten entsprechend zu repräsentieren, sodass diese für die Eingabe in nachfolgenden Schichten verwendet werden können. Dadurch kann nicht nur eine Menge Zeit sondern auch Arbeit gespart werden (Joshua Saxe 2018). Da neuronale Netze oftmals aus tausenden von Neuronen bestehen, bedarf es einer effizienten Art der Parameteroptimierung. Während des Trainings bekommt das Netz eine Eingabe x und errechnet daraus eine Ausgabe $\hat{y}$. Das Ziel ist es nun die Parameter eines Neurons dahingehend zu verändern, dass $\hat{y}$ mehr dem tatsächlichen Wert y entspricht. Die iterative Berechnung und Aktualisierungen von Parametern wird als *gradient descent* bezeichnet. Weiterführende Informationen hierzu finden sich bei Bonaccorso, Fandango und Shanmugamani (2018). Bei einem Netzwerk von tausenden Neuronen und Millionen von Parametern bedeutet dies enorm viel Rechenaufwand. Um diesen zu umgehend wird der *backpropagation* Algorithmus verwendet. Dieser ermöglicht es Optimierungen entlang von Graphen, wie neuronalen Netzen, effizient zu berechnen. Eine ausführliche Beschreibung hierzu findet sich in Krohn, Beyleveld und Bassens (2019).

Die eben beschriebene Funktionsweise beschreibt die eines Feed Forward Netzes,

welches das standard Netz bildet. Des Weiteren gibt es **Convolutional Neural Networks (CNNs)**. Diese beinhalten eine *convolutional* Schicht, wobei die Eingabe jedes Neurons durch ein Fenster definiert wird, welches über den Eingabebereich gleitet. Aus diesen Ausschnitten wird anschließend meist der größte Wert der nächsten, so genannten *pooling* Schicht übergeben. Durch dieses herauszoomen wird die Anzahl der Features reduziert, was zu einer schnelleren Berechnung führt. Ihre Fähigkeit sich auf lokalisierte Bereiche in Eingabedaten zu fokussieren, macht sie besonders nützlich für Bilderkennungsverfahren oder Natural Language Processing (NLP).

Eine weitere Art eines neuronalen Netzes ist das **Recurrent Neural Network (RNN)**. Dieses unterscheidet sich insofern von dem standard Netz, da sich der Informationsfluss nicht auf eine Richtung beschränkt. Das bedeutet es gibt nicht nur Verbindungen zur nächsten Schicht wie bei Feed Forward Netzen, sondern auch Verbindungen zu der selben oder der vorangegangenen Schicht. Diese Art von Netz ist besonders dann sinnvoll, wenn die Reihenfolge von Daten eine Rolle spielt, wie beispielsweise bei Spracherkennung, Sprachübersetzung und Zeitreihenanalysen. Eine ausführliche Beschreibung zu dieser Art von Netzen findet sich in Ravichandiran (2018).

Um die Effizienz der einzelnen Algorithmen messen zu können, bedarf es **Leistungsmetriken**. Anhand dieser Metriken kann die Leistung individuell gemessen werden, wodurch deutlich wird wie zuverlässig der Algorithmus arbeitet. Zusätzlich kann dadurch der Trainingserfolg nach jedem Durchlauf verfolgt werden. Ohne Leistungsmessung wäre es zudem schwierig Parameter zu optimieren.
Basis der im Folgenden vorgestellten Leistungsmetriken sind vier Ergebnisse:

- **True positive**
  Die Datei ist Malware und wird auch als solche erkannt
- **False negative**
  Die Datei ist Malware wird aber nicht als solche erkannt
- **False positive**
  Die Datei ist keine Malware wird aber als solche erkannt
- **True negative**
  Die Datei ist keine Malware und wird auch nicht als solche erkannt

Entsprechend gibt es zwei Szenarien in denen Erkennungssysteme unzureichende Aussagen treffen können: *false negative* und *false positive*. Dem zufolge sind *true negative* und *true positive* erstrebenswert. Jedoch sind diese Werte allein nicht ausreichend, um ein Ergebnis vollständig zu bewerten. Eine hohe true positive Rate

kann erzielt werden, wenn das Verhältnis zwischen Malware und gutartiger Software extrem unausgewogen ist. Beispielsweise ein solches Verhältnis von 1:10 könnte eine sehr hohe true negativ Rate erzielen auch wenn das Lernmodell insgesamt eine schlechte Leistung erbringt. Um dies zu vermeiden, ist ein ausgewogenes Verhältnis von bösartiger und gutartiger Software erforderlich, sowie eine prozentuale Betrachtung der true positive und der true negative Rate (Aldwairi, Perera und Novotny 2018).

Die **Genauigkeit** spiegelt den Anteil den ein Modell korrekt klassifiziert hat wider:

$$Genauigkeit = \frac{TP + TN}{TP + TN + FP + FN}$$

In einem wie im letzten Abschnitt erwähnten Beispiel würde diese einen sehr hohen Wert erzielen trotz der schlechten Leistung des Modells.

Ein weiterer Indikator bietet die **Receiver Operator Characteristic Area Under Curve (ROC AUC)** dabei werden die True Positive Rate (TPR) sowie die False Positive Rate (FPR) betrachtet. Diese setzen sich folgendermaßen zusammen:

$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$

$$False\ Positive\ Rate = \frac{FP}{TN + FP}$$

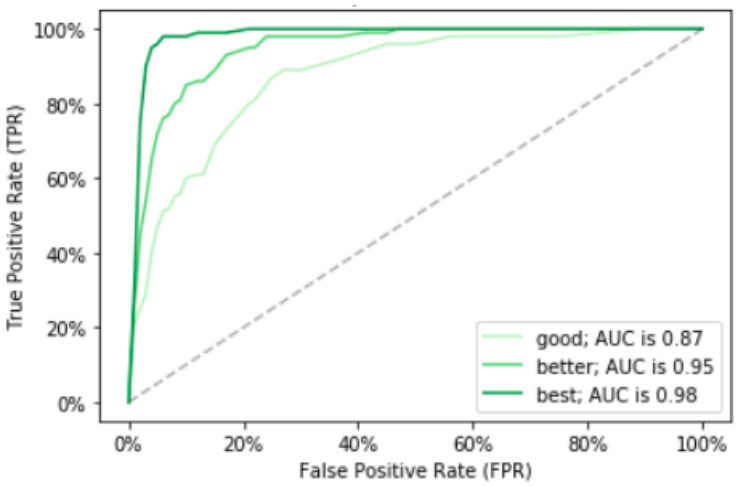


Abbildung 4.2.5.: Beispielhafte ROC Kurven (Molin 2019)

Wie in Abbildung 4.2.5 ersichtlich, ist eine ROC AUC die gegen 1 strebt wünschenswert.

Als weiterer Leistungsindikator dient die **Precision**. Hierbei wird der Anteil an richtig klassifizierten Samples mit den insgesamt als positiv klassifizierten betrachtet.

$$Precision = \frac{TP}{TP + FP}$$

Ein weiterer interessanter Indikator ist der **Recall**. Dieser betrachtet nicht nur die positiv klassifizierten Samples, sondern berücksichtigt auch jene, die fälschlicherweise für gutartig befunden wurden, obwohl es sich um Malware handelt. Diese Einteilung ist fatal, da sie für Benutzer und Systeme gefährlich sein kann.

$$Recall = \frac{TP}{TP + FN}$$

Ein Indikator welche die beiden vorangegangenen vereinigt ist der **F-measure** (auch F1-measure oder F-score genannt). Dieser setzt sich aus Precision und Recall zusammen und bildet deren harmonisches Mittel.

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Ein Modell mit perfekter Precision und Recall erzielt einen F-measure von 1. Jedoch erzielt ein Modell mit einer perfekten Precision aber einem Recall von 0 insgesamt einen F-measure von 0. Des Weiteren gibt es den F2-measure, welcher den Recall höher gewichtet als die Precision. Respektive gibt es den F0.5-measure, welcher die Precision höher bewertet als den Recall.

Die Metrik **log loss** (logarithmic loss) misst die Leistung eines Modells, welches als Vorhersageeingabe einen Wahrscheinlichkeitswert zwischen 0 und 1 hat. Ein perfektes Modell hätte einen log loss von 0. Dieser steigt wenn der vorhergesagte Wert für ein bestimmten Label vom tatsächlichen Label abweicht.

$$logloss = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \log(p_{ij})$$

Dabei repräsentiert N die Anzahl an Samples im Testset und M die Anzahl an Labels. log ist der natürliche Logarithmus und y_{ij} ist 1, wenn das vorhergesagte Label mit dem tatsächlichen übereinstimmt und 0 wenn dies nicht zu trifft. p_{ij} ist, dass das Sample i zur Klasse j gehört. Der detailliertere mathematische Hintergrund kann in Nielsen (2019) eingesehen werden.

4.3. Angewandte Ansätze

Für die Recherche wurden alle Verfahren in einem Zeitraum von 2015 bis heute berücksichtigt. Diese Periode wurde gewählt, da sich die Zahl der Cyberangriffe, sowie die der zur Verfügung stehenden Schadsoftware schon innerhalb weniger Jahre deutlich vermehrt beziehungsweise verändern kann. Somit soll verhindert werden, bereits ausführlich erforschte Angriffsvektoren zu analysieren. Zusätzlich gibt es bereits vergleichbare Arbeiten aus dem Jahr 2016 (s. Buczak und Guven 2016), in welchem Machine Learning Ansätze vor dieser Zeit analysiert werden.
In der folgenden Auflistung wird die Bezeichnung *Erkennung* für binäre Klassifikation verwendet. Wenn sich ein Ansatz beispielsweise darauf beschränkt Daten entweder in die Rubrik A *bösartig* oder in die Rubrik B *gutartig* zu klassifizieren. Erfolgt in einem Analyseverfahren eine Einteilung in mehr als zwei Klassen, wird nachfolgend der Begriff *Klassifizierung* verwendet.

4.3.1. Erkennung von Malware - Hybride Analyse (2015)

Im Jahr 2015 haben Shijo und Salim (2015) einen, auf zwei Analysen basierenden, Ansatz gewählt, um Malware zu erkennen. Dabei vermischten sie die statische Analyse mit der dynamischen, um so einen hybriden Ansatz zu erreichen. Zum einen verwendeten sie ein statisches Analyseverfahren bei dem sie *Printable Strings*, also nicht kodierte Zeichenfolgen wie z. B. *FindFirstFile* aus Binärdateien extrahierten. Zum anderen konfigurierten sie eine Cuckoo Sandbox, in der sie Schadsoftware ausführten und deren API Aufrufe in einer Protokolldatei speicherten.
Sie untersuchten die Ähnlichkeit in API-Aufrufsequenzen anhand von n-Grammbasierter Ähnlichkeitsmessung. Als Features dienten Tri- und Tetragramme ab einer gewissen Häufigkeit, sowie PrintableStrings ab einer Häufigkeit von zwei. Für die Klassifizierung wurden die Algorithmen RF und SVM verwendet. Es wurden jeweils beide Ansätze separat, sowie in Kombination getestet. Analysen mit SVM erzielten eine Genauigkeit von 95.88 % für die statische Analyse und 97.16 % für die dynamische Analyse und waren somit erfolgreicher, als Untersuchungen mit Random Forest. Die besten Ergebnisse erzielte der hybride Ansatz mit SVM mit einer Genauigkeit von 98.71 % und der geringsten FPR von 0.026.
Die Forschung von Shijo und Salim (2015) zeigt also, dass mit den von ihnen gewählte Features, mit einem hybriden Ansatz, deutlich genauere Aussagen, als mit rein statischen oder rein dynamischen Analysen, getroffen werden können.

4.3.2. Erkennung von Malware - Statische Analyse (2016)

More und Gaikwad (2016) untersuchten EXE-Dateien auf Schadsoftware. Dazu konvertierten sie die Dateien zunächst in Operation Code (Opcode), also in den Teil der Maschinensprachanweisung der die auszuführenden Operationen angibt, z.B. `55 8B EC 83 EC 5C 83 7D 0C 0F 74 2B 83 7D 0C 46`. Das ausgewählte Feature Datenset wurde anschließend nochmals zu einer Attribute-Relation File Format (ARFF) Datei konvertiert, um die Datei nachfolgend mit der Machine Learning Software Weka bearbeiten zu können. In Weka wurden die Algorithmen JRip, C4.5 und Instance-Based k (IBk) verwendet. Wobei es sich bei JRip und C4.5 um DT und bei IBk um k-NN Implementierungen handelt. Um die Erkennungsgenauigkeit zu erhöhen, wurden nicht nur die einzelnen Algorithmen, sondern ein Klassifikatorensemble angewandt, um Methoden wie Mehrheitsvoting, Veto-Voting und vertrauensbasiertes Veto-Voting verwenden zu können. Ersteres folgt demokratischen Regeln, das heißt, die Klasse mit den meisten Stimmen ist das Ergebnis. Veto-Voting hingehen basiert auf Annahmen über die Wahl der anderen Algorithmen. Vertrauensbasiertes Veto-Voting ergänzt voriges Voting um eine Vertrauensberechnung, wodurch jedem Algorithmus ein bestimmtes Vertrauensniveau zugeteilt wird. Weiterführende Informationen bezüglich dieser Methoden können Shahzad und Lavesson (2013) entnommen werden.
More und Gaikwad (2016) konnten zeigen, dass durch die Verwendung von Veto-Voting eine Genauigkeit von 80.7 % erzielt werden kann. Im Vergleich dazu, lag das beste Ergebnis, welches durch singulären Algorithmeneinsatz von IBk erzielt wurde, bei einer Genauigkeit von nur 73.5 %.

Dieses Ergebnis stützt die These von Shijo und Salim (2015) aus dem Jahr zuvor, welche zeigten, dass ein nicht-hybrider Ansatz weniger genau ist, als einer, der die statische und die dynamische miteinander verknüpft.

4.3.3. Erkennung bösartiger XML-basierter Office Dokumente(2016)

Durch das neue Dateiformat, welches 2007 auf den Markt gebracht hat, sollten Sicherheitslücken geschlossen werden. Das Binärformat wurde durch ein XML basiertes Dateiformat ersetzt. Dadurch werden neue digitale Funktionen unterstützt, sowie vertrauensbasierte Bereiche geschaffen, welche das Format weniger riskikoreich gestalten sollen. Dennoch können Attacken gegen XML-basierte Office Dokumente gestartet werden. Zu den möglichen Angriffsvektoren zählen beispielsweise macrobasierte Attacken. Durch den Missbrauch von VBA kann die zugehörige

shell gestartet werden, um willkürliche Kommandos zu senden. Außerdem können externe Bibliotheken sowie Programme aufgerufen werden, welche Schaden verursachen können. Eine weitere Bedrohung durch den Gebrauch von Macros bildet die Fähigkeit dieser, bösartige Dateien aus dem Internet herunterzuladen.
Cohen u. a. (2016) haben in ihrer Arbeit diese Art von Angriffsvektoren untersucht, um eine Infizierung durch bösartige Macros frühzeitig zu erkennen. Dazu haben sie zunächst Office Dokumente in eine Liste von Pfaden konvertiert. Diese dienen der Analyse als Features.

Feature	Structural path	Category
f1	*word\vbaProject.bin*	*Macro*
f2	*word_rels\vbaProject.bin.rels*	*Macro*
f3	*word_rels\vbaProject.bin.rels\Relationships*	*Macro*
f4	*word_rels\vbaProject.bin.rels\Relationships\Relationship*	*Macro*
f5	*word\vbaData.xml*	*Macro*
f6	*word\vbaData.xml\wne:vbaSuppData*	*Macro*
f7	*word\vbaData.xml\wne:vbaSuppData\wne:mcds*	*Macro*
f8	*word\vbaData.xml\wne:vbaSuppData\wne:mcds\wne:mcd*	*Macro*
f9	*word\embeddings*	*Embedded*
f10	*word\embeddings\oleObject1.bin*	*OLE*
f11	*word\vbaData.xml\wne:vbaSuppData\wne:docEvents*	*Macro*
f12	*word\vbaData.xml\wne:vbaSuppData\wne:docEvents\wne:eventDocOpen*	*Macro*
f13	*word\media\image1.emf*	*EMF*

Abbildung 4.3.1.: Features als Pfade dargestellt (Cohen u. a. 2016)

Dadurch wurde jedoch eine so hohe Anzahl an Features generiert, dass die Untersuchung mit verschiedenen Datensets durchgeführt wurde, welche Top Features von 10 bis 2000 beinhalteten. Um die Feature-Repräsentation, also das Vorhandensein bzw. die Wichtigkeit von Features zu bestimmen, wurden zwei Verfahren angewandt. Zum einen ein binäres Verfahren, welches lediglich die Ab-, respektive Anwesenheit eines Features misst und zum anderen wurde das statistische Verfahren Term Frequency - Inverse Document Frequency (TF-IDF) verwendet, um die Wichtigkeit eines Terms in Bezug auf ein Dokument zu bestimmen. Anschließend wurden die Daten mit folgenden Algorithmen untersucht: J48, RF, LR, Naïve Bayes (NB), Bayesian Network (BN), LogitBoost (LB), Sequential Minimal Optimization (SMO), Bagging und AdaBoost (AB).

Wie die Ergebnisse zeigen, erzielt das Datenset mit den Top 200 Features, welches mit Random Forest analysiert wurde die besten Werte mit einem F-measure von 0.66. Wie sich demonstrieren ließ, ist es möglich bösartige Office Dokumente durch eine Analyse deren Pfade zu erkennen. Die Untersuchung beschränkt sich jedoch auf direkte Gefahren innerhalb von Dokumenten. Indirekte Gefahren, wie etwa die durch weiterführende Links, wurden in dieser Forschungsarbeit nicht berücksichtigt.

4.3.4. Erkennung bösartiger HTTP-Anfragen (2016)

Pham, Hoang und Vu (2016) haben in ihrer Arbeit ein Intrusion Detection System (IDS) für HTTP - Anfragen entwickelt. Dazu nutzten sie ein von ihnen entwickeltes Modul, um Netzwerkpakete zu erfassen. Dieses Modul basiert auf derselben Erfassungstechnik die `Wireshark` verwendet. Um ein geeignetes Model zu evaluieren, welches es ermöglicht die Pakete in Echtzeit zu klassifizieren, wurden diverse MLAs anhand des CSIC 2010 HTTP Datensets von Carmen Torrano Giménez, Alejandro Pérez Villegas (2010) getestet. Dieses besteht aus 223585 Daten die entweder als *normal* oder *anomal* gelabelt sind. Es enthält Attacken wie SQL-Injection, Buffer Overflow und XSS. Pham, Hoang und Vu (2016) trainierten und testeten die Daten mit den Algorithmen Decision Tree, Random Forest, AdaBoost, Logistic Regression und dem Stochastic Gradient Descent Classifier (SGDClassifier). Als Evaluationsmethode wurden Precision, Recall und F-measure verwendet. Den höchsten dieser Werte erzielte die Logistic Regression Methode mit einem F-measure von 0.96 für abnormen und 0.95 für normalen Verkehr. Zukünftig sollen diese Ergebnisse an Paketen, welche durch das eigens entwickelte Modul der Forscher erfasst wurden, evaluiert werden.

4.3.5. Klassifizierung von Netzwerkattacken (2017)

Yin u. a. (2017) implementierten ein IDS, für welches sie Recurrent Neural Networks (RNNs) verwendeten. Zusätzlich wurde die Leistung des Models bei binärer als auch bei Multi-Klassen Klassifikation untersucht. Um die Effizienz zu prüfen, wurde ferner ein Vergleich mit diversen MLAs gezogen. Die Analyse basiert auf dem NSL-KDD Datenset aus dem Jahr 2009 (Cybersecurity 2019). Dieses beinhaltet neben dem normalen Netzwerkverkehr, Daten zu vier verschiedenen Angriffstypen die da wären: DoS, R2L, U2R und Probe Attacken. Das Datenset besteht aus 41 Features. Um Vergleiche mit anderen MLAs herzustellen, wurden parallel Experimente mit Artificial Neural Network (ANN), RF, NB, Multi-Layer Perceptron (MLP), SVM, J48, Random Tree und Naïve Bayesian Tree (NBTree) für die binäre Klassifikation durchgeführt. Auf dieselbe Weise wurde die Multi-Klassen Klassifikation überprüft. Als Qualitätsmerkmal der Ergebnisse wurde der Wert *Accuracy* gewählt, welcher die Genauigkeit der Analyse wider spiegelt. Dieser Wert basiert auf dem Prozentsatz der korrekt klassifizierten Daten im Vergleich zur Gesamtheit der Daten. Das Ergebnis zeigt, dass RNNs eine qualitativ hochwertigere Analyse produzieren als die zu verglichenen MLAs. Bei der binären Klassifikation des Testsets erzielte auf RNNs basierende Untersuchungen eine Genauigkeit von 83.28% gefolgt von NBTree mit 82.02%. Bei der Klassifizierung in fünf Klassen erreichte das RNN mit 81.29% ebenfalls ein besseres Ergebnis als die restlichen Algorithmen. Jedoch fanden Yin

u. a. (2017) heraus, dass RNNs deutlich mehr Zeit für das Training beanspruchen, dieses Problem soll zukünftig durch Nutzung der Graphics Processing Unit (GPU) beschleunigt werden.

4.3.6. Erkennung von Ransomware - Dynamische Analyse (2017)

Maniath u. a. (2017) haben eine dynamische Analyse entwickelt um Ransomware anhand von API Aufrufen zu klassifizieren. Ransomware beschreibt eine Art Schadprogramm, welche dem Benutzer Ressourcen entzieht und eine Lösegeldsumme verlangt um diese wieder verfügbar zu machen. Um die Aufrufe zu extrahieren verwendeten die Forscher die dafür entwickelte Umgebung von Cuckoo Sandbox. Dadurch konnten die Schadprogramme in einer Umgebung ausgeführt werden ohne Schaden zu erzeugen. Die Anwendung erfasst alle API Aufrufe und speichert diese in einer `.json` Datei. Für die Analyse wurden 157 schadhafte Dateien aus nicht näher beschriebenen Onlinequellen verwendet. Dabei konnten 239 Aufrufe extrahiert werden, welche der Untersuchung als Features dienten. Um eine einheitliche Länge dieser zu gewährleisten, wurden die einzelnen Aufrufe entsprechend der längsten Sequenz mit Nullen aufgefüllt. Anschließend wurden die Daten entweder mit 0 für *gutartig* oder mit 1 für *ransomware* gelabelt. Anhand dieser Daten wurde ein LSTM Netzwerk trainiert. Nach einer Trainingszeit von zwei Stunden erreichte das Model eine Genauigkeit von 96.67% bei der Analyse der Testdaten. Die komplette Prozess einschließlich der Gewinnung der API Sequenzen dauerte allerdings 56 Stunden, was in Anbetracht der geringen Menge an Datensätzen doch sehr nachteilig ist.

4.3.7. Erkennung von Malware - Imageanalyse (2017)

Eine potenziell schnellere Methode um Malware zu erkennen, entwickelten Choi u. a. (2017) anhand einer Image Analyse. Dazu generieren sie Bilder von ausführbaren Dateien. Dabei hat jedes Pixel einen Wert zwischen 0 und 255. Um aus einer Datei ein Bild zu erhalten, wird jedes Byte eingelesen und zu einer Ganzzahl konvertiert die einem Pixel entspricht. Dadurch entstehen 256x256 Bilder. Da dadurch während einer Analyse der Speicher zur Neige geht wurden die Daten auf 32x32 reduziert. Die dadurch generierten Bilder dienen der Analyse mit einem Convolutional Neural Network (CNN) als Features. Die 2000 dafür verwendeten Schadprogramme, stammen aus einem koreanischen Cyber Security Forschungszentrum. Als Metrik zur Überprüfung der Genauigkeit wurde hier ebenfalls die Genauigkeit gewählt, welche sich auf 95.66% beläuft.

Bedauerlicherweise haben Choi u. a. (2017) nicht erwähnt in welcher Geschwindigkeit sie ihre Analyse durchführen konnten. In Anbetracht der von ihnen aufgestellten These - Imageanalysen seien viel schneller als statische und dynamische Analysen - wäre dies ein interessantes Detail gewesen.

4.3.8. Erkennung böswilliger MS Office Dateien (2017)

Bearden und Lo (2017) konzentrierten sich in ihrer Forschung darauf eine Methode zu entwickeln, mit welcher sich Microsoft Office Dokumente anhand ihrer Macros nach *gutartig* und *bösartig* klassifizieren lassen. Dazu untersuchten sie 158 Dateien, von welchen 40 schadhaft waren. Dokumente die Macros enthalten, beinhalten nicht nur VBA Code sondern auch *p-code*. Dabei handelt es sich um Assembler-Code, welcher vom VBA-Interpreter generiert wird, nachdem der Code einmal ausgeführt wurde (Bearden und Lo 2017). Diese Codes dienten der Analyse mit k-NN als Features. Die Effizienz wurde, wie die beiden zuletzt erläuterten Ansätze, anhand der Genauigkeit gemessen. Es wurden verschiedene Experimente mit unterschiedlichen K und L durchgeführt, wobei K die Anzahl an Clustern und L die Anzahl der besten Features impliziert. Den besten Wert erzielte eine Kombination mit K=3 und den Top 75 Features mit einer Genauigkeit von 96.3%.
Es wurde ausschließlich ein Algorithmus getestet, da die Intention der Forschung darin bestand, einen *Proof of concept* für die Erkennung bösartiger Macros anhand von p-codes bereitzustellen. Der Vorteil den Bearden und Lo (2017) mit ihrer Forschung geschaffen haben, besteht darin, dass potenziell bösartige Dateien nicht geöffnet werden müssen, bevor sie analysiert werden können. Jedoch weisen auch sie auf den Mangel an adäquaten Trainingsdaten hin, welchen es zukünftig zu beheben gilt.

4.3.9. Klassifizierung von DDoS Attacken (2017)

Bereits seit den 80er Jahren gibt es DoS Attacken, welche Netzwerk Ressourcen erschöpfen und dadurch die Verfügbarkeit von Services blockieren. Es gibt zwei Arten diese Attacken zu erkennen: Zielseitige Verteidigung und Quellenseitige Verteidigung (Z. He, Zhang und Lee 2017). Die Zielseitige Erkennung hat den Nachteil, dass Attacken erst entdeckt werden nachdem sie bereits beim Opfersystem ankamen. Z. He, Zhang und Lee (2017) forschen an einem pro aktiven Ansatz, bei dem die Erkennung auf der Quellenseite erfolgen soll. Dadurch können Angriffe auf mehrere Systeme verhindert werden. Dabei konzentrieren sie sich auf vier gängige DDoS Angriffe aus der Cloud: Secure Shell (SSH) brute-force Attacken, Internet Control Message Protocol (ICMP) flooding Attacken, Domain Name System

(DNS) reflection Attacken und TCP SYN Attacken. Als Feature für SSH brute-force Attacken dient die Anzahl an Diffie-Hellman Schlüsselaustausche, da dieser Wert während einer solchen Attacke steigt (Z. He, Zhang und Lee 2017). Als Feature für DNS reflection Attacken wurde das Verhältnis von eingehenden und ausgehenden DNS-Paketen gewählt, da bei einer Attacke mehr Anfragen als Antworten gesendet werden. Die ICMP Paketrate wurde als Feature für ICMP flooding Attacken gewählt, da bei normalem Verkehr eine geringere Anzahl dieser Pakete vorhanden ist. Um TCP-SYN Attacken zu identifizieren, wurde das SYN/ACK Verhältnis als Feature gewählt, da während einer solchen Attacke mehr SYN Tags als ACK Tags in den Paketen zu finden sind. Für das Experiment wurde der Netzwerkverkehr 9 Stunden lang verfolgt und anschließend mit den Algorithmen LR, SVM, DT, NB, RF, k-means und Gaussian-Mixture Model for Expectation-Maximization (GMM-EM) klassifiziert. Zur Evaluation dienten die Metriken: Precision, Genauigkeit, Recall und F-measure. Das beste Ergebnis mit einem F-measure von 0.99 und einer Genauigkeit von 99.73% konnte mit SVM erzielt werden. Durch diesen vielversprechenden Ansatz sollen zukünftig weitere DDoS Attacken pro aktiv erkannt werden.

4.3.10. Schwachstellen Scanner für Web Applikationen (2017)

Um Schwachstellen in Web Applikationen zu finden, wurde bei diesem Ansatz ein System namens Bug Terminating Bot (BTB) entwickelt (Robin Tommy, Gullapudi Sundeep 2017). Dieser Schwachstellenscanner überprüft Websites auf potenzielle Angriffsvektoren und liefert gleichzeitig Lösungen, um diese zu beheben. Zunächst überträgt der Bot alle Seiten einer Webapplikation und sucht innerhalb dieser nach ausnutzbaren Schwachstellen. Das Überprüfen basiert auf dem Ausführen von Payloads für gefundene Konflikte. Beispielsweise können Seiten auf SQL-Injections und XSS Attacken überprüft werden, in dem adäquate Payloads ausgeführt werden. Anschließend werden Code Vorschläge geliefert, welche die gefundenen Schwachstellen schließen sollen. Das Ergebnis des Scans, sowie die Verbesserungsvorschläge basieren auf Machine Learning. Nach dem Scan werden die Daten an einen zentralisierten Server geschickt, auf welchem eine SVM Informationen zu Schwachstellen analysiert und entsprechende Vorschläge für Payloads und gleichzeitig verfügbare Patches liefert. Um die Effizienz dieses Systems zu messen, wurde ein Leistungsfaktor E berechnet. Dieser setzt sich aus der benötigten Zeit für den ersten Scan sowie dem des letzten Scans zusammen. Wie Experimente zeigen, nimmt die Dauer der Scans mit BTB ab, während Überprüfungen mit Scannern ohne Machine Learning Komponenten in der Dauer konstant bleiben.
Mit dieser Forschung wurde gezeigt, dass durch Machine Learning schnellere Ergebnisse bereitgestellt werden können.

4.3.11. Erkennung von Malware anhand von PE-Header (2017)

Raff, Sylvester und Nicholas (2017) verzichten in ihrem Ansatz auf explizite Feature Konstruktionen und zeigen, dass Malware anhand von Bytes von neuronalen Netzen erkannt werden kann. Für diese Analyse untersuchten sie zwei verschiedene Ansätze, ein Fully Connected Neural Network (FC Neural Network) und ein RNN, bei welchem sie sich für das LSTM Model entschieden. Gerade deshalb war es nötig sich bei der Analyse auf den Header der PE-Dateien zu konzentrieren, da LSTM Modelle für die Berechnung aller Daten enorm viel Zeit und Ressourcen in Anspruch nehmen würden (Raff, Sylvester und Nicholas 2017). Die dabei verwendeten Features bestehen aus 328 geordneten Bytes. Zusätzlich wurden Modelle entwickelt um Vergleiche mit den Ergebnissen der Neuronalen Netze anzustellen. Dazu gehören Extra Random Trees (ET), RF und LR. Um die Vorhersagen zu evaluieren wurden die Metriken Genauigkeit sowie Area Under the Curve (AUC) verwendet. Entsprechende Daten sammelten die Forscher sowohl bei *VirusShare* als auch bei *OpenMalware*. Wie die Ergebnisse zeigen liegt der vielversprechendste Ansatz in FC Neural Network mit einer Genauigkeit von 89.9% gefolgt von dem RNN LSTM mit 79.7%.
Durch diese Forschung zeigen die Autoren das Potenzial neuronaler Netze im Erkennen von Schadsoftware lediglich anhand von rohen Bytes. Dies impliziert nicht nur eine enorme Zeit sondern auch eine bemerkenswerte Ressourcen Einsparung.

4.3.12. Erkennung von Malware anhand von PE-Header mit erweitertem Feature-Set (2017)

Im Gegensatz zu dem zuvor erläuterten Ansatz von Raff, Sylvester und Nicholas (2017), verwenden Kumar, Kuppusamy und Aghila (2017) ein erweitertes Feature Set für ihre Analyse. Dieses setzt sich aus rohen Features, wie beispielsweise der Anzahl der `FILE_HEADER`-Abschnitte, und abgeleiteten Features zusammen. Bei Letzterem handelt es sich um Werte, die durch den Abgleich von Feature-Werten mit vorab definierten Regeln entstehen. Beispielsweise könnte der Wert der Kompilierungszeit, bei welchem es sich um eine Ganzzahl, die die vergangene Zeit seit 1969 in Sekunden angibt, handelt, allein wenig Aussagekraft besitzen (Kumar, Kuppusamy und Aghila 2017). Deswegen wird die Zahl in ein Datumsformat konvertiert und mit einem bestimmten Gültigkeitsbereich verglichen. Dadurch ergibt sich ein boolescher Wert, welcher in die Analyse mit einfließt. Kumar, Kuppusamy und Aghila (2017) verwenden insgesamt 11 abgeleitete und 55 rohe Features. Für ihre Analyse sammelten die Autoren bösartige Dateien bei *VirusShare* und *download.cnet*. Es wurden Experimente anhand roher Feature Sets als auch anhand eines integrierten Feature Sets (welches aus rohen und abgeleiteten Features besteht)

mit LR, Linear Discriminant Analysis (LDA), RF, DT, NB und k-NN durchgeführt. Als Evaluationsmetriken wurden Genauigkeit, Precision, Recall und der F-measure verwendet. Bis auf k-NN erzielten alle Algorithmen ein besseres Ergebnis, wenn das integrierte Feature Set verwendet wurde. Den besten Wert erreicht RF mit einer Genauigkeit von 89.23% und einem F-measure von 0.90.
Diese Ergebnisse erzielen somit eine höhere Genauigkeit als die der Untersuchungen von Raff, Sylvester und Nicholas (2017), jedoch ist eine weitaus intensivere Vorarbeit zu leisten.

4.3.13. Erkennung von Exfiltration und C&C Tunnels (2017)

Das u. a. (2017) entwickelten ein auf Machine Learning basierendes System um die Exfiltration von Daten eines kompromittierten Systems, sowie den Aufbau eines Command & Control (C&C)-Servers zu erkennen. Um Daten zu exfiltrieren kann Schadsoftware DNS Abfragen nutzen. Dazu kodiert und/oder komprimiert sie die zu versendenden Informationen zunächst. Anschließend kann die daraus entstandene Zeichenkette als Subdomain einer DNS Abfrage angehängt werden.
Über DNS TXT-Records können Texte anstatt einer IP Adresse an einen Benutzer gesendet werden. Angreifer können diese Funktion nutzen, um einen Tunnel zum Senden von Anweisungen oder zum Öffnen einer Sitzung einzurichten. Bei der Exfiltration gehen die Autoren davon aus, dass eine kodierte Subdomain ein Indiz hierfür ist, weshalb diese Zeichenkette analysiert wird. Zunächst generieren sie aus dieser acht Features wie beispielsweise dem Verhältnis der Anzahl an Ziffern zum Rest der Elemente. Das von Das u. a. (2017) entworfene Model basiert auf Logistic Regression und erreicht einen F-measure von 0.96.
Für die Klassifizierung von TXT-Records wählten sie einen *unsupervised learning* Ansatz, d.h. das Modell arbeitete mit einem ungelabelten Datenset. Für die Analyse verwendeten sie zehn Features, unter anderem die Anzahl der Groß- sowie der Kleinbuchstaben, Punkte oder Unterstriche sowie die Anzahl von Ziffern. Dadurch konnten von 2356 bösartigen TXT-Records 2160 von ihrem Modell identifiziert werden.

4.3.14. Erkennung bösartiger PowerShell-Befehle (2018)

Hendler, Kels und Rubin (2018) untersuchten in ihrer Forschung von 2018 bösartige PowerShell Befehle. Dazu analysierten sie ein Datenset welches aus über 66.000 Befehlen bestand. Die hierbei verwendeten bösartigen Befehle stammen von Microsoft-Sicherheitsexperten. Das Datenset musste zunächst vorverarbeitet werden.

Dazu wurden beispielsweise kodierte Befehle zunächst dekodiert, Leerzeichen wurden entfernt und Nummern durch ein Sternchen (*) ersetzt. Anschließend wurden die Daten sowohl mit einem CNN als auch mit einem auf NLP basierenden Detektoren untersucht. Das beste Ergebnis erzielte allerdings ein Ensemble-Detektor, der einen NLP-basierten Klassifikator mit einem CNN-basierten kombiniert. Dieser erzielte eine TPR von 0.92.
Da das verwendete Datenset nicht einsehbar ist, bleibt hier unklar, was einen PowerShell Befehl bösartig macht. Da viele Befehle von Administratoren als auch von Angreifern gleichermaßen genutzt werden, wäre dies ein interessantes Detail gewesen.

4.3.15. Klassifizierung von Netzwerkverkehr in 5 Klassen (2018)

Ding und Zhai (2018) wählten den Ansatz eines CNN um ein IDS aufzubauen, da laut ihnen Systeme, welche mit traditionellen MLAs arbeiten nicht explizit und nicht zuverlässig genug sind. Um diese These zu verifizieren verglichen sie die Ergebnisse des CNN mit traditionellen Algorithmen wie RF und SVM und Deep Learning Methoden wie Deep Belief Network (DBN) und LSTM. Wie Yin u. a. (2017), verwendeten sie das NSL-KDD Datenset, welche dieses bereits mit RNN analysierten. Auch Ding und Zhai (2018) wählten als Leistungsindikator die Genauigkeit, sowie die TPR und die FPR. Tatsächlich lag die Genauigkeit des CNN mit 80.13% deutlich über der, der traditionellen als auch der Deep Learning Methoden. Jedoch erzielten Yin u. a. (2017) mit ihrem RNN eine um knapp 2% bessere Analyse bei der Klassifikation des Netzwerkverkehrs in 5 Klassen. Da laut Yin u. a. (2017) RNNs deutlich mehr Trainingszeit benötigen als traditionelle MLAs, wäre es interessant zu vergleichen gewesen, wie sich die Trainingszeit bei dem von Ding und Zhai (2018) entwickelten CNN verhielt. Leider ist dieses Detail in der Arbeit nicht dokumentiert.

4.3.16. Anomalieerkennung anhand von Systemprotokollen (2018)

Computersysteme produzieren täglich Terabytes an Daten, welche unmöglich manuell inspiziert werden können. Dadurch scheint dies ein für MLAs prädestinierter Use Case zu sein. Brown u. a. (2018) untersuchten in ihrer Arbeit die Analysefähigkeit von RNNs in Systemprotokollen. Da die Autoren auf unbeaufsichtigte Lernmethoden setzen, kann auf die zeitaufwändige Beschaffung von gelabelten Daten verzichtet

werden. Um das von ihnen entwickelte Modell zu evaluieren, verwendeten sie das Los Alamos National Laboratory (LANL) (Kent 2015), welches Host Ereignisse protokolliert. Dieses besteht aus über einer Milliarde Protokollzeilen, welche jeweils 21 Features beinhalten. Von diesen verwendeten die Autoren lediglich acht für ihre Analyse und reduzierten den Datensatz auf 14 Millionen Protokollzeilen, da sie nur mehr zwei Tage der Aufzeichnungen berücksichtigten. Um die Ergebnisse zu überprüfen verwendeten sie die ROC AUC. Dieser ergab einen AUC Wert von 0.97 und zeigt deutlich, dass die Analyse mit unbeaufsichtigten RNNs ein hohes Potenzial für die Untersuchung von Protokolldateien besitzt.

4.3.17. Erkennung von bösartigem Netzwerkverkehr (2018)

Aldwairi, Perera und Novotny (2018) verwendeten in ihrer Arbeit eine Restricted Boltzmann Machine (RBM) um Netzwerkverkehr binär nach *normal* oder *anormal* zu klassifizieren. Im Vergleich zu Neuronalen Netzen die aus input, hidden und output Schicht bestehen, enthält eine RBM lediglich die erste beiden Schichten. Im Vergleich zu Yin u. a. (2017) und Ding und Zhai (2018), verwenden Aldwairi, Perera und Novotny (2018) nicht das NSL-KDD Datenset, sondern das ISCX Datenset (Shiravi u. a. 2012), in welchem Netzwerkverkehr aus dem Jahr 2012 aufgezeichnet wurde. Für ihre Analyse verwendeten die Forscher 16 Features die verschiedene Merkmale des Netzwerkverkehrs beschreiben. Um die Leistung der RBM zu evaluieren verwendeten auch sie den Leistungsindikator der Genauigkeit und erzielten dabei einen Wert von 89%. Außerdem vermerkten sie eine TPR und eine False Negative Rate (FNR) von jeweils 0.88. Da Aldwairi, Perera und Novotny (2018) lediglich eine zweischichtige RBM verwendeten, wäre es interessant weitere Experimente mit so genannten Deep Restricted Boltzmann Machines (DRBMs), welche aus mehreren Schichten bestehen, durchzuführen.

Z. Wang (2015) haben 2015 einen Ansatz aus der Industrie auf der Blackhat Konferenz 2015 vorgestellt, um Netzwerkverkehr zu klassifizieren. Dafür verwendeten sie hauptsächlich Artificial Neural Networks (ANNs). Intern wurden 300.000 TCP Verkehrsdaten gesammelt, welche anschließend analysiert wurden. Dadurch konnten 58 Protokolltypen wie SSL, MySQL, SMB, Kerberos, DNS und Lightweight Directory Access Protocol (LDAP) identifiziert werden. Für jedes der Protokolle konnte eine Precision von über 0.9 erzielt werden. Lediglich 17% der Verkehrsdaten konnten nicht klassifiziert werden und erhielten daher das Label „unknown“.

4.3.18. Erkennung von Botnetzen (2018)

Botnetze sind eine Sammlung an kompromittierten, miteinander verbundenen Maschinen, die durch einen Master gesteuert werden. Diese Netze können Bedrohungen wie DoS Attacken, Spamming oder Datendiebstahl auslösen. Laut Leonard, Xu und Sandhu (2009) besteht der Lebenszyklus eines Botnetzes aus vier Phasen: Formation, C&C, Attacke, post Attack-Phase. Um diese Art von bösartigem Verhalten zu analysieren, entwickelten Mathur, Raheja und Ahlawat (2018) ein Modell, welches Botnetze in den ersten beiden dieser Phasen erkennt. Um Attacken so früh wie möglich zu erkennen, setzen die Forscher auf ein zeitsparendes Analyseverfahren bei welchem lediglich der Header von TCP/User Datagram Protocol (UDP)-Paketen untersucht wird. Um entsprechende Daten zu generieren wurde Netzwerkverkehr von Linux als auch von Windows Systemen erfasst. Zusätzlich verwendeten die Forscher das CTU-13 (Garcia u. a. 2014), sowie das ISOT (Victoria 2010) Datenset. Insgesamt wurden 11 Features wie beispielsweise Fließdauer, Ziel und Quell IP Adresse sowie das zu verwendete Protokoll verwendet. Bei der Analyse kamen die Algorithmen LR, Random SubSpace, Randomizable Filtered Classifier, MultiClass Classifier und Random Committee zum Einsatz. Dabei erzielte der MultiClass Classifier eine Genauigkeit von 98.4% sowie eine FPR von lediglich 0.004 in einer Zeit von 0.03 s. Somit beweisen Mathur, Raheja und Ahlawat (2018) einen praktikablen Ansatz im Erkennen von Botnetzen, allerdings bemängeln sie die Genauigkeit bei einem exponentiellen Anstieg von Netzwerkverkehrsdaten. Sie empfehlen daher für eine größere Menge an Daten neuronale Netze zur Erkennung zu verwenden.

4.3.19. Klassifizierung von Microsoft Malware (2018)

Sabar, Yi und Song (2018) setzen bei der Erkennung von Malware auf SVM. Jedoch optimieren sie deren Konfiguration durch Verwendung von hyper Heuristiken. Dabei handelt es sich um eine Suchmethode welche unter diversen Heuristiken auswählt und diese gegebenenfalls miteinander kombiniert, um eine bestmögliche problemspezifische Konfiguration zu gewährleisten. Die Daten für die Analyse entnahmen sie der Microsoft Kaggle Challenge von 2015 (Kaggle 2015), bei welcher es 500 GB Schadsoftware Dateien in neun verschiedene Klassen zu klassifizieren galt. Zusätzlich verwendeten sie das NSL-KDD Datenset. Um den von ihnen entwickelten Ansatz zu evaluieren, analysierten sie die Datensätze zusätzlich mit den Algorithmen Fuzzy Classifier (FC) und DT und einer NB Implementierung. Als Leistungsindikatoren wählten sie die Genauigkeit sowie den *logloss* (dt. Logarithmischer Verlust), welcher die Leistung eines Modells misst, dessen Ausgabe einen Wahrscheinlichkeitswert zwischen 0 und 1 annehmen kann. Dabei steigt der Wert wenn die Annahme von dem tatsächlichen Wert abweicht und sinkt wenn er mit diesem übereinstimmt.

Dementsprechend ist ein Wert welcher gegen 0 strebt wünschenswert. Tatsächlich erzielte der Ansatz von Sabar, Yi und Song (2018) den niedrigsten logloss mit einem Wert von 0.0031 [1]. Des Weiteren konnte ebenfalls damit die höchste Genauigkeit von 85.69% erreicht werden. Dadurch konnten die Forscher die Effektivität ihres Ansatzes beweisen.

4.3.20. Klassifizierung von Malware anhand von Datenpaketen (2018)

Mit ihrem Ansatz möchten Yeo u. a. (2018) die Schwächen von portbasierten Ansätzen und Deep Packet Inspection (DPI) kompensieren. Laut ihnen ist der portbasierte Ansatz nicht verlässlich bei unbekannten Ports und DPI (Dharmapurikar u. a. 2003) führt zu einer zu zeit intensiven Analyse, welche für große Datenmengen nicht praktikabel ist. Für ihre Arbeit verwendeten sie ebenfalls das CTU-13 Datenset (Garcia u. a. 2014), welches Malware Datenpakete vor, sowie nach einer Infektion eines Windows XP Systems beinhaltet. Die Daten wurden den sechs Klassen *Neris, rbot, Virut, Murlo, NSIS* und *normal* zugeordnet. Insgesamt wurden 35 Features aus den Netzwerkpaketen extrahiert. Dabei wurde beispielsweise die Größe der jeweiligen Pakete sowie die Dauer der Übertragung untersucht. Für die Analyse verwendeten sie ein CNN sowie die Klassifikatoren MLP, SVM und RF. Um die Leistung der jeweiligen Algorithmen zu evaluieren, wurde die Genauigkeit, Precision und Recall verwendet. Dabei wurde deutlich, dass die besten Ergebnisse sowohl durch RF mit einer Genauigkeit von 93% als auch durch Analysen mit einem CNN mit 85%iger Genauigkeit erzielt werden können.
Leider gingen die Forscher nicht auf ihre anfangs aufgestellte These ein und erläuterten nicht, ob ihr Ansatz nun effektiver und zeitsparender ist als die portbasierte bzw die DPI Analyse.

4.3.21. Erkennung von Port-Scans (2018)

Aksu und Ali Aydin (2018) konzentrierten sich in ihrer Arbeit auf die Erkennung von Port-Scans. Dafür verwendeten sie das CICIDS2017 Datenset (Sharafaldin, Habibi Lashkari und Ghorbani 2018), welches vom *Canadian Institute for Cyber Security* entwickelt wurde. Dieses besteht aus insgesamt knapp 290.000 Aufnahmen von Netzwerkverkehr, wobei jede über 85 Features wie Quell IP, Quell Port, Ziel Port, Dauer und weitergeleitete Pakete verfügt. Um Port-Scans zu identifizieren

[1] Das Gewinner Team der Kaggle Challenge erzielte einen logloss von 0.0028 (*Microsoft Malware Classification Challenge (BIG 2015) Leaderboard* 2019).

wurden Deep Learning Algorithmen, sowie eine SVM verwendet. Die Leistung wurde anhand von Precision, Recall, Genauigkeit und dem F-measure evaluiert. Für ihre Analyse verwendeten Aksu und Ali Aydin (2018) alle Features des Datensets und verzichteten somit auf zusätzliche Feature Auswahl Algorithmen. Dadurch erzielten sie eine Genauigkeit von 69.8% und einem F-measure von 0.65 mit der SVM. Allerdings konnten sie diese Ergebnisse mit Hilfe von Deep Learning Algorithmen deutlich verbessern, dadurch wurde eine Genauigkeit von 97.80% und ein F-measure von 0.99 erreicht.

4.3.22. Erkennung von Netzwerkverkehr (2018)

Teoh u. a. (2018) wenden Deep Learning an, um bösartigen Netzwerkverkehr zu identifizieren. Bei ihrer Analyse setzen sie auf die Klassifikation mit MLP. Durch ihre Forschung soll gezeigt werden, dass die Zukunft von Malware Erkennung in Deep Learning liegt. Als Grundlage für das Experiment diente das Advanced Security Network Metrics & Non-Payload-Based Obfuscations (ASNM-NPBO) Datenset (Homoliak 2016), welches aus legitimer und bösartiger TCP Kommunikation besteht. Das Datenset führt zwei Arten von Kennzeichnung auf. Zum einen wird der Verkehr binär in *gutartig* oder *bösartig* klassifiziert, zum anderen werden die Daten drei Klassen zu geordnet: *gutartig*, *direkte Attacke* oder *verschleierte Attacke*. Teoh u. a. (2018) beschränkten sich bei ihrer Untersuchung auf das binär klassifizierte Datenset, welches aus circa 9000 Aufzeichnungen besteht. Von den knapp 900 Attributen verwendeten die Forscher lediglich 15 für ihre Analyse. Anschließend wurden zwei Modelle entwickelt: MLP und J48. Letzterer erzielte eine Genauigkeit von 99.35%. Mit Hilfe des MLP wurden lediglich 15(!) Daten analysiert, welche eine Genauigkeit von 100% aufweisen. Da dies aber keine repräsentative Anzahl an Daten ist kann über die Effizienz einer Analyse mit MLP sowie über die zu Beginn aufgestellte These - die Zukunft von Malware Erkennung liegt in Deep Learning - keine relevante Aussage getroffen werden.

4.3.23. Erkennung bösartiger SQL-Abfragen (2018)

Jayaprakash und Kandasamy (2018) stellen in ihrer Arbeit einen Anomalie basierten Ansatz vor, um ein IDS für SQL Datenbanken zu entwickeln. Laut Jayaprakash und Kandasamy (2018) reichen bisherige signaturbasierte Lösungen hierfür nicht aus, da diese lediglich auf bekannte Signaturen reagieren und dadurch neuartige Angriffe nicht erkennen. Bei ihrer Analyse möchten sie Angriffe von innerhalb einer Organisation als auch von außerhalb erfassen. Dabei setzen sie auf eine Datenstruktur welche aus einer Beziehung von acht Arrays besteht. In diesen Arrays

werden SQL Abfragen entsprechend repräsentiert. Für die Analyse verwendeten sie den Naïve Bayes Klassifikator, welcher Daten in Klassen, entsprechend der vorhandenen Benutzerrollen zuteilt und die Protokolldatei als Eingabe verwendet. Dabei wird ein Profil erstellt, welches mit bisherigen Profilen verglichen, und mit einem Punktesystem von 0 bis 10 bewertet wird. Befindet sich der dabei ermittelte Schweregrad über 8.0 wird der Administrator alarmiert. Liegt der Wert zwischen 6.0 und 8.0 wird die Abfrage geblockt. Andererseits wird die Abfrage ausgeführt. Das Modell wurde mit 239 Abfragen getestet. Dabei konnten 59.92% korrekt klassifiziert werden.

4.3.24. Erkennung von LDDoS Attacken (2018)

In ihrer Arbeit stellen Siracusano, Shiaeles und Ghita (2018) eine Methode vor um LDDoS Angriffe zu erkennen. Bei dieser Art von Angriff handelt es sich im Gegensatz zu DoS Attacken, bei welchen ein Server mit Anfragen geflutet wird, um Anfragen welche sehr langsam und nacheinander an einen Host gesendet werden. Dadurch hält der Server Ressourcen bereit die auf den Rest der Nachricht warten, wodurch andere Benutzeranfragen nicht abgearbeitet werden können. Um LDDoS Attacken zu erkennen, verwenden sie TCP-Verbindungsparameter. Diesbezügliche Daten extrahierten sie aus einem eigens zu diesem Zweck simulierten Netzwerk aus Clients, Angreifern und einem Webserver. Zusätzlich verwendeten sie das CIC Datenset (Jazi u. a. 2017), welches DoS Attacken auf der Anwendungsebene beinhaltet. Diverse Analysen wurden mit LR, k-NN, SVM, DT, RF und Deep Neural Network (DNN) durchgeführt. Die Ergebnisse zeigen das besonders Analysen mit k-NN und DT sehr effizient sind. Alle Modelle erreichten eine Genauigkeit von 95%. Die Analyse anhand eines Decision Tree erreichte nicht nur eine FPR und einer FNR von 0, sondern konnte zudem mit einer Evaluierungszeit von 0.019 s am schnellsten durchgeführt werden. Somit wurde nicht nur gezeigt, dass eine Analyse von TCP Daten mit MLAs erfolgreich sein kann, sondern auch welcher Algorithmus sich am besten dafür eignet.

4.3.25. Klassifizierung von Wi-Fi Netzwerkdaten (2018)

Qin u. a. (2018) verwendeten für ihre Analyse das Aegean WiFi Intrusion Dataset (AWID) welches Angriffe auf kabellose Netzwerke beinhaltet. In ihrer Analyse klassifizierten die Forscher die Daten nach Flooding oder Injection Attacken sowie nach normalem Netzwerkverkehr. Für ihre Analyse verwendeten sie 18 von 154 möglichen Features wie zum Beispiel der Dauer der Verbindung, das Zeitdelta vom letzten erfassten Frame, die Datenrate sowie die Quelladresse. Sie wählten eine SVM

um die Daten zu analysieren. Da Support Vector Machines (SVMs) grundsätzlich nur binär klassifizieren wurde eine Methode verwendet die eine zusätzliche Bibliothek benötigt um eine Multi-Klassen Klassifikation durchführen zu können. Hierbei werden mehrere SVMs verwendet um Daten zu klassifizieren. Gibt es k Klassen, werden $k(k-\frac{1}{2})$ SVMs für die Analyse verwendet (Qin u. a. 2018). Diese Methode erzielte eine Genauigkeit für Flooding Attacken, Injection Attacken und normale Daten von 89.18%, 87.34%, und 99.88%.

4.3.26. Klassifizierung von verschleierter Malware (2019)

Han u. a. (2019) möchten mit ihrem Ansatz das Problem der Schadsoftwareverschleierung lösen. Diese Technik wird von Malware mehr und mehr verwendet um einer Erkennung zu entgehen (Li, Peng u. a. 2016). Zu diesen Techniken zählen Packing, Metamorphismus und Polymorphismus. Bei ersterem handelt es sich um komprimierten Schadcode, welcher zunächst entpackt werden muss um diesen analysieren zu können. Bei Polymorphismus wird eine Technik angewandt bei welcher der Binärcode verschlüsselt und mutiert wird. Dabei wird sobald der Code in den Speicher geladen wird, eine neue Version desselben generiert, wodurch divergierende Signaturen für denselben Code entstehen können. Metamorphismus ist ein weiterer Verschlüsselungsansatz bei welchem die Opcode Sequenz bei jedem Programmstart geändert wird. Dies erschwert dem Detektor ein stabiles Featureset für die Malware zu erstellen. Weitere Informationen zu diesen Techniken können P. He u. a. (2017) entnommen werden.
Um Daten zu generieren setzen die Forscher auf eine dynamische Analyse der Malware. Das bedeutet die Schadsoftware wird in einer gesicherten Umgebung ausgeführt. Aus den dadurch erfassten API und Dynamic Link Library (DLL) Informationen, der Interaktion mit Dateien und dem Netzwerk, sowie Informationen aus den PE-Dateien, werden Features abgeleitet, mit welchen der Klassifikator trainiert wird. Dadurch entwickelten sie folgende Modelle: DT, RF, k-NN und Extreme Gradient Boosting (XGBoost). Als Leistungsindikatoren wählten sie Genauigkeit, Precision, Recall und F-measure. Durch diverse Experimente konnten sie nachweisen, dass durch die Analyse der Informationen bezüglich der Interaktion mit Dateien, dem Netzwerk und der Registry durch RF eine Genauigkeit von 97.21% erzielt werden kann. Dadurch konnte eine verlässliche Analyse selbst bei verschleierter Malware nachgewiesen werden.

4.3.27. Klassifizierung von Malware - Imageanalyse (2019)

Um Polymorphismus, Metamorphismus und Packing mit traditionellen MLAs zu erkennen ist ein umfangreiches Feature Engineering, sowie beträchtliche Kenntnisse auf Domain-Ebene nötig (Rhode, Burnap und Jones 2018). Zudem können Angreifer der automatischen Malware Erkennung entgehen sobald sie die verwendeten Features kennen (Anderson, Kharkar u. a. 2017). Diesen Problemen wollen Vinayakumar u. a. (2019) mit ihrem Deep Learning Ansatz begegnen.
Dazu verglichen sie klassische Algorithmen für maschinelles Lernen mit Deep Learning Architekturen. Die Vergleiche basieren auf statischen und dynamischen Analysen, sowie auf Bildverarbeitungstechnologien.
Für die statische Analyse verwendeten sie privat gesammelte Proben sowie das Ember Datenset (Anderson und Roth 2018), welches aus je rund 70.000 bösartigen und gutartigen Dateien besteht. Mit Hilfe von LR, NB, k-NN,DT, AB, RF, SVM, Light Gradient Boosting Machine (LightGBM) sowie DNN und CNN entwickelten sie einen hybriden Ansatz um Malware statisch zu klassifizieren. Die Ergebnisse zeigen, dass Deep Neural Networks (DNNs) eine höhere Genauigkeit (98.9%) als traditionelle MLAs (LightGBM: 97.5%) erreichen.
Auch bei der dynamischen Analyse konnten Deep Learning Architekturen klassische MLAs übertreffen. Für diese Art Analyse wurden Daten in einer Cuckoo Sandbox generiert. Das beste Ergebnis erzielte hierbei ein CNN mit einem AUC von 0.99. Als drittes Experiment wurde eine Imageanalyse durchgeführt, wobei Malware Dateien als grau skalierte Bilder dargestellt werden. Für diese Analyse verwendeten sie das Malimg Datenset (Nataraj u. a. 2011), welches knapp 10.000 Malware Bilder beinhaltet, sowie privat gesammelte Proben. Bei Experimenten wurde deutlich, dass Analysen mit LSTM, mit einer Genauigkeit von 96.3% die höchste Effizient bieten.
Wie sich zeigte ist die Imageanalyse schneller als die statische und die dynamische Analyse, da diese auf Rohdaten basiert, unabhängig von Packing ist und komplett auf Zerlegung oder Ausführung von Code verzichtet.

4.3.28. Erkennung von FF Netzwerken (2019)

Um böswillige Netzwerkangriffe wie DDoS, Phishing und Spaming zu verschleiern, setzen Angreifer vermehrt auf Fast-Flux (FF). Durch diese Technologie entsteht ein sich ständig änderndes Netzwerk kompromittierter Hosts die als Proxy dienen. Durch die schnelle Änderung der IP Adresse des Kontrollterminals werden solche Angriffe häufig nicht erkannt, da IP Blacklists hierbei nicht funktionieren. Zudem erschwert die Analogie zu Content Distribution Networks (CDNs) einer Differenzierung dieser beiden Arten von Netzwerken. Diesem Differenzierungsproblem haben

sich Chen u. a. (2019) in ihrer Arbeit angenommen. Als eines der Features verwenden sie den Domain Namen. Dieser ist in FF Netzen schlecht leserlich, da die Reihenfolge von Konsonanten und Vokalen unregelmäßig und zudem mit Nummern vermischt ist. Außerdem sind hierbei die Domain Namen oft länger als in CDNs. Als weiteres Feature wird der CNAME verwendet, welcher einer Domäne als zusätzlichem Namen bzw Alias dient. Zusätzlich wird der A-Record, welcher einer Domain eine feste IP Adresse zuteilt, als Feature verwendet. Im Gegensatz zu CDNs sind FF Domain Namen kurzlebiger und weisen einen geringeren Netzwerkverkehr auf, daher wurde auch dieses Merkmal als Feature aufgenommen. Als zusätzliches Feature werden geographische Unterschiede der Ergebnisse von Domänen verwendet. In CDNs werden Knoten in verschiedenen geographischen Gebieten bereit gestellt. Deswegen bekommen Nutzer die weit voneinander entfernt sind unterschiedliche Auflösungen als Ergebnis, wenn sie Domain Namen abfragen. In FF Netzen werden jedoch immer dieselben Ergebnisse geliefert, da diese über eine viel kleinere Anzahl von IP Adressen verfügen. Ihre Erkennungsmethode basiert auf einem LSTM dessen Effizienz sie anhand der Genauigkeit, dem Recall und dem F-measure gemessen haben. Die Analyse erzielte bei einer Trainingszeit zwischen 35 und 75 Sekunden eine Genauigkeit und einen F-measure von über 0.95.

4.3.29. Erkennung von drive-by Download-Attacken bei Twitter (2019)

Das Kürzen von Links innerhalb von Tweets hat den Vorteil, dass Nutzer auch in Anbetracht der vorgegebenen Länge von 140 Zeichen pro Tweet, lange URLs tweeten können. Dieses Feature birgt allerdings gleichzeitig die Gefahr Opfer eines drive-by Download Angriffs zu werden. Hierbei nutzen Angreifer die Kürzung der Links, um Benutzer über Bilder oder Text auf bösartige Seiten zu locken. Diese Links können dazuführen, dass der Angreifer Fernzugriff zum System des Opfers bekommt, von welchem er Daten extrahieren oder dessen Computer in ein Botnetz integrieren kann (Provos u. a. 2007). Javed, Burnap und Rana (2019) haben in ihrer Arbeit diese Art von Links untersucht und nach *bösartig* und *gutartig* klassifiziert. Für ihre Analyse sammelten sie Tweets über die Twitter Streaming API, zur Fussball Europameisterschaft von 2016 (#Euro2016) und zu den Olympischen Spielen (#Rio2016) desselben Jahres. Da diese zu den Top Themen dieses Jahres gehörten beinhalteten diese die meisten Tweets. Für ihre Experimente nutzten sie ein Tool, welches jede URL in einer sicheren Umgebung besucht und etwaige Systemlevel Operationen, wie beispielsweise Datei-, Prozess- oder Registry-Änderungen aufzeichnet. Ergeben sich daraus clientbasierte Änderungen wie die Freigabe von Speicher, Startzeit eines Prozesses oder gesendete Bytes, fließen diese als Features

in die Analyse ein. Insgesamt werden dabei 54 Metriken aufgezeichnet. Der zweite Teil des Feature-Sets, setzt sich aus 24 Tweet Attributen wie Benutzernamen, Art des Benutzers, Anzahl an Followern und der Anzahl der Retweets zusammen. Diese insgesamt 78 Features analysierten sie anhand von vier Modellen: Naïve Bayes, Bayesian Network, J84 (Decision Tree) und MLP. Als Leistungsindikatoren verwendeten sie Precision, Recall, F-measure und FPR. Die Modelle wurden mit Daten der Fussball Europameisterschaft trainiert und anschließend mit Daten der Olympiade getestet. Dabei kam ein Votum Meta-Klassifikator zum Einsatz, welcher es erlaubt Ergebnisse mehrerer Klassifikatoren zu verbinden und somit die beste Klassifizierungswahrscheinlichkeit zu generieren. Wie sich zeigte führen Kombinationen aus J48 und NB sowie NB und MLP zu den besten Ergebnissen. Erstere erzielte einen F-measure von 0.86, letztere erreichte einen Wert von 0.75.

Seymour und Tully (2016) stellten auf der Blackhat Konferenz 2016 einen interessanten Ansatz vor, bei welchem sie Twitter nicht wie Javed, Burnap und Rana (2019) nutzen um Angreifer zu finden, sondern um potenzielle Opfer zu identifizieren. Sie entwickelten ein System namens `SNAP_R` mit welchem sie Nutzern anhand von deren Profilen sowie deren Posts eine entsprechende Erfolgswahrscheinlichkeit, um Opfer eines Phishing Angriffs zu werden, zusprechen. Ist diese relativ hoch wird der Benutzer als Ziel eingestuft und `SNAP_R` sendet diesem automatisch Inhalte mit einem eingebetteten Phishing Link. Dabei wählt das System ein Thema aus zu dem der Benutzer kürzlich getweetet hat und sendet die Antwort zu der Zeit zu welcher der Benutzer am häufigsten tweetet. Seymour und Tully trainierten ein neuronales Netz, welches die Tweets generiert, mit Pentesting Daten, Reddit Beiträgen und Tweets. Um ihr Modell zu evaluieren platzierten sie Links in einem Tweet. Wurde dieser geklickt, zeichneten sie den Zeitstempel, den User-Agent sowie den Benutzernamen auf. Bei Tests die aus 90 Benutzern bestanden, konnte ihr Modell eine Erfolgsrate zwischen 30% und 66% aufweisen.
Seymour und Tully haben somit ein System entwickelt, welches es Penetrationstestern ermöglicht eine große Zielgruppe im Zuge von Penetrationstests anzusprechen und so ein besseres Verständnis für Phishing Angriffe zu fördern und Benutzer dahingehend zu sensibilisieren.

4.3.30. Erkennung von DGA Domains (2019)

Der Domain Generation Algorithm (DGA) ist ein Algorithmus, mit dem Domainnamen generiert werden, die häufig von Malware verwendet werden, um domainbasierten Firewall-Steuerelementen auszuweichen. Dadurch können C2 Server verschleiert werden.
Li, Xiong u. a. (2019) fokussieren sich in ihrer Arbeit darauf DGA Domains zu

identifizieren. Für ihre Analyse verwendeten sie Feed-Listen (Bambenek 2019) welche DGA-generierte Domänen beinhalten, die von Malware genutzt werden. Diese Listen sammelten sie über einen Zeitraum von einem Jahr. Für die Analyse der DGA Domains betrachteten sie jede Domain als Zeichenkette, aus welcher sie zwei Arten von Features extrahierten: sprachliche Features und DNS-Features. Zu den sechs sprachlichen Features gehören beispielsweise die Länge, das Verhältnis bedeutungsvoller Wörter, sowie der Prozentsatz numerischer Zeichen. Unter den 27 DNS-Features befinden sich beispielsweise Erzeugungszeit und Ablaufdatum, da DGA-Domains zeitnah erstellt werden und nur eine sehr kurze Zeit gültig sind. Ihre Analyse besteht aus zwei Teilen: zunächst werden die Domains nach *normal* oder *DGA* klassifiziert. Dafür wurden die folgenden Modelle entwickelt: J48, ANN, SVM, LR, NB, Gradient Boosting Tree (GBT) und RF. Anhand der Ergebnisse wurde deutlich, dass durch die Decision Tree Implementierung J48 die besten Werte mit einer Genauigkeit von 95.89% erzielt werden können, weshalb dieser Algorithmus als endgültiger Klassifikator gewählt wurde. Im Anschluss an diese erste Analyse wurden Domains, welche der Klasse DGA angehörten anhand des DBSCAN Algorithmus entweder in eines von drei Malware Cluster, oder dem Cluster für *normale* Domains zugewiesen. Hierbei beträgt die Durchschnittsgenauigkeit 87.64%. Zusätzlich entwickelten sie Deep Learning Modelle und verglichen diese mit dem erfolgreichsten Machine Learning Algorithmus der vorherigen Experimente, J48. Dabei fanden sie heraus, dass gerade bei einer hohen Anzahl an Daten ein LSTM Modell die höchste Genauigkeit mit 98.77% liefert. Dadurch konnten sie die Effizienz von DNNs auch bei der Erkennung von DGA-Domains nachweisen.

4.3.31. Erkennung von Phishing Websites (2019)

Phishing zielt im Vergleich zu anderen Attacken nicht darauf ab Schwachstellen im System auszunutzen, sondern durch gezielte Irreführung und Täuschung des Benutzers, an dessen sensitive Informationen wie Benutzernamen und Passwörter zu gelangen. In der Forschung gibt es momentan vier Verfahren, um Phishing Websites zu erkennen: Blacklists, Heuristiken, Inhaltsanalysen und Machinelles Lernen (Alswailem u. a. 2019). Blacklists gleichen URLs mit bekannten Phishing Websites ab, Heuristiken verwenden Signaturdatenbanken bekannter Angriffe um sie mit der Signatur eines heuristischen Musters abzugleichen. Inhaltsanalysen versuchen Phishing Websites mit Hilfe bekannter Algorithmen wie TF-IDF zu identifizieren. Darüber hinaus können solch eine Art von Website anhand von Alexa erkannt werden (L. A. T. Nguyen u. a. 2013).
Der im Folgenden beschriebene Ansatz von Alswailem u. a. (2019) verwendet Machine Learning Verfahren, um Phishing Websites zu erkennen.

Für ihre Analyse sammelten sie 12000 Phishing URLs bei PhishTank (2019). Zusätzlich beschafften sie 4000 legitime URLs. Zunächst generierten sie 36 Features aus den gesammelten URLs wie beispielsweise deren Länge, Anzahl an Sonderzeichen wie : ; % & ? +, die Einstufung nach Alexa sowie die Anzahl an Formularen, an Links und an Buttons innerhalb der jeweiligen Seite. Anschließend trainierten sie mit diesen Daten einen RF Klassifikator. Um die effizienteste Kombination an Features zu verwenden, stellten sie Experimente mit verschiedenen Feature-Sets an. Dabei fanden sie heraus das eine Kombination aus 26 Features eine maximale Genauigkeit mit 98.85% und eine minimale Genauigkeit von 53.92% bietet.

Hillary und Joshua (2000) untersuchten in ihrem Ansatz knapp 30 Millionen URLs, welche sie bei PhishTank, CommonCrawl, VirusTotal und Sophos zwischen Januar und März 2017, sammelten. Anhand dieser Daten trainierten sie diverse CNNs. Das beste Ergebnisse erzielte das Modell welches mit Daten von VirusTotal trainiert wurde. Bei ihren Experimenten fanden sie heraus, dass nur eine minimale Verzerrung (*bias*) der Trainingsdaten in Bezug auf die Testdaten, die Genauigkeit von Deep Learning Modellen beeinträchtigen. Selbst bei einer Vielzahl an Daten können Modelle bei der Genauigkeit schwächeln, wenn die Daten die zum Training verwendet werden nicht die Daten imitieren, auf welche sie getestet werden. Laut Hillary und Joshua ist eine Möglichkeit dies zu verhindern, zu erwartende Fehler zu simulieren. Dadurch können Modelle eine genauere Vorhersage treffen.

4.3.32. Erkennung von Insider Bedrohungen (2019)

Nicht nur Bedrohungen von Außen können eine Gefahr für Unternehmen darstellen, immer öfter haben diese auch mit Bedrohungen innerhalb einer Firma durch autorisiertes Personal zu kämpfen (Partners 2019). Diesem Problem stellen sich Le und Nur Zincir-Heywood (2019) in ihrer Untersuchung. Anhand des CERT Datensets (Glasser und Lindauer 2013) evaluierten sie das von ihnen entwickelte System. Dieses Set besteht aus System-, E-Mail- und Webprotokollen sowie zusätzlichen Organisations- und Benutzerinformationen. Diese Daten gruppierten sie in drei Sorten von Features: Häufigeitsfeatures wie die Anzahl gesendeter E-Mails, statistische Features wie die Standardabweichung der E-Mail Größe und Benutzerinformationen. Sie implementierten die Modelle LR, RF und ANN um *normale* und *bösartige* Benutzer zu erkennen. Die besten Ergebnisse erzielte das RF Modell mit einer Precision von 0.99. Dabei wurde deutlich, dass Benutzerdaten den meisten Mehrwert für diese Art Analyse bieten. Doch gerade diese sind aus datenschutzrechtlichen Gründen besonders schwierig zu generieren.

Um Insider Bedrohungen innerhalb von Enterprise-Resource-Planning (ERP)-Systemen

zu erkennen, entwickelte Neyolov (2018) eine Analysemethode. Diese stellte er 2018 auf der Hack In The Box (HITB) Konferenz vor. Zunächst sammelte er Protokolldaten des SAP Auditlogs, welche anschließend normalisiert und somit für machine und Deep Learning Algorithmen anwendbar gemacht wurden. Anhand von über hundert Szenarien wurde eine Richtlinie erstellt, welche *normales* Benutzerverhalten widerspiegeln soll. Anhand dieser kann *anomales* Verhalten identifiziert werden. Die normalisierten Features wurden auf ein LSTM angewendet. Dieses lieferte Aussagen über anomales Verhalten mit einer Genauigkeit von 95%.

4.3.33. Erkennung von bösartigen PDFs (2019)

Jeong, Woo und Kang (2019) untersuchten in ihrer Arbeit nichtexekutierbare Dateien wie PDFs. Die hierfür verwendeten Daten sammelten sie bei diversen Anti-Virus Unternehmen. Zunächst labelten sie diese manuell. Dabei fanden sie heraus, dass alle bösartigen PDFs JavaScript enthalten. Jedoch verwendeten sie dies nicht als Feature, da sie für die Analyse ein CNN verwendeten, welches sie ausschließlich mit einer Rohbyte-Sequenz belieferten. Um die Effizienz ihres Modells zu evaluieren, implementierten und testeten sie zusätzlich die Modelle SVM, DT, Naïve Bayes und RF. Als Leistungsindikatoren verwendeten sie Precision, Recall und den F-measure, für jeweils *bösartig* oder *gutartig* klassifizierte Samples. Die Ergebnisse zeigen, dass CNNs mit einer Precision von über 99% traditionellen MLAs, mit der höchsten Precision von 96%, deutlich überlegen sind. Zusätzlich zeigten CNNs durchweg einen besseren F-measure. Des Weiteren testeten Jeong, Woo und Kang diverse CNN Strukturen und fanden dabei heraus, dass eine Embedding Layer in Kombination mit wenigen Convolutional Layers, zuverlässigere Aussagen treffen kann als Kombinationen ohne Embedding Layer. Jedoch benötigt diese Konstellation die höchste Trainingszeit. Der Erfolg ihrer Forschung bestärkt Jeong, Woo und Kang in ihrem Entschluss, diese Art von Analyse auf weitere nichtexekutierbare Dateiformate wie `.rtf` Dateien auszuweiten.

4.4. Ergebnisse der Untersuchung der Analyseverfahren

Die Untersuchung der Analyseverfahren, zeigt einen deutlichen Trend Richtung Malware Klassifizierung. Zwar gibt es auch viele Untersuchungen, welche sich mit der Erkennung von Netzwerkverkehr beschäftigen, diese belaufen sich jedoch insgesamt auf nur sieben. Die zehn Untersuchungen welche, in Abbildung 4.4.1, unter *Sonstiges* zusammengefasst wurden, beziehen sich auf Themen, zu welchen es höchstens zwei Analysen gab. Dazu zählt beispielsweise die Erkennung von Insider Bedrohungen, sowie die Erkennung von drive-by Download Attacken bei Twitter. Das Schlusslicht bildet die gezielte Erkennung von Botnetzen, sowie von DoS Attacken. Abbildung 4.4.1 zeigt, dass sich die Mehrzahl der untersuchten Verfahren mit der Erkennung von Malware beschäftigt.

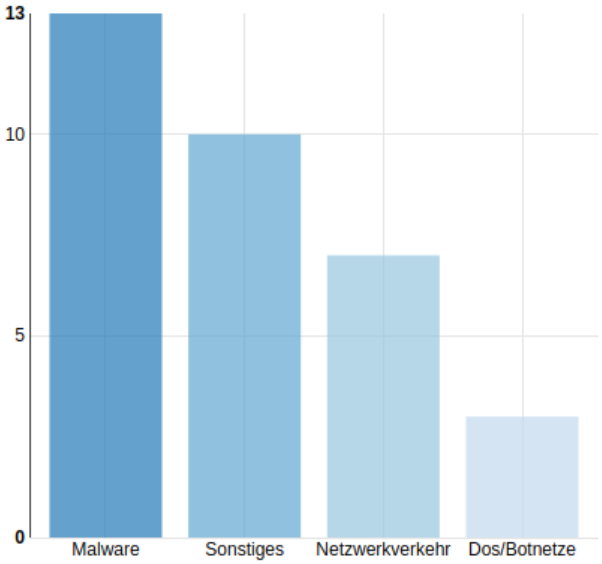


Abbildung 4.4.1.: Verteilung der Analyseverfahren nach Themen (eigene Darstellung)

Bei sieben der 13 Ansätze, bei welchen Malware analysiert wurde, verwendeten die Forscher MLAs. Die restlichen sechs Untersuchungen verwendeten Deep Learning Verfahren. Die besten Ergebnisse konnten dabei mit CNNs und LSTM Netzen erzielt werden.
Aus der Industrie konnten drei konkrete Ansätze gefunden werden. Davon beschäftigt sich einer mit der Klassifikation von Netzwerkverkehr, ein weiterer mit der Klassifizierung potenzieller Opfer von Phishing Angriffen über Twitter und ein

zusätzlicher Ansatz mit der Erkennung bösartiger URLs. Bei letzterem wurden ebenfalls Deep Learning Verfahren verwendet.
Zusammenfassend lässt sich sagen, dass es schwierig ist konkrete Ansätze aus der Industrie ausfindig zu machen. Jedoch wird das Thema Machine Learning im Kontext von Cyber Security vermehrt diskutiert (Pinto 2014), (Polyakov 2018), (Grieco 2016), (Chio 2017).
Als Ergebnis der Business Understanding Phase, kann festgehalten werden, dass Bedarf an weiterer Forschung im Bereich Malware Klassifizierung besteht.

5. Datensätze

Daten sind die Basis jeglicher Analyse, welche mit Hilfe von Machine Learning Verfahren durchgeführt wird. Doch gerade diese sind im Bereich Cyber Security schwer zugänglich und aufwändig zu generieren.
Das liegt unter anderem daran, dass sich in den Daten sowohl Firmeninterna als auch personenbezogene Daten befinden können, welche es zu schützen gilt. Dies bedeutet die Anonymisierung bis hin zur Entfernung eines Teils der Daten. Wie beispielsweise dem Entfernen des *body* in IP Paketen (Uramova u. a. 2018).
Eine weitere Schwierigkeit stellt die Diversität eines Datensets dar. Der Lernerfolg eines Algorithmus basiert hauptsächlich auf den Daten die ihm dazu zur Verfügung stehen. Wird ein Modell also mit einem auf wenige Angriffe reduzierten Datenset trainiert, wird dessen Erkennung beschränkt.
Des Weiteren wächst Malware rapide, was aus der Generierung von Datensets ein dynamisches Unterfangen macht. Denn um auch aktuellste Angriffe erkennen zu können, bedarf es einer kontinuierlichen Aktualisierung eines Datensets.
Dies ist nur ein Teil der Schwierigkeiten, welche sich beim Erstellen von Datensets zeigen, wie Uramova u. a. (2018) heraus fanden.
Eine ebenfalls umfangreiche Untersuchung von Anforderungen an Datensets stellten Gharib u. a. (2016) in ihrer Arbeit an. Dabei kristallisierten sich elf Eigenschaften heraus, welche für ein umfassendes Datenset entscheidend sind. Laut Gharib u. a. (2016) beinhalten diese Angriffsvielfalt, Anonymität, verfügbare Protokolle, vollständige Erfassung, Interaktion, Netzwerkkonfiguration, Datenverkehr, Funktionsumfang, Heterogenität, gelabelte Daten und Metadaten.
Um den momentanen Status Quo der in der Wissenschaft verwendeten Datensets zu analysieren, werden im folgenden Abschnitt die Datensets dokumentiert und untersucht, welche den zuvor beschriebenen Ansätzen als Analysebasis dienten. Zusätzlich sollen die Datensets anhand ihrer Attribute Umfang, Inhalt, Labels und Features miteinander verglichen werden, um eine Aussage über die qualitativen Unterschiede der Sets treffen zu können.

5.1. HTTP Datenset CSIC 2010

Der Datensatz von Carmen Torrano Giménez, Alejandro Pérez Villegas (2010) beinhaltet Web Anfragen zu einer E-Commerce Web Applikation und wurde im Jahr 2010 generiert. Dabei wurden drei Arten anomaler Anfragen berücksichtigt:

- Statische Attacken um an versteckte Ressourcen wie Konfigurationsdateien oder Sitzungs IDs zu gelangen
- Dynamische Attacken wie SQL Injections, XSS und Buffer Overflows
- Unbeabsichtigte illegale Anfragen wie beispielsweise Buchstaben in einer Telefonnummer

Dennoch besteht dieses Set aus lediglich zwei Klassen: *normal* und *anomal*, wobei erstere Klasse 36.000 und letztere 25.000 Samples beinhaltet. Je nachdem, ob es sich bei den Samples um GET oder POST Anfragen handelt bestehen diese aus 11 respektive 13 Features wie dem Host, der Anfrage selbst, Cookies und dem User-Agent. Dieses Datenset besteht aus einem Trainingsset mit normalen Daten und je einem Testset mit normalen und anomalen Anfragen. Es wurde wie bereits in Abschnitt 4.3.4 von Pham, Hoang und Vu (2016) analysiert.

5.2. NSL-KDD

Das aus dem Jahre 2009 stammende NSL-KDD Datenset (Cybersecurity 2019), ist eine Weiterentwicklung des KDD'99 Datensets. Dieses beinhaltete redundante Daten im Trainings- als auch im Testset zudem ist dies mittlerweile veraltet.
Das NSL-KDD Datenset beinhaltet insgesamt über 160.000 Daten an Netzwerkverkehr. Dieses Set segmentiert Angriffe in 4 Klassen: DoS, Probe, U2R und R2L. Dafür bietet es 41 Features wie beispielsweise dem Protokolltyp, der Anzahl fehlgeschlagener Logins und dem Service über welchen kommuniziert wurde. Das komplette Set besteht aus einem Trainingsset welches 67.343 normale und 58.630 anomale Daten beinhaltet und einem Testset mit 9711 normalen und 12.833 anomalen Daten. Zusätzlich bietet es ein besonders schwieriges Testset *Test-21*, welches 2152 normale und 9698 anomale Daten beinhaltet. Jedes Dieser Sets ist als `.txt` und als `.arff` Datei erhältlich. Dieses Datenset wurde bereits wie in den Abschnitten 4.3.5, 4.3.15 und 4.3.19 besprochen von Yin u. a. (2017), Ding und Zhai (2018) und Sabar, Yi und Song (2018) für deren Analysen verwendet.

5.3. LANL

Das Datenset von 2015 der Los Alamos National Laboratory (Kent 2015) bietet umfassende, quellenübergreifende Cybersicherheitsereignisse wie windowsbasierte Authentifizierungsereignisse, Start und Stop Ereignisse von Prozessen, DNS Suchanfragen, Netzwerkdaten und Red Team Events von 58 Tagen. Es besteht aus 1.648.275.307 Events welche von 12.425 Benutzern, 17.684 Computern und 62.974 Prozesse gesammelt wurden. Es beinhaltet fünf verschiedene Datenelemente mit jeweils unterschiedlichen Features:

- **Authentifizierung**
  Zeit, Quellbenutzer@Domäne, Zielbenutzer@Domäne, Quellcomputer, Zielcomputer, Authentifizierungstyp, Anmeldetyp, Authentifizierungsausrichtung, Erfolg/Misserfolg
- **Prozesse**
  Zeit, Benutzer@Domäne, Computer, Prozessname, Start/Ende
- **Netzwerkverkehr**
  Zeit, Dauer, Quellcomputer, Quellport, Zielcomputer, Zielport, Protokoll, Paketanzahl, Byteanzahl
- **DNS**
  Zeit, Quellcomputer, Computer aufgelöst
- **Red Team**
  Zeit, Benutzer@Domäne, Quellcomputer, Zielcomputer

Brown u. a. (2018) verwendeten dieses Set für ihre Analyse wie in Abschnitt 4.3.16 dokumentiert. LANL bietet zwei weitere Datensets zu Analysen an. Zum einen handelt es sich um Host- und Netzwerkdaten, welche über 90 Tage aufgezeichnet wurden, zum anderen um Benutzerauthentifizierungsdaten, welche über 9 Monate hinweg gesammelt wurden.

5.4. ISCX

Dieses aus dem Jahre 2010 stammende Datenset wurde mit Hilfe realer Netzwerkeinstellungen generiert, dabei wurden Pakete in einem Zeitraum von sieben Tagen in Echtzeit erfasst. Es beinhaltet Protokolle wie HTTP, Simple Mail Transfer Protocol (SMTP), Post Office Protocol v. 3 (POP3), Internet Message Access Protocol (IMAP), SSH und das File Transfer Protocol (FTP). Ein Nachteil dieses Sets ist die Abwesenheit von Hypertext Transfer Protocol Secure (HTTPS) Verkehr, denn

dieser macht einen Großteil des heutigen Datenverkehrs aus. Dadurch bietet dieses Datenset kein repräsentatives reales Szenario für Analysen. Um das Datenset für IDS nützlich zu machen, wurden verschiedene Attacken auf das Netzwerk ausgeführt und aufgezeichnet. Dazu wurde das Netzwerk von innen durch beispielsweise eine Reverse Shell infiltriert. Außerdem wurden DoS und DDoS sowie Brute-Force SSH Angriffe ausgeführt. Ein Vorteil dieses Datensets ist, dass es bereits gelabelt ist. Es besteht somit aus zwei Klassen *normal* und *angriff*. Insgesamt beinhaltet es 2.450.324 Samples, von welchen es sich bei 68.792 um Angriffe handelt. Es besteht aus 19 Features wie beispielsweise der Gesamtzahl der Quell-/Zielpakete, dem Quell-/Zielport, dem Namen des Protokolls sowie der Start-und Endzeit. Wie bereits in Abschnitt 4.3.17 beschrieben, verwendeten Aldwairi, Perera und Novotny (2018) 16 dieser Features für deren Analyse von Netzwerkverkehr.

5.5. CTU-13

Die Intention in der Generierung dieses Datensets lag darin, umfangreiche Daten für die Analyse von Botnetzen zu erzeugen (Garcia u. a. 2014). Dazu wurde Botnetz-, normaler und Hintergrundnetzwerkverkehr aufgezeichnet. Das dabei entstandene CTU-13 Datenset besteht aus 13 Aufzeichnungen. In jeder dieser *Szenarien* wurde eine bestimmte Malware, die mehrere Protokolle verwendet und unterschiedliche Aktionen ausführt verwendet. Dazu wurde Schadsoftware wie Neris, RBot, Virut und Murlo eingesetzt. Insgesamt erfassten die Forscher der Tschechischen Universität *CTU*, Daten in einer Größe von 697 GB was deutlich mehr als der des zuvor beschriebenen ISCX Datensets mit 85 GB entspricht. Auch dieses Datenset ist gelabelt. Es besteht aus den vier Klassen *Botnet, normal,* C&C und *Background*. Eine genaue Anzahl an Features ist leider nicht bekannt. Wie bereits in Abschnitt 4.3.18 dokumentiert, verwendeten Mathur, Raheja und Ahlawat (2018) dieses Datenset als Grundlage für ihre Analyse.

Des Weiteren verwendeten sie das ISOT Datenset, da dies aber wiederum aus zwei eigenständigen Datensets besteht, die beide aus dem Jahre 2004/05 stammen und wovon eines nicht mehr verfügbar ist, wird in Ermangelung an Relevanz auf die Beschreibung dieses verzichtet.

5.6. Microsoft Malware Classification Challenge (BIG 2015)

Für diese Kaggle Challenge stellte Microsoft ein Datenset von über neun verschiedenen Schadsoftware-Klassen zusammen und belohnte das Team mit dem besten Klassifikationsalgorithmus mit 16.000$ (Kaggle 2015). Dieses Datenset verwendeten wie bereits in Abschnitt 4.3.19 erläutert, Sabar, Yi und Song (2018) für ihre Analyse von Schadsoftware in Microsoft Systemen. Das Set aus dem Jahre 2015 beinhaltet Dateien zu folgenden Klassen:

- Ramnit
- Lollipop
- Kelihos_ver3
- Vundo
- Simda
- Tracur
- Kelihos_ver1
- Obfuscator.ACY
- Gatak

Jede Datei verfügt über eine ID, welche aus einem 20 zeichenlangen Hashwert sowie der korrespondierenden Klasse besteht. Zu jedem Sample stehen die Binärdateien (`.bytes`) sowie die dazugehörigen disassemblierten Dateien (`.asm`) zur Verfügung. Aus sicherheitsgründen wird der PE-Header nicht preisgegeben. Allerdings wir zusätzlich ein *Metadata Manifest* herausgegeben. Dieses Protokoll beinhaltet Metadaten der Binärdateien wie beispielsweise Funktionsaufrufe und Zeichenketten. Insgesamt besteht das Datenset aus 21651 Daten von welchen es sich bei 10868 um Trainingsdaten handelt. Das komplette Set hat einen Umfang von knapp einem halben Terabyte und steht weiterhin öffentlich zur Verfügung.

5.7. CICIDS2017

Sharafaldin, Habibi Lashkari und Ghorbani (2018) haben es sich zur Aufgabe gemacht die elf Charakteristiken, von Gharib u. a., eines umfassenden Datensets zu realisieren. Dafür entwickelten sie im Jahr 2017 am *Canadian Institute for Cyber*

Security das CICIDS2017 Datenset. Hierfür zeichneten sie Netzwerkverkehr mit E-Mail Protokollen sowie weiteren Protokollen wie HTTP, HTTPS, FTP und SSH auf. Um ein breites Spektrum aktueller Angriffe widerspiegeln zu können, tätigten sie Angriffe wie DoS, DDoS, Heartbleed, Web Attacken (XSS, SQL Injection), Port Scans und Botnetze. Die dazugehörigen Features wurden mit Hilfe der Software *CICFlowMeter* (Habibi Lashkari u. a. 2017) extrahiert. Insgesamt generierten sie dadurch 80 Features, wie beispielsweise der Anzahl an weitergeleiteten Paketen, die Dauer des Verkehrs, Paketgröße sowie die Anzahl an Bytes pro Sekunde. Die Daten sind entsprechend der Angriffe gelabelt. Das Datenset hat einen Umfang von 286.467 Samples, wovon es sich bei 127.537 um normalen und bei 158.930 um anomalen Netzwerkverkehr handelt. Dadurch konnten die elf Charakteristiken von Gharib u. a. umgesetzt werden. Jedoch plädieren auch die Autoren selbst auf kontinuierlich neue Generierung an Sets beziehungsweise auf deren Erweiterung, um ein auf dem neuesten Stand basierendes Datenset gewährleisten zu können (Sharafaldin, Habibi Lashkari und Ghorbani 2018).

5.8. CIC

Jazi u. a. (2017) entwickelten 2017 ein Datenset speziell für DoS-Angriffe auf Anwendungsebene. Ihre Motivation gründet sich in dem immer häufigeren Auftreten dieser Attacken (Netscout 2019). Ihre Forschung lehnt sich an die von Shiravi u. a. (2012) an. Sie generierten vier Arten von Angriffen mit verschiedenen Tools, um acht unterschiedliche DoS-Angriffe auf Applikationsschichten zu erstellen. Die Angriffe unterteilen sich in *Low-Volume HTTP* und in *High-Volume HTTP* Attacken. Erstere zeichnen sich dadurch aus, dass Datenverkehr in regelmäßig kurzen Intervallen auftritt, durch das Ausnutzen von Zeitparametern durch langsames Senden/Empfangen oder durch eine einzige Verbindung die aufrecht erhalten wird um die Ressourcen des Opfers zu verbrauchen. Letzteres ist auch unter *Flooding* bekannt und verbraucht Ressourcen des Opfers durch eine immense Anzahl an HTTP-GET oder DNS Anfragen. Jazi u. a. zeichneten Netzwerkverkehr inklusive dieser Attacken über 24 Stunden hinweg auf und generierten so eine `.pcap` Datei mit einer Größe von 4.5 GB. Allerdings ist das Datenset weder gelabelt noch verfügt es über HTTPS Verkehr. Wie bereits in Abschnitt 4.3.24 erläutert wird dieses Datenset von Siracusano, Shiaeles und Ghita (2018) verwendet.

5.9. ASNM-NPBO

Das ASNM-NPBO Datenset wurde 2016 von Homoliak (2016) entwickelt. Dafür wurden mit Hilfe von `tcpdump` verschleierte bösartige sowie legitime TCP Kommunikation aufgezeichnet. Dabei entstanden Aufnahmen folgender Services: Apache Tomcat, DistCC, MSSQL, PostgresSQL und Samba. Das Datenset besitzt drei Arten von Labels. Das erste Label *label_2* klassifiziert die Daten binär nach bösartigem Netzwerkverkehr oder nicht und kann die Werte *true* oder *false* annehmen. Das Label *label_poly* ist ein zusammengesetztes Label aus *label_2* und dem entsprechenden Netzwerk Service, welcher bei der Kommunikation verwendet wurde. Ein noch ausführlicheres labeling bietet das dritte Label *label_poly_o*, welches die Werte der beiden Labels *label_2* und *label_poly* verbindet, sowie eine zusätzliche Information bezüglich der Verschleierungstechnik beinhaltet (siehe Abb. 5.9.1).

Technique	Parametrized Instance	ID
Spread out packets in time	• constant delay: 1s	(a)
	• constant delay: 8s	(b)
	• normal distribution of delay with 5s mean 2.5s standard deviation (25% correlation)	(c)
Packets' loss	• 25% of packets	(d)
Unreliable network channel simulation	• 25% of packets damaged	(e)
	• 35% of packets damaged	(f)
	• 35% of packets damaged with 25% correlation	(g)
Packets' duplication	• 5% of packets	(h)
Packets' order modifications	• reordering of 25% packets; reordered packets are sent with 10ms delay and 50% correlation	(i)
	• reordering of 50% packets; reordered packets are sent with 10ms delay and 50% correlation	(j)
Fragmentation	• MTU 1000	(k)
	• MTU 750	(l)
	• MTU 500	(m)
	• MTU 250	(n)
Combinations	• normal distribution delay ($\mu = 10ms$, $\sigma = 20ms$) and 25% correlation; loss: 23% of packets; corrupt: 23% of packets; reorder: 23% of packets	(o)
	• normal distribution delay ($\mu = 7750ms$, $\sigma = 150ms$) and 25% correlation; loss: 0.1% of packets; corrupt: 0.1% of packets; duplication: 0.1% of packets; reorder: 0.1% of packets	(p)
	• normal distribution delay ($\mu = 6800ms$, $\sigma = 150ms$) and 25% correlation; loss: 1% of packets; corrupt: 1% of packets; duplication: 1% of packets; reorder 1% of packets	(q)

Abbildung 5.9.1.: Experimentelle Verschleierungstechniken mit Parametern und IDs (Homoliak u. a. 2019)

Das ASNM-NPBO Datenset bietet ein ausführliches Featureset mit knapp 900 Attributen, allerdings verfügt es lediglich über knapp 7.000 Samples. Wie in Abschnitt 4.3.22 beschrieben wurde dieses Set bereits von Teoh u. a. (2018) für die Analyse von Netzwerkverkehr verwendet.

5.10. CERT

Die CERT Division hat ein Datenset mit synthetischen Insider Bedrohungen erstellt. Dieses enthält fünf unterschiedliche Szenarien einer solchen Bedrohung (Glasser und Lindauer 2013):

1. Benutzer, die zuvor keine Wechseldatenträger verwendet haben oder nach Feierabend arbeiten, melden sich nach Feierabend mit einem Wechseldatenträger an und laden Daten auf wikileaks.org hoch. Verlässt die Organisation kurz danach.
2. Der Benutzer beginnt, auf Jobwebsites zu surfen und eine Stelle bei einem Wettbewerber zu suchen. Bevor das Unternehmen verlassen wird, wird ein USB-Stick (mit deutlich höheren Raten als bei ihrer vorherigen Aktivität) verwendet, um Daten zu stehlen.
3. Der verärgerte Systemadministrator lädt einen Keylogger herunter und überträgt ihn mithilfe eines USB-Sticks auf den Computer seines Vorgesetzten. Am nächsten Tag verwendet er die gesammelten Keylogs, um sich als Vorgesetzter anzumelden und eine alarmierende Massen-E-Mail zu versenden, die in der Organisation Panik auslöst. Er verlässt die Organisation sofort.
4. Ein Benutzer meldet sich auf dem Computer eines anderen Benutzers an, sucht nach interessanten Dateien und sendet eine E-Mail an seine private E-Mail. Dieses Verhalten tritt immer häufiger über einen Zeitraum von 3 Monaten auf.
5. Ein Benutzer meldet sich auf dem Computer eines anderen Benutzers an, sucht nach interessanten Dateien und sendet eine E-Mail an seine private E-Mail. Dieses Verhalten tritt immer häufiger über einen Zeitraum von 3 Monaten auf.

Das Datenset besteht aus mehreren Dateien und Ordnern. Zunächst ist ein LDAP Verzeichnis vorhanden, welches die Namen aller Benutzer, deren ID, E-Mail Adressen sowie Position beinhaltet. Des Weiteren gibt es eine `device.csv` Datei, in welcher Zugriffe auf Dateien festgehalten werden. In der Datei `email.csv` werden alle Daten zu E-Mail Transaktionen zwischen Mitarbeitern dokumentiert. Zusätzlich wird ein Web Protokoll in Form von `http.csv` erstellt, in welchem die aufgerufenen URLs pro Benutzer aufgelistet werden. Außerdem besteht eine `logon.csv` Datei, welche die An- und Abmeldungen pro Benutzer aufzeichnet. In der Datei `insiders.csv` werden alle bösartigen Benutzeraktivitäten pro Benutzer gelistet. Dort wird auf die jeweilige Kompromittierung in anderen Dateien verwiesen. Wie bereits in Abschnitt 4.3.32 beschrieben, analysierten Le und Nur Zincir-Heywood

(2019) im Zuge ihrer Forschung dieses knapp 85 GB große Datenset.
Ein Vorteil eines solch synthetisierten Datensets bietet der Verzicht auf die Anonymisierung von Daten, da es keine tatsächlichen personenbezogenen Daten oder Betriebsgeheimnisse zu schützen gibt. Allerdings kann somit die Wirksamkeit des Modells nicht realitätsgetreu bewiesen werden. Außerdem ist die Vielzahl möglicher Bedrohungen anhand der lediglich fünf verwendeten Szenarien nicht ausreichend abgedeckt.

5.11. Ember

Das Ember Datenset wurde von Anderson und Roth (2018) generiert und besteht aus über einer Million Windows PE-Dateien. Die Features hierfür wurden mit Hilfe von Library to Instrument Executable Formats (LIEF) erstellt, wobei es sich um eine plattformübergreifende Bibliothek zum Parsen, Ändern und Abstrahieren von PE-Dateien handelt (Quarkslab 2019).
Insgesamt gibt es zwei Versionen von Features. Version 1 wurde mit einer älteren LIEF Version erstellt. Für Version 2 wurde ein neuerer Release verwendet, außerdem beinhaltet diese zusätzliche Features und eine verbesserte Importverarbeitug ordinaler Features.
Insgesamt stellen Anderson und Roth drei Sets zur Verfügung:

- Version 1 mit Daten aus 2017
- Version 2 mit Daten aus 2017
- Version 2 mit Daten aus 2018

Bei Letzterem wurden die Samples so ausgewählt, dass die resultierenden Trainings- und Testsätze für Lernalgorithmen schwieriger zu klassifizieren sind.
Die Samples liegen in JavaScript Object Notation (JSON) Format vor. Ein JSON Objekt besteht unter anderem aus einem hash256 der Datei, der Zeitangabe wann die entsprechende Datei erstmals auftrat und einem Label (0/1/-1),wobei 0 für gutartig, 1 für bösartig und -1 für ungelabelt steht.
Das Ember Datenset besteht insgesamt aus acht Gruppen von Features (siehe Abbildung 5.11.1), welche sich aus geparsten und formatunabhängigen Untergruppen zusammen setzen. Zu den zunächst mit Hilfe von LIEF geparsten Features gehören generelle Dateiinformationen wie die Größe der Datei sowie die Anzahl der importierten und exportierten Funktionen. Außerdem gehören dazu die Header Information, welche den Zeitstempel sowie die Zielmaschine enthält. Zusätzlich verfügt diese Gruppe von Features über Informationen bezüglich importierten und exportierten Funktionen, sowie über Informationen bezüglich weiterer Sektionen der Portable

Executable Datei (PE-Datei). Zur zweiten Gruppe, der formatunabhängigen Features gehören das Byte-Histogramm, welches die Anzahl einzelner Bytes innerhalb einer Datei darstellt, das Byte-Entropie-Histogramm, welches die gemeinsame Verteilung der Entropie und der einzelner Bytewerte approximiert, sowie Informationen über Zeichenketten wie beispielsweise deren Anzahl und Durchschnittslänge.


```
"sha256": "000185977be72c8b007ac347b73ceb1ba3e5e4dae4fe98d4f2ea92250f7f580e",
"appeared": "2017-01",
"label": -1,
"general": {
  "file_size": 33334,
  "vsize": 45056,
  "has_debug": 0,
  "exports": 0,
  "imports": 41,
  "has_relocations": 1,
  "has_resources": 0,
  "has_signature": 0,
  "has_tls": 0,
  "symbols": 0
},
"header": {
  "coff": {
    "timestamp": 1365446976,
    "machine": "I386",
    "characteristics": [ "LARGE_ADDRESS_AWARE", ..., "EXECUTABLE_IMAGE" ]
  },
  "optional": {
    "subsystem": "WINDOWS_CUI",
    "dll_characteristics": [ "DYNAMIC_BASE", ..., "TERMINAL_SERVER_AWARE" ],
    "magic": "PE32",
    "major_image_version": 1,
    "minor_image_version": 2,
    "major_linker_version": 11,
    "minor_linker_version": 0,
    "major_operating_system_version": 6,
    "minor_operating_system_version": 0,
    "major_subsystem_version": 6,
    "minor_subsystem_version": 0,
    "sizeof_code": 3584,
    "sizeof_headers": 1024,
    "sizeof_heap_commit": 4096
  }
},
```

```
  "imports": {
    "KERNEL32.dll": [ "GetTickCount" ],
    ...
  },
  "exports": []
  "section": {
    "entry": ".text",
    "sections": [
      {
        "name": ".text",
        "size": 3584,
        "entropy": 6.368472139761825,
        "vsize": 3270,
        "props": [ "CNT_CODE", "MEM_EXECUTE", "MEM_READ"]
      },
      ...
    ]
  },
  "histogram": [ 3818, 155, ..., 377 ],
  "byteentropy": [0, 0, ... 2943 ],
  "strings": {
    "numstrings": 170,
    "avlength": 8.170588235294117,
    "printabledist": [ 15, ... 6 ],
    "printables": 1389,
    "entropy": 6.259255409240723,
    "paths": 0,
    "urls": 0,
    "registry": 0,
    "MZ": 1
  },
}
```


Abbildung 5.11.1.: Rohe Features, die aus einer einzelnen PE-Datei extrahiert wurden (Anderson und Roth 2018)

Wie bereits in Abschnitt 4.3.27 dokumentiert, verwendeten Vinayakumar u. a. (2019) dieses Datenset bei ihrer Analyse von Schadsoftware.

5.12. Evaluation der Datensätze

Wie zu Beginn des Kapitels erläutert sollen die Datensätze miteinander verglichen werden, um eine Aussage über deren jeweilige Qualität zu treffen. Da all die in Tabelle 5.1 aufgelisteten Datensets, Samples zu verschiedenen Use Cases beinhalten, ist es nicht möglich diese direkt miteinander zu vergleichen. Deshalb wurden Attribute gewählt, welche für jegliche Datensets von hoher Wichtigkeit sind. Dazu

wurden folgende Attribute gewählt: Jahr, Größe beziehungsweise Umfang, Inhalt, Labels und Features.
Das Jahr soll die Aktualität eines Datensets widerspiegeln. In einer sich schnell ändernden IT Welt sind Angriffe, beziehungsweise Schwachstellen, die vor einigen Jahren noch sehr häufig auftraten unter Umständen nicht mehr Zeitgemäß.
Da die Genauigkeit von MLAs abhängig von der Größe und Diversität ihres zugrunde liegenden Trainingssets ist, wäre ein umso größeres Datenset von Vorteil, weshalb der Aspekt der Größe als auch der des Inhalts betrachtet wird.
Ein bereits gelabeltes Datenset erspart Data Scientists viel Zeit und somit Arbeit und verstärkt dadurch die positive Bewertung eines Datensets.
Eine genaue Beschreibung der Features und eine große Anzahl dieser, ist dahingehend von Vorteil, dass dadurch Feature Engineering betrieben werden kann, was zu einer deutlichen Verbesserung der Ergebnisse führen kann.
Nachfolgende Tabelle 5.1 gibt einen Überblick über die Ergebnisse der Datenset Analyse. Dabei sind die Datensets in derselben Reihenfolge angeordnet, in welcher sie zuvor beschrieben wurden.

Name	Jahr	Inhalt	Features	Umfang	Label
HTTP CSIC	2010	HTTP Web Anfragen	13	∼60 MB	normal/ anomal
NSL-KDD	2009	Netzwerkverkehr (>12 Protokolle)	41	∼290 GB	DoS, Probe, U2R, R2L
LANL	2015	Authentifizierung, Prozesse, DNS Suchanfragen, Netzwerkdaten, Red Team Events	30	∼270 GB	Red Team/normal
ISCX	2010	Netzwerkverkehr (6 Protokolle)	19	∼85 GB	normal/ angriff
CTU-13	2014	Netzwerkverkehr (10 Protokolle)	-	∼697 GB	Hintergrund, Botnet, C&C, normal
MMCC	2015	.byte Dateien	-	∼500 GB	Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak
CICIDS	2017	Netzwerkverkehr (6 Protokolle)	80	∼50 GB	normal/ anomal
CIC	2017	Netzwerkverkehr	-	∼46 GB	-
ASNM-NPBO	2016	Netzwerkverkehr	∼900	∼40 MB	normal/anomal (inkl. Verschleierungs-technik)
CERT	2013	Synthetische Insiderbedrohungen	23	∼85 GB	-
EMBER	2018	PE Dateien	2381	∼20 GB	gutartig, bösartig, ungelabelt

Tabelle 5.1.: Evaluierung ausgewählter Datensätze

Die Tabelle 5.1 zeigt deutlich den Nachteil der Datensets CTU-13, Microsoft Malware Classification Challenge (MMCC) und CIC, da diese nicht über Features verfügen. Zusätzlich fehlt es Letzterem, sowie dem CERT Datenset, an Labels. Von den sechs Sets welche Daten zu Netzwerkverkehr beinhalten, bietet das NSL-KDD Datenset die meisten Samples, Features und Protokolle. Allerdings ist dies bereits aus dem Jahr 2009 und somit veraltet. Eine Alternative hierzu bietet das aktuellere ASNM-NPBO Datenset aus dem Jahr 2016. Dieses verfügt über mehr als 900 Features. Zusätzlich ist es gelabelt. Allerdings ist es deutlich kleiner als das NSL-KDD

Datenset.
In Sachen Malware mangelt es der Forschungsgemeinde deutlich an Datensets. Wie diese Untersuchung zeigt befinden sich lediglich zwei Sets bezüglich Schadsoftware unter der Analyse. Das MMCC aus dem Jahr 2015 stellt zwar ein halbes Terabyte an Daten zur Verfügung, jedoch fehlt es diesem an Features. Eine bessere Alternative bietet das Ember Datenset. Dieses verfügt über mehr als eine Million PE-Dateien, ist gelabelt und besitzt über 2381 Features. Da es sich bei diesem zusätzlich um das aktuellste Datenset handelt, weist es die, im Vergleich zu allen anderen, besten Attribute auf.

6. Prototypische Implementierung

Basierend auf den Ergebnissen der Untersuchung der Analyseverfahren, wie in Kapitel 4.4 dokumentiert, wird ein Prototyp eines Schadsoftwareklassifikators erstellt, welcher im Folgenden erläutert wird.

Durch die Untersuchung aus dem vorangegangenen Kapitel wurde deutlich, dass sich das Ember Datenset am besten für diesen Use Case eignet, da dieses bereits gelabelte Daten besitzt und zudem das aktuellste Set darstellt. Somit bildet dies die Grundlage des Prototyps. Um das Datenset eingehender zu untersuchen wurde dieses zunächst einer Validierung unterzogen.

Das Ember Datenset besteht, wie in Kapitel 5.11 erläutert, aus 1.1 Millionen PE-Dateien aus dem Jahr 2017. Diese bilden das Set *EMBER2017*. Das Set aus dem Jahre 2018 *EMBER2018*, besteht aus einer Million PE-Dateien. Die beiden Sets unterscheiden sich dahin gehend, dass letzteres Samples enthält, welche für Lernalgorithmen schwieriger zu klassifizieren sind. Aus diesem Grund und weil Anderson und Roth (2018) bereits Untersuchungen zu EMBER2017 veröffentlicht haben, wurde für den Prototyp das EMBER2018 Set verwendet. Dieses verfügt über 2381 Features in JSON Format. Zusätzlich zu den Datensets stellen die Entwickler ein Ember Modul zur Verfügung, wodurch es möglich ist, die Features zu vektoriesieren. Dies ist besonders für neuronale Netze wichtig, da sie nicht mit Text-, Grafik- oder anderen Nicht-Vektor/Matrixdarstellungen arbeiten können. Durch die immense Anzahl an Daten nimmt die Vektorisierung einige Zeit in Anspruch. Die vektorisierten Daten liegen anschließend in Binärformat als `.dat` Dateien vor, von wo aus sie zu Pandas Dataframes, Comma-Separated Values (CSV) oder anderen Dateiformaten konvertiert werden können.
Das Ember Modul bietet ein implementiertes LightGBM Modell welches PE-Dateien klassifiziert. Dies wurde zunächst anhand eines bösartigen, als auch anhand eines gutartigen Programms validiert. Hierfür wurde ein Python Testskript erstellt, welches zunächst das LightGBM Modell aus der Datei `ember_model_2018.txt` lädt und dieses anschließend auf ein ausgewähltes Programm anwendet. Dabei wird ein Wahrscheinlichkeitswert im Intervall von [0:1] berechnet, wobei 0 für gutartig und 1 für bösartig steht. Das Testskript (siehe Anlage 3) liest die Programme `putty.exe` sowie `mimikatz.exe` ein, berechnet mit Hilfe des Modells die Wahrscheinlichkeit

und gibt folgendes Ergebnis zurück:

```
Is Putty Malware?
0.0  !Benign!
Is Mimikatz Malware?
0.99 !Malicious!
```

Listing 6.1: Ergebnis Python Testskript

Da es sich bei `putty.exe` um einen frei verfügbaren SSH Client für Windows und bei `mimikatz.exe` um ein bösartiges Programm, welches unter anderem Passwörter von Windows Benutzern auslesen kann, handelt, hat das LightGBM Modell beide Programme exakt richtig klassifiziert.
Anderson und Roth (2018) stellen zusätzlich ein Jupyter Notebook zur Verfügung, in welchem sie das LightGBM Modell mit dem EMBER2017 Set implementierten. Dieses erzielte eine ROC AUC von 0.99. Zur weiteren Validierung des Datensets in Bezug auf dessen Verwendung für den Prototyp, wurde das EMBER2018 Set ebenfalls anhand dieses Notebooks analysiert. Dabei erzielte das Modell eine etwas schlechtere ROC AUC von 0.996, als die von Anderson und Roth (2018) veröffentlichte für das ältere Datenset. Anhand dieser Tests konnte eine gute Qualität der Daten nachgewiesen werden.

So vorteilhaft die Fülle an Daten für das Training von Lernalgorithmen erscheinen mag, so nachteilig erweist sich diese in der Praxis. Bereits das Einlesen der Daten erfordert so viel Speicherplatz, dass eine Analyse weder auf lokaler noch auf virtueller Hardware der THU möglich ist. Aus diesem Grund wurde nach einer Alternative gesucht, welche sich in Google Colaboratory (kurz Colab) fand. Dabei handelt es sich um eine Cloud Umgebung, auf welcher Python Notebooks ausgeführt werden können. Google stellt dabei sowohl GPU als auch Tensor Processing Unit (TPU) Hardware zur Verfügung.
Der Prototyp wurde in Google Colab mit einem GPU Backend implementiert. Als Deep Learning Framework wurde die Keras API ausgewählt, da diese einfach anzuwendende Best Practices anbietet und so eine effiziente Implementierung gewährleistet. Des Weiteren hat Keras eine breite Akzeptanz in der Industrie und in der Forschungsgemeinschaft und verfügt dadurch über eine sehr gute Dokumentation, sowie Hilfestellungen der Community bei Problemen (Keras 2020). Keras ist eine Bibliothek die High-Level Operationen anbietet. Für Low-Level Operationen können Backends wie Tensorflow eingebunden werden. Dieses wurde auch für den folgenden Prototyp verwendet. Zu analysierende Daten können direkt aus Google Drive in Colab verwendet werden. Da Google Drive allerdings nur 15 GB kostenlos zur Verfügung stellt, das Ember Datenset jedoch aus knapp 20GB besteht, musste zunächst der Speicherplatz erweitert werden. Das Ember Set bietet den Vorteil, dass die Features bereits extrahiert wurden und lediglich noch vektorisiert werden

müssen. Im Anschluss an die Vektorisierung liegen die Daten bereits als Test- und Trainingsset bereit, wodurch ein Großteil der sonst üblichen Datenvorbereitung entfällt.

Wie aus der vorangegangenen Untersuchung der Analyseverfahren hervorgeht, scheinen CNNs sowie LSTM Netze, welche eine besondere Form von RNNs bilden, die besten Ergebnisse bei der Erkennung von Malware zu bieten. Zusätzlich weisen Joshua Saxe (2018) darauf hin, dass sich RNNs am besten zur Klassifizierung von sequenziellen Daten, wie Binärdaten aus PE-Dateien, eignen. Basierend auf diesen Erkenntnissen wurden drei LSTM Modelle mit unterschiedlichen Parametern implementiert und getestet um daraus, das für die Klassifizierung von PE-Dateien am besten geeignetste Modell zu evaluieren.

Wie bereits in Kapitel 4.2 beschrieben, verwenden neuronale Netze Backpropagation um Fehlergradienten zunächst von der Ausgabe- zur Eingabeschicht weiterzugeben. Sobald der Algorithmus den Gradienten auf die Kostenfunktion in Bezug auf jeden Parameter im Netzwerk berechnet hat, verwendet er diese um Parameter mit einem Gradientenabstiegsschritt zu aktualisieren. In tiefen neuronalen Netzen kann dies zu sehr großen Gradienten führen, welche wiederum zu einer zu großen Aktualisierung und so zu einem instabilen Netzwerk führen. Gleichzeitig kann es aber auch vorkommen, dass die Gradienten immer kleiner werden und dadurch kaum mehr eine Aktualisierung stattfindet. Dieses Problem wird *vanishing gradient descent* genannt (Géron 2019).
LSTM Netzwerke können dieses Problem umgehen (Hochreiter und Schmidhuber 1997). Diese Netze bestehen aus drei Toren: einem Eingangstor (**i**), einem Vergessenstor (**f**) und einem Ausgangstor (**o**). Das Eingangstor ist dafür zuständig wichtige, eingegangene Informationen zu erkennen und diese über einen langen Zeitraum hinweg zu speichern. Wird die Information nicht mehr benötigt wird sie gelöscht, was die Aufgabe des Vergessenstor darstellt. Das Ausgangstor entscheidet wie die Werte der versteckten Einheiten zu aktualisieren sind. Die genauere Struktur einer LSTM Zelle ist in Abbildung 6.0.1 dargestellt.

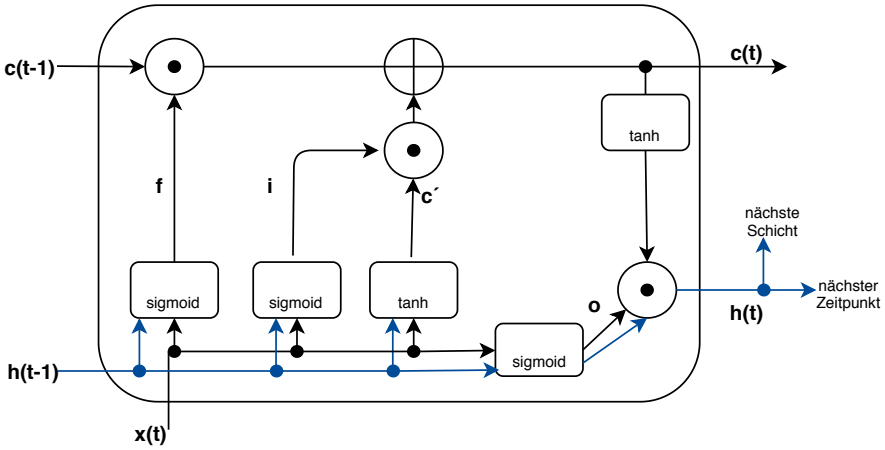


Abbildung 6.0.1.: Struktur einer LSTM Zelle (eigene Darstellung in Anlehnung an Raschka und Mirjalili (2019))

Zunächst kommt der Zellzustand **c(t-1)** aus der vorherigen Zelle an, im Anschluss wird der neue Zustand **c(t)** an die Nächste Zelle weitergegeben. $\odot$ bezieht sich auf die elementweise Multiplikation, $\oplus$ auf die elementweise Addition, **x(t)** auf die Eingangsdaten und **h(t-1)** auf versteckten Einheiten.
Der erste Schritt besteht darin zu entscheiden, wie viel vorherige Information in der Zelle aufbewahrt wird. Dafür ist das Vergessenstor zuständig. Die Sigmoid Funktion liefert dazu einen Wert zwischen 0 und 1, wobei 1 die Mitnahme aller Informationen und 0 den Verzicht aller Informationen darstellt. Anschließend wird der Wert innerhalb der Zelle aktualisiert. Hierbei entscheidet wiederum eine Sigmoid Schicht, welche Werte aktualisiert werden, jedoch erstellt die tanh Schicht den Vektor für die neuen Kandidatenwerte. Der neue Wert **c'** wird der tanh Funktion übergeben und deren Ausgabe mit der Ausgabe der Sigmoid Funktion multipliziert. Der neue Wert der versteckten Einheiten wird an die nächste Schicht weitergegeben.
Wie bereits erwähnt muss diese komplexe Struktur nicht selbst implementiert werden, da es bereits vorgefertigte Lösungen wie Tensorflow gibt.

Da der Großteil der Vorbereitung der Daten des Ember Datensets bereits von den Entwicklern übernommen wurde, müssen lediglich Algorithmus spezifische Vorbereitungen getroffen werden. Dazu zählt zunächst die Normalisierung der Daten. Dies ist ein Prozess bei dem Daten mit unterschiedlichen Größenordnungen einander angeglichen werden um diese besser miteinander vergleichen zu können. Die Labels lagen bereits als Integer Werte vor, allerdings verarbeitet Keras lediglich positive Labels, da das Ember Datenset nicht klassifizierte Samples mit -1 gelabelt hat, wurden diese durch das Label 2 ersetzt.
Um bei dem Training ein Under- bzw Overfitting zu vermeiden wird die Lernrate

schrittweise verringert. Um die Effizienz des Trainings zu überwachen, wurden Tensorflow Protokolle erstellt, wodurch sich die Ergebnisse einfach plotten lassen. Zusätzlich wurde eine Tensorflow Funktion verwendet um Protokollierungsrauschen zu reduzieren und lediglich jede hundertste Epoche einen vollständigen Satz von Metriken auszugeben. Eine Rückruf Funktion wurde eingebaut um unnötige Trainigszeiten zu vermeiden. Diese basiert auf dem Wert `vall_loss` also dem loss der Validierungsdaten und wurde so gewählt, dass das Training beendet wird wenn sich dieser Wert innerhalb von 200 Epochen nicht verbessert.
Für jedes der drei Modelle wurden dieselben Trainingskonfigurationen sowie dieselben Einstellungen für `Model.compile` und `Model.fit` verwendet. Erstere Methode konfiguriert das Modell für das Training. Hierbei wurde *Adam* als Optimierer gewählt, da dieser recheneffizient ist, wenig Speicher belegt und sich besonders für große Datenmengen eignet (Kingma und Ba 2015). Als *loss function* wurde passend zu den Ember Daten `sparse_categorical_crossentropy` verwendet, da das Modell drei Labels besitzt, welche als Integer kodiert sind. Zusätzlich wurde die Genauigkeit als Metrik gewählt, welche vom Modell während des Trainings ausgewertet werden soll. Die Methode `Model.fit` trainiert das Modell für eine feste Anzahl an Epochen. Da Google Colab Ressourcen dem laufenden Betrieb anpasst, kann die Verfügbarkeit der Ressourcen schwanken. Gerade bei Langzeitberechnungen besteht die Gefahr, dass der Zugang zu GPUs zeitweise beschränkt wird, wodurch die Laufzeitumgebung getrennt wird und die Berechnungen neu gestartet werden müssen. Um dies zu verhindern wurden die Epochen für das Training auf 500 limitiert. Als Eingabedaten werden die normalisierten Trainingsdaten als dreidimensionales Array übergeben, da ein LSTM diese Form benötigt um Berechnungen anzustellen. Zusätzlich kann in dieser Methode die Anzahl der Daten angegeben werden, welche im Training als Validierungsdaten dienen. Diese wurden auf 20% festgesetzt und entsprechen somit 160.000 Samples.
Die mit diesen Konfigurationen trainierten Modelle sind `modelRnn1`, `modelRnn2` und `modelRnn3`.
Das erste Modell besteht aus einer Input Layer mit 32 Neuronen, 6 Hidden Layers und einer Output Layer mit 3 Neuronen, da die Daten nach drei Klassen klassifiziert werden sollen. 32 Neuronen wurden gewählt, da sich das Training bei dieser Anzahl zeitlich in einem akzeptablen Rahmen befand. Untersuchungen mit einer höheren Anzahl führten zu einem viel zu zeitintensiven Trainningsprozess. Hierbei kam es zur Trennung mit der Laufzeitumgebung. In den Hidden Layers befinden sich Dropout Layers, welche einen gewissen Anteil an Neuronen auf null setzen um eine Überanpassung zu vermeiden. Zusätzlich wurde eine Batch Normalisierungs Layer eingebaut um die Aktivierungen der vorangegangenen Schicht zu normalisieren. Dies soll zu einer verbesserten Leistung und Geschwindigkeit von neuronalen Netzen führen (Ioffe und Szegedy 2015). Für das erste Modell wurde *softmax* als Aktivierungsfunktion gewählt, da diese aussagekräftige Klassenwahrscheinlichkeiten

bei Multi-Klassen Klassifikation im Bereich [0:1] berechnet.
Für das zweite Modell wurde die Aktivierungsfunktion zu *tanh* geändert. Diese hyperbolische Tangensfunktion ist eine Neuskalierung der Sigmoid Funktion und produziert eine Ausgabe zwischen [-1:1]. Zusätzlich wurde eine Methode verwendet um die Regulierung der Gewichte sicherzustellen.
Das dritte Modell übernimmt die Parameter des ersten, wurde jedoch um die Kernel Regularisierung, welche die Regulierung der Gewichte sicherstellt, erweitert.
Insgesamt beanspruchte der Trainingsprozess eine Dauer von mehr als vier Stunden.

Zur Evaluation der Modelle wurden die Metriken Genauigkeit und loss verwendet. Um diese besser auswerten zu können, wurden entsprechende Plots erstellt.

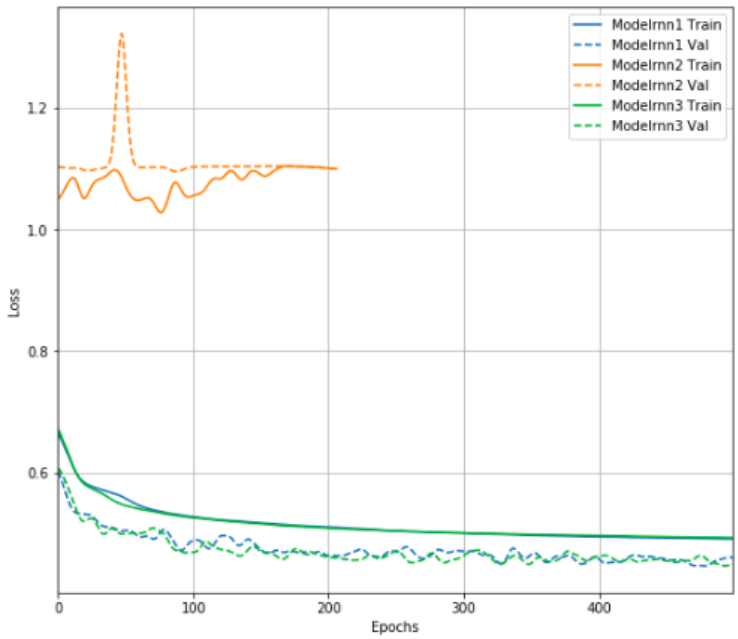


Abbildung 6.0.2.: Loss nach Epochen für Trainings- und Testdaten (eigene Darstellung)

Wie aus Abbildung 6.0.2 hervorgeht, hat die Rückruf Funktion bei `modellRnn2` bereits nach knapp 220 Epochen gegriffen, da sich dieses nicht mehr weiterentwickelt hat. Dies führt zu der Annahme, dass die *tanh* Aktivierungsfunktion für diese Art von Daten und diese Art der Analyse nicht geeignet ist. `modelRnn2` und `modelRnn3` konnten ihren loss hingegen kontinuierlich verringern und wurden daher nicht vorzeitig gestoppt.

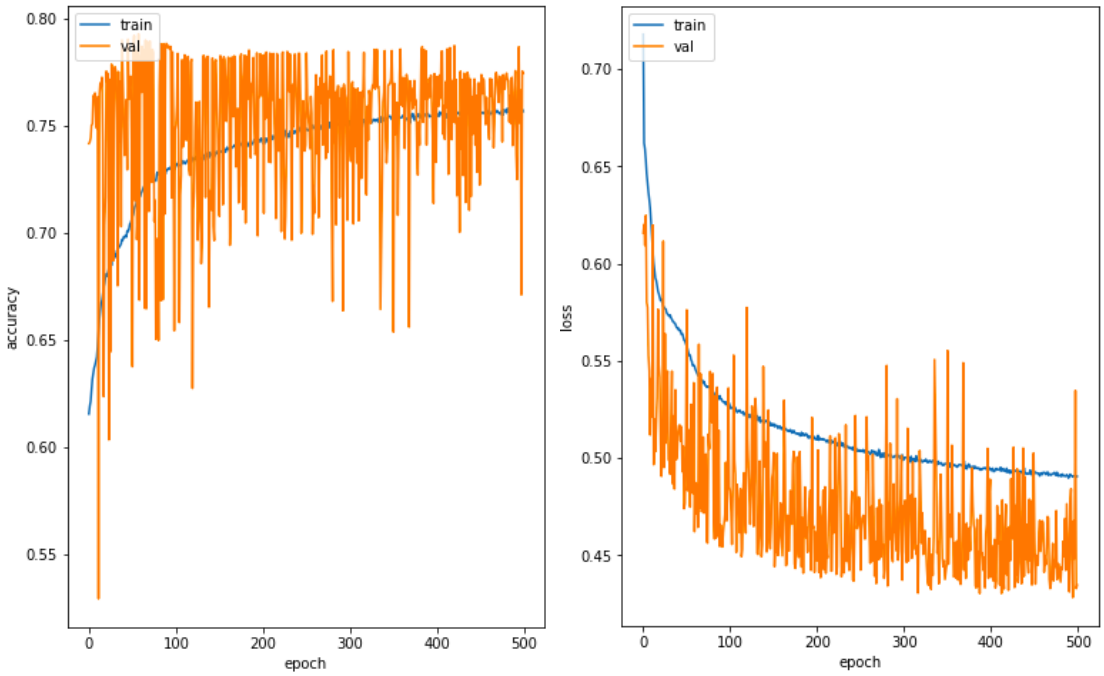


Abbildung 6.0.3.: Genauigkeit und loss `modelRnn1`
(eigene Darstellung)

Eine genauere Betrachtung des ersten Modells in Abbildung 6.0.3 zeigt die kontinuierliche Verbesserung der Genauigkeit über den kompletten Trainingsverlauf hinweg, sowie die fortwährende Abnahme des loss. Dadurch kann ein signifikanter Trainingserfolg nachgewiesen werden.

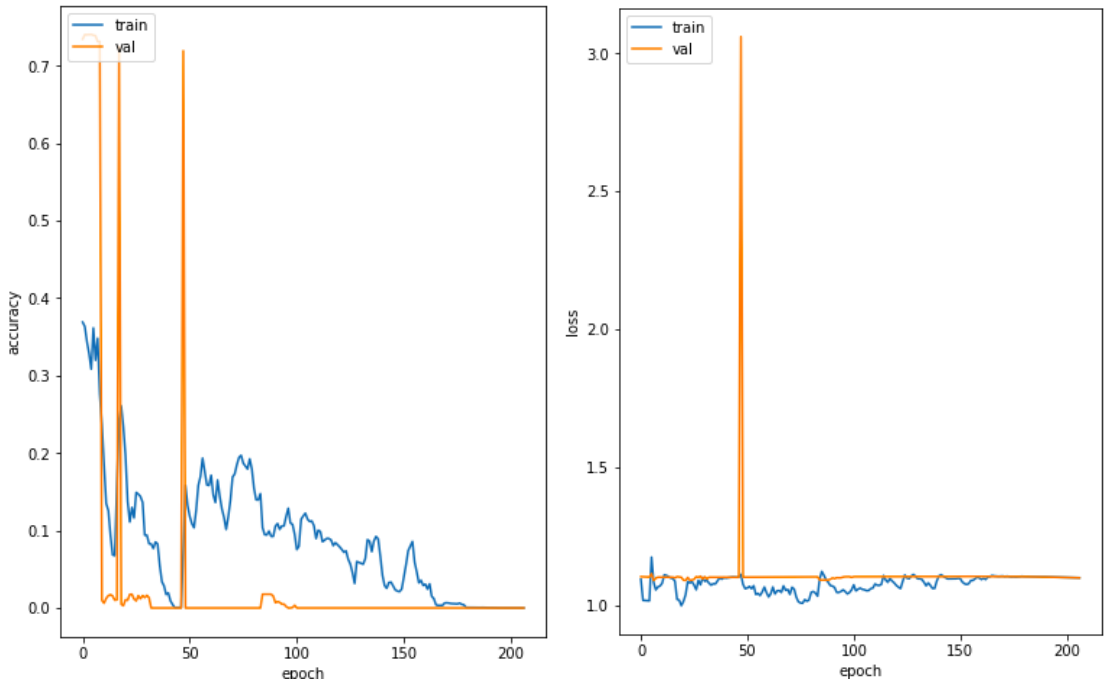


Abbildung 6.0.4.: Genauigkeit und loss `modelRnn2`
(eigene Darstellung)

Wie in der Gesamtübersicht bereits ersichtlich ist, kann das zweite Modell keinen Trainingserfolg nachweisen. Abbildung 6.0.4 zeigt deutlich die Zunahme des loss anstatt der Abnahme, sowie einer Abnahme der Genauigkeit anstatt einer Zunahme. Durch die schlechte Entwicklung wurde das Training vorzeitig durch die Rückruf Funktion abgebrochen.

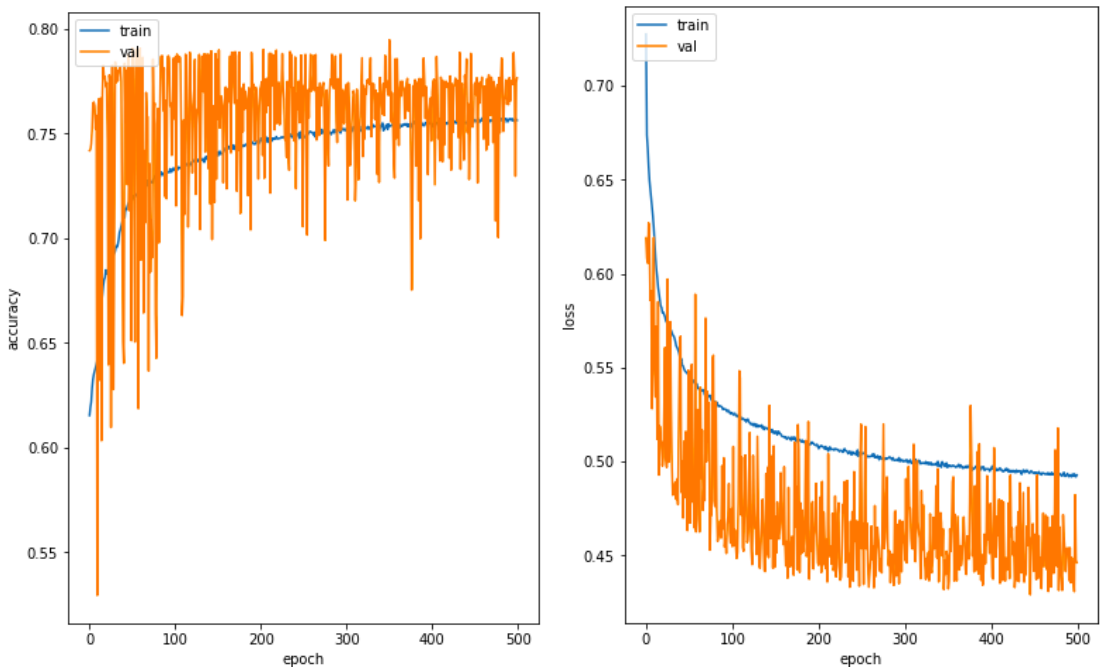


Abbildung 6.0.5.: Genauigkeit und loss `modelRnn3`
(eigene Darstellung)

Das dritte Modell zeigt wiederum einen ähnlichen Trainingserfolg wie das erste Modell. Allerdings fällt der loss bei der Validierung bereits zu Beginn niedriger aus als beim ersten Modell. Dahin gehend fällt die Genauigkeit insgesamt etwas schlechter aus. Nachfolgende Tabelle 6.1 zeigt eine Übersicht der Genauigkeit der getesteten Modelle.

Modell	**Genauigkeit**
modelRnn1	0.787
modelRnn2	0.0
modelRnn3	0.783

Tabelle 6.1.: Ergebnisse der getesteten Modelle

Bei einem Vergleich der drei Modelle wird deutlich, dass das erste Modell für die Analyse am besten geeignet ist. Die Kombination aus der *softmax* Aktivierungsfunktion, den Dropout Layers sowie der Batch Normalisierungs Layer erzielt mit einer Genauigkeit von 0.787 das beste Ergebnis. Dieses Modell wurde mit,

ihm unbekannten, Daten getestet. Dabei erbrachte es die, wie in Abbildung 6.0.6 ersichtlichen, Klassifizierungen.

sample	zero	one	label
1	0.01	0.99	1
2	0.35	0.65	0
3	0.05	0.95	1
4	0.71	0.29	0
5	0.66	0.34	0
6	0.03	0.97	1
7	0.58	0.42	0
8	0.73	0.27	0
9	0.68	0.32	1
10	0.21	0.79	0

Abbildung 6.0.6.: Klassifizierung von unbekannten Testdaten (eigene Darstellung)

Diese Abbildung zeigt die vorhergesagten Klassenzugehörigkeiten von Testsamples des ersten Modells. Da die *softmax* Aktivierungsfunktion verwendet wurde, wird die jeweilige Klassenzugehörigkeit mit Wahrscheinlichkeitswerten dargestellt, welche in Summe eins ergeben. Die Spalte Label spiegelt die tatsächliche Klasse des Samples wider. Sample 1 ist für das Modell beispielsweise eindeutig identifizierbar, da es für Klasse null eine Wahrscheinlichkeit von 0.1, für Klasse 1 jedoch eine Wahrscheinlichkeit von 0.99 berechnet.

7. Zusammenfassung und Ausblick

Wie gezeigt werden konnte, ist das aufspüren von IoCs essenziell, für eine erfolgreiche Malware Analyse, da diese einer späteren Analyse als Features dienen. Es wurde herausgefunden, dass IoCs sowohl durch statische als auch durch dynamische Analysen herausgefiltert werden können. Bei der statischen Untersuchung finden sich Indikatoren, wie verdächtige Imports oder Exports, in PE-Dateien. Dynamische Analysen können hingegen bedenkliche API Aufrufe zum Vorschein bringen. Da dieser Bereich jedoch sehr umfassend ist, konnte in dieser Arbeit lediglich ein kleiner Ausschnitt dessen was als IoCs in Bezug auf Malware gilt, erörtert werden.
Um den momentanen Stand der Forschung sowie die Adaption von Machine Learning Ansätze im Bezug auf Cyber Security zu erforschen, wurden anhand einer umfassenden Literaturrecherche Ansätze zu diesem Thema analysiert. Die Untersuchung zeigt eine breite Adaption von Machine Learning im Bereich von Cyber Security. Dabei konnte ein Trend in Richtung Malware Analyse festgestellt werden. Jedoch wurden auch Bereiche wie Netzwerkverkehr und Botnetze auffallend oft untersucht. Zudem gibt es breit gefächerte Untersuchungen von einzelnen Themen wie Phishing Websites, Insider Bedrohungen in Unternehmen und bösartigen Power-Shell Befehlen.
Im Gegensatz zu den ausführlich dokumentierten Untersuchungen der Wissenschaftsgemeinde, konnten im Bereich der Industrie nicht sehr viele Verfahren gefunden werden. Allerdings konnte anhand der Literaturrecherche festgestellt werden, dass das Thema Machine Learning im Kontext von Cyber Security viel diskutiert und somit auch in der Industrie aktuell ist. Dennoch konnte, in Anbetracht des geringen Zeitfensters in welchem die Arbeit angefertigt wurde, kein zufriedenstellender Überblick über die Verwendung von Machine Learning in der Industrie erbracht werden.
Da die Basis jeglicher Machine Learning Analysen Daten bilden, galt es auch diesen Aspekt zu beleuchten. Die Untersuchung vorhandener Datensets ergab, dass es zwar Datensets für viele Bereiche wie Malware Klassifizierung, Netzwerkanalysen oder Insider Bedrohungen gibt, jedoch wurden bei allen kleinere bis gröSSere Mängel festgestellt. In Anbetracht des Trends zur Malware Analyse kann zusätzlich festgehalten werden, dass ein deutlicher Mangel an Datensets für diesen Use Case vorhanden ist. Dennoch erwies es sich als schwierig die Datensets objektiv miteinander zu vergleichen, da es keine standardisierte Evaluation für Datensets

gibt. In Ermangelung dessen, musste ein eigens dafür kreierter Evaluationsprozess angewendet werden.
Durch die erfolgreiche Implementierung des Prototyps, konnte zunächst nachgewiesen werden, dass die Verwendung des Ember Datensets in Kombination mit neuronalen Netzen grundsätzlich möglich ist. Zusätzlich konnte dieser seine Klassifikationsfähigkeit, mit einer Genauigkeit mit 0.787, zeigen. Allerdings konnten, durch anfängliche Probleme bezüglich des Zugangs zu erforderlicher Hardware, in Anbetracht der verbleibenden Zeit, lediglich drei Modelle umgesetzt werden.
AbschlieSSend kann gesagt werden, dass ein guter Überblick über den momentanen Stand der Forschung gegeben werden konnte. Auch die Evaluierung der Datensets bietet einen schnellen Einstieg in dieses Thema und zeigt die Vielfalt der vorhandenen Sets. Zudem konnte eine Prototyp implementiert werden, welcher zukünftigen Forschungen als Basis dienen kann.

Weiterhin bietet sich an zu untersuchen, wie Machine Learning Verfahren in der Industrie adaptiert werden um diese mit praktischen Hilfestellungen zu unterstützen. Des Weiteren gilt es einen einheitlichen Evaluationsstandard für Datensets zu generieren. Dadurch können weit mehr Forschungen zu dem Thema betrieben werden, da die Implementierung vereinfacht und weniger zeitintensiv durchgeführt werden kann. Zudem ist es schwierig Ergebnisse gleicher Ansätze aber mit unterschiedlichen Daten zu vergleichen. Auch hierbei würden einheitliche Datensets Abhilfe leisten.
Der Prototyp bietet ebenfalls viel Potenzial für weitere Forschungen. Beispielsweise können weitere Modelle eines LSTM entwickelt werden. Des weiteren wäre es interessant andere neuronale Netze wie Gated Recurrent Units (GRU) oder MLP mit dem Ember Datenset zu trainieren. Zusätzlich empfiehlt es sich Untersuchungen mit einem TPU Backend durchzuführen, da dieses die Trainingszeit nochmals verkürzen könnte.

8. Literatur

Bücher und Journals

Aksu, D. und M. Ali Aydin (2018). "Detecting Port Scan Attempts with Comparative Analysis of Deep Learning and Support Vector Machine Algorithms". In: *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*. IEEE, S. 77–80 (siehe S. 39, 40).

Aldwairi, T., D. Perera und M. A. Novotny (2018). "An evaluation of the performance of Restricted Boltzmann Machines as a model for anomaly network intrusion detection". In: *Computer Networks* 144, S. 111–119 (siehe S. 25, 37, 54).

Alswailem, A. u. a. (Mai 2019). "Detecting Phishing Websites Using Machine Learning". In: *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*. IEEE, S. 1–6 (siehe S. 17, 46).

Anderson, H. S., A. Kharkar u. a. (2017). "Evading machine learning malware detection". In: *Black Hat* (siehe S. 43).

Anderson, H. S. und P. Roth (2018). "Ember: an open dataset for training static PE malware machine learning models". In: *arXiv preprint arXiv:1804.04637* (siehe S. 43, 59, 61, 65, 66).

Andress, J. (2019). *Foundations of Information Security*. No Starch Press, S. 248 (siehe S. 15).

Bearden, R. und D. C.-T. Lo (2017). "Automated microsoft office macro malware detection using machine learning". In: *2017 IEEE International Conference on Big Data (Big Data)*. Bd. 2018-Janua. IEEE, S. 4448–4452 (siehe S. 32).

Bonaccorso, G., A. Fandango und R. Shanmugamani (2018). *Python: Advanced Guide to Artificial Intelligence: Expert machine learning systems and intelligent agents using Python*. Packt Publishing Ltd (siehe S. 23).

Brown, A. u. a. (2018). "Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection". In: *Proceedings of the First Workshop on Machine Learning for Computing Systems*, S. 1 (siehe S. 3, 36, 53).

Buczak, A. L. und E. Guven (2016). "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection". In: *IEEE Communications Surveys & Tutorials* 18.2, S. 1153–1176 (siehe S. 27).

Bundeskriminalamt (2018). “Cybercrime, Bundeslagebild 2017”. In: (Siehe S. 1, 2).

Chapman, P. u. a. (1999). “The CRISP-DM user guide”. In: *4th CRISP-DM SIG Workshop in Brussels in March*. Bd. 1999 (siehe S. 7).

Chen, X. u. a. (2019). “A Deep Learning Based Fast-Flux and CDN Domain Names Recognition Method”. In: *Proceedings of the 2019 2nd International Conference on Information Science and Systems - ICISS 2019*. Bd. Part F1483. New York, New York, USA: ACM Press, S. 54–59 (siehe S. 44).

Chio, C. (2017). “Practical Machine Learning in Infosec”. In: (Siehe S. 50).

Choi, S. u. a. (Okt. 2017). “Malware detection using malware image and deep learning”. In: *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, S. 1193–1195 (siehe S. 31, 32).

Cohen, A. u. a. (2016). “SFEM: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods”. In: *Expert Systems with Applications* 63, S. 324–343 (siehe S. 17, 29).

Das, A. u. a. (Dez. 2017). “Detection of Exfiltration and Tunneling over DNS”. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. Bd. 2018-Janua. IEEE, S. 737–742 (siehe S. 35).

Dharmapurikar, S. u. a. (2003). “Deep packet inspection using parallel bloom filters”. In: *11th Symposium on High Performance Interconnects, 2003. Proceedings.* IEEE, S. 44–51 (siehe S. 39).

Ding, Y. und Y. Zhai (2018). “Intrusion Detection System for NSL-KDD Dataset Using Convolutional Neural Networks”. In: *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence - CSAI ’18*. New York, New York, USA: ACM Press, S. 81–85 (siehe S. 36, 37, 52).

e.V., B. (2017). *Cybercrime: Jeder zweite Internetnutzer wurde Opfer*. (Besucht am 01. 10. 2019) (siehe S. 2).

Evans, K. und F. S. Reeder (2010). *A human capital crisis in cybersecurity : technical proficiency matters : a report of the CSIS Commission on Cybersecurity for the 44th Presidency*. November, S. 35 (siehe S. 2).

Garcia, S. u. a. (2014). “An empirical comparison of botnet detection methods”. In: *computers & security* 45, S. 100–123 (siehe S. 38, 39, 54).

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media (siehe S. 67).

Gharib, A. u. a. (2016). “An Evaluation Framework for Intrusion Detection Dataset”. In: *2016 International Conference on Information Science and Security (ICISS)*, S. 1–6 (siehe S. 51, 55, 56).

Glasser, J. und B. Lindauer (2013). “Bridging the gap: A pragmatic approach to generating insider threat data”. In: *2013 IEEE Security and Privacy Workshops*. IEEE, S. 98–104 (siehe S. 47, 58).

Grieco, G. (2016). “Getting started with vulnerability discovery using Machine Learning”. In: (Siehe S. 50).

Habibi Lashkari, A. u. a. (2017). “Characterization of Tor Traffic using Time based Features”. In: *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science und Technology Publications, S. 253–262 (siehe S. 56).

Han, W. u. a. (Jan. 2019). “MalInsight: A systematic profiling based malware detection framework”. In: *Journal of Network and Computer Applications* 125.June 2018, S. 236–250 (siehe S. 10, 42).

He, P. u. a. (2017). “Model approach to grammatical evolution: deep-structured analyzing of model and representation”. In: *Soft Computing* 21.18, S. 5413–5423 (siehe S. 2, 10, 42).

He, Z., T. Zhang und R. B. Lee (2017). “Machine Learning Based DDoS Attack Detection from Source Side in Cloud”. In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, S. 114–120 (siehe S. 32, 33).

Hendler, D., S. Kels und A. Rubin (2018). “Detecting Malicious PowerShell Commands using Deep Neural Networks”. In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security - ASIACCS ’18*. New York, New York, USA: ACM Press, S. 187–197 (siehe S. 35).

Hillary, S. und S. Joshua (2000). “10.1037/e475262008-002”. In: *CrossRef Listing of Deleted DOIs* 1, S. 6 (siehe S. 47).

Hochreiter, S. und J. Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, S. 1735–1780 (siehe S. 67).

Homoliak, I. (2016). “Intrusion Detection in Network Traffic”. Diss. (siehe S. 40, 57).

Homoliak, I. u. a. (Jan. 2019). “Improving Network Intrusion Detection Classiers by Non-payload-Based Exploit-Independent Obfuscations: An Adversarial Approach”. In: *ICST Transactions on Security and Safety* 5.17, S. 156245 (siehe S. 3, 57).

Hu, Z. u. a. (2019). “Reinforcement Learning for Adaptive Cyber Defense Against Zero-Day Attacks”. In: *Adversarial and Uncertain Reasoning for Adaptive Cyber Defense: Control- and Game-Theoretic Approaches to Cyber Security*. Hrsg. von S. Jajodia u. a. Cham: Springer International Publishing, S. 54–93 (siehe S. 3).

Ingeno, J. (2018). *Software Architect’s Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing Ltd (siehe S. 15, 16).

Ioffe, S. und C. Szegedy (2015). “Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: arXiv: arXiv:1502.03167v3 (siehe S. 69).

Javed, A., P. Burnap und O. Rana (Mai 2019). "Prediction of drive-by download attacks on Twitter". In: *Information Processing & Management* 56.3, S. 1133–1145 (siehe S. 16, 44, 45).

Jayaprakash, S. und K. Kandasamy (Apr. 2018). "Database Intrusion Detection System Using Octraplet and Machine Learning". In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. Icicct. IEEE, S. 1413–1416 (siehe S. 40).

Jazi, H. H. u. a. (2017). "Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling". In: *Computer Networks* 121, S. 25–36 (siehe S. 41, 56).

Jeong, Y.-S., J. Woo und A. R. Kang (Apr. 2019). "Malware Detection on Byte Streams of PDF Files Using Convolutional Neural Networks". In: *Security and Communication Networks* 2019, S. 1–9 (siehe S. 3, 48).

Joshua Saxe, H. S. (2018). *Malware Data Science*. No Starch Press, S. 243 (siehe S. 2, 3, 21–23, 67).

Kent, A. D. (2015). "Cybersecurity Data Sources for Dynamic Network Research". In: *Dynamic Networks in Cybersecurity*. Imperial College Press (siehe S. 37, 53).

Kingma, D. P. und J. L. Ba (2015). "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, S. 1–15 (siehe S. 69).

Krohn, J., G. Beyleveld und A. Bassens (2019). *Deep Learning Illustrated: A Visual, Interactive Guide to Artificial Intelligence*. Addison-Wesley Professional (siehe S. 23).

Kumar, A., K. Kuppusamy und G. Aghila (2017). "A learning model to detect maliciousness of portable executable using integrated feature set". In: *Journal of King Saud University - Computer and Information Sciences* 31.2, S. 252–265 (siehe S. 34).

Le, D. C. und A. Nur Zincir-Heywood (2019). "Machine learning based insider threat modelling and detection". In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, S. 1–6 (siehe S. 47, 58).

Leonard, J., S. Xu und R. Sandhu (2009). "A framework for understanding botnets". In: *2009 International Conference on Availability, Reliability and Security*. IEEE, S. 917–922 (siehe S. 38).

Li, Y., Z. Peng u. a. (2016). "Facial age estimation by using stacked feature composition and selection". In: *The Visual Computer* 32.12, S. 1525–1536 (siehe S. 42).

Li, Y., K. Xiong u. a. (2019). "A Machine Learning Framework for Domain Generation Algorithm-Based Malware Detection". In: *IEEE Access* 7, S. 32765–32782 (siehe S. 45).

Maniath, S. u. a. (2017). "Deep learning LSTM based ransomware detection". In: *2017 Recent Developments in Control, Automation & Power Engineering (RDCAPE)*. Bd. 3. IEEE, S. 442–446 (siehe S. 18, 31).

Mathur, L., M. Raheja und P. Ahlawat (2018). "Botnet Detection via mining of network traffic flow". In: *Procedia Computer Science* 132, S. 1668–1677 (siehe S. 18, 38, 54).

Molin, S. (2019). *Hands-On Data Analysis with Pandas*. Packt, S. 740 (siehe S. 25).

More, S. S. und P. P. Gaikwad (2016). "Trust-based Voting Method for Efficient Malware Detection". In: *Procedia Computer Science* 79, S. 657–667 (siehe S. 28).

Nataraj, L. u. a. (2011). "Malware images: visualization and automatic classification". In: *Proceedings of the 8th international symposium on visualization for cyber security*. ACM, S. 4 (siehe S. 43).

Neyolov, E. (2018). "Machine Learning for User Behavior Anomaly Detection". In: HITB Security Conference (siehe S. 48).

Nguyen, C. N. und O. Zeigermann (2018). *Machine Learning–kurz & gut: Eine Einführung mit Python, Pandas und Scikit-Learn*. O'Reilly (siehe S. 22).

Nguyen, L. A. T. u. a. (2013). "Detecting phishing web sites: A heuristic URL-based approach". In: *2013 International Conference on Advanced Technologies for Communications (ATC 2013)*. IEEE, S. 597–602 (siehe S. 46).

Pham, T. S., T. H. Hoang und V. C. Vu (2016). "Machine learning techniques for web intrusion detection A comparison". In: *2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)*. IEEE, S. 291–297 (siehe S. 30, 52).

Pinto, A. (2014). "Secure because Math: A deep-dive on Machine Learning- based Monitoring". In: *Black Hat Briefings* 25.2, S. 1–11 (siehe S. 50).

Polyakov, A. (2018). "Machine Learning and Cybersecurity". In: (Siehe S. 50).

Provos, N. u. a. (2007). "The Ghost in the Browser: Analysis of Web-based Malware." In: *HotBots* 7, S. 4 (siehe S. 44).

Qin, Y. u. a. (Sep. 2018). "Attack Detection for Wireless Enterprise Network: a Machine Learning Approach". In: *2018 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*. IEEE, S. 1–6 (siehe S. 41, 42).

Raff, E., J. Sylvester und C. Nicholas (2017). "Learning the PE Header, Malware Detection with Minimal Domain Knowledge". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security - AISec '17*. New York, New York, USA: ACM Press, S. 121–132 (siehe S. 34, 35).

Rahalkar, S. (2018). *Network Vulnerability Assessment: Identify security loopholes in your network's infrastructure*. Packt Publishing Ltd (siehe S. 16).

Raschka, S. und V. Mirjalili (2017). *Python machine learning*. Packt Publishing Ltd (siehe S. 19).

Raschka, S. und V. Mirjalili (2019). *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd (siehe S. 68).

Ravichandiran, S. (2018). *Hands-on Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow*. Packt Publishing Ltd (siehe S. 24).

Rhode, M., P. Burnap und K. Jones (2018). "Early-stage malware prediction using recurrent neural networks". In: *computers & security* 77, S. 578–594 (siehe S. 43).

Robin Tommy, Gullapudi Sundeep, H. J. (2017). "Automatic Detection and Correction of Vulnerabilities using Machine Learning". In: *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, S. 1062–1065 (siehe S. 33).

Sabar, N. R., X. Yi und A. Song (2018). "A Bi-objective Hyper-Heuristic Support Vector Machines for Big Data Cyber-Security". In: *IEEE Access* 6, S. 10421–10431 (siehe S. 3, 38, 39, 52, 55).

Seymour, J. und P. Tully (2016). *Weaponizing data science for social engineering: Automated E2E spear phishing on Twitter*. Techn. Ber. (siehe S. 45).

Shahzad, R. K. und N. Lavesson (2013). "Comparative analysis of voting schemes for ensemble-based malware detection". In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 4.1, S. 98–117 (siehe S. 28).

Sharafaldin, I., A. Habibi Lashkari und A. A. Ghorbani (2018). "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science und Technology Publications, S. 108–116 (siehe S. 39, 55, 56).

Shijo, P. und A. Salim (2015). "Integrated Static and Dynamic Analysis for Malware Detection". In: *Procedia Computer Science* 46.Icict 2014, S. 804–811 (siehe S. 27, 28).

Shiravi, A. u. a. (2012). "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". In: *computers & security* 31.3, S. 357–374 (siehe S. 37, 56).

Sikorski, M. (2012). *Praise for Practical Malware Analysis*, S. 802 (siehe S. 12, 13).

Singla, A. und E. Bertino (Mai 2019). "How Deep Learning Is Making Information Security More Intelligent". In: *IEEE Security & Privacy* 17.3, S. 56–65 (siehe S. 2).

Siracusano, M., S. Shiaeles und B. Ghita (2018). "Detection of LDDoS Attacks Based on TCP Connection Parameters". In: *2018 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, S. 1–6 (siehe S. 17, 41, 56).

Teoh, T. T. u. a. (2018). "Anomaly detection in cyber security attacks on networks using MLP deep learning". In: *2018 International Conference on Smart*

Computing and Electronic Enterprise (ICSCEE). IEEE, S. 1–5 (siehe S. 40, 57).

Uramova, J. u. a. (Nov. 2018). "Infrastructure for Generating New IDS Dataset". In: *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, S. 603–610 (siehe S. 51).

Vinayakumar, R. u. a. (2019). "Robust Intelligent Malware Detection Using Deep Learning". In: *IEEE Access* 7, S. 46717–46738 (siehe S. 43, 61).

Wang, Z. (2015). "The Applications of Deep Learning on Traffic Identification". In: *Black Hat USA* (siehe S. 37).

Webster, J. und R. T. Watson (2002). "Analyzing the Past to Prepare for the Future: Writing a Literature Review." In: *MIS Quarterly* 26.2, S. xiii–xxiii (siehe S. C, 5, 6).

Wilkes, M. V. und B. M. Spatz (1995). *Computing perspectives*. Morgan Kaufmann San Mateo, CA (siehe S. 1).

Yeo, M. u. a. (2018). "Flow-based malware detection using convolutional neural network". In: *2018 International Conference on Information Networking (ICOIN)*. Bd. 2018-Janua. IEEE, S. 910–913 (siehe S. 39).

Yin, C. u. a. (2017). "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks". In: *IEEE Access* 5, S. 21954–21961 (siehe S. 3, 30, 36, 37, 52).

Internetquellen

Bambenek (2019). *OSINT Feeds from Bambenek Consulting*. URL: http://osint.bambenekconsulting.com/feeds/ (besucht am 21. 11. 2019) (siehe S. 46).

Carmen Torrano Giménez, Alejandro Pérez Villegas, G. Á. M. (2010). *CSIC 2010 HTTP dataset*. URL: http://www.isi.csic.es/dataset/ (siehe S. 30, 52).

Cybersecurity, C. I. for (2019). *NSL-KDD Datasets*. URL: https://www.unb.ca/cic/datasets/nsl.html (besucht am 26. 11. 2019) (siehe S. 30, 52).

Kaggle (2015). *Microsoft Malware Classification Challenge*. URL: https://www.kaggle.com/c/malware-classification/data (besucht am 08. 11. 2019) (siehe S. 38, 55).

Keras (2020). *Why use Keras*. URL: https://keras.io/why-use-keras/ (besucht am 29. 01. 2020) (siehe S. 66).

McAfee (2019). *McAfee Labs Threats Report*. Techn. Ber. (siehe S. 2).

MerlinOne (2019). *The History of Digital Content*. URL: https://merlinone.com/history-of-digital-content-infographic/ (besucht am 16. 12. 2019) (siehe S. 18).

Microsoft (2019). *MICROSOFT SECURITY INTELLIGENCE REPORT*. Techn. Ber. URL: https://clouddamcdnprodep.azureedge.net/gdc/gdc6fw6Tl/original (siehe S. 1).

Microsoft Malware Classification Challenge (BIG 2015) Leaderboard (2019). URL: https://www.kaggle.com/c/malware-classification/leaderboard (besucht am 28. 11. 2019) (siehe S. 39).

Netscout (2019). *Network Security Infrastructure Report*. URL: https://www.netscout.com/report/ (besucht am 29. 11. 2019) (siehe S. 56).

Nielsen, M. (2019). *Neural networks and deep learning - Chapter 3 Improving the way neural networks learn*. URL: http://neuralnetworksanddeeplearning.com/chap3.html (besucht am 19. 12. 2019) (siehe S. 26).

Partners, C. R. (2019). *2018 INSIDER THREAT REPORT*. URL: https://crowdresearchpartners.com/insider-threat-report/ (besucht am 24. 11. 2019) (siehe S. 47).

PhishTank (2019). *Join the fight against phishing*. URL: https://phishtank.com/ (besucht am 24. 11. 2019) (siehe S. 47).

Quarkslab (2019). *LIEF - Library to Instrument Executable Formats*. URL: https://lief.quarkslab.com/ (besucht am 05. 12. 2019) (siehe S. 59).

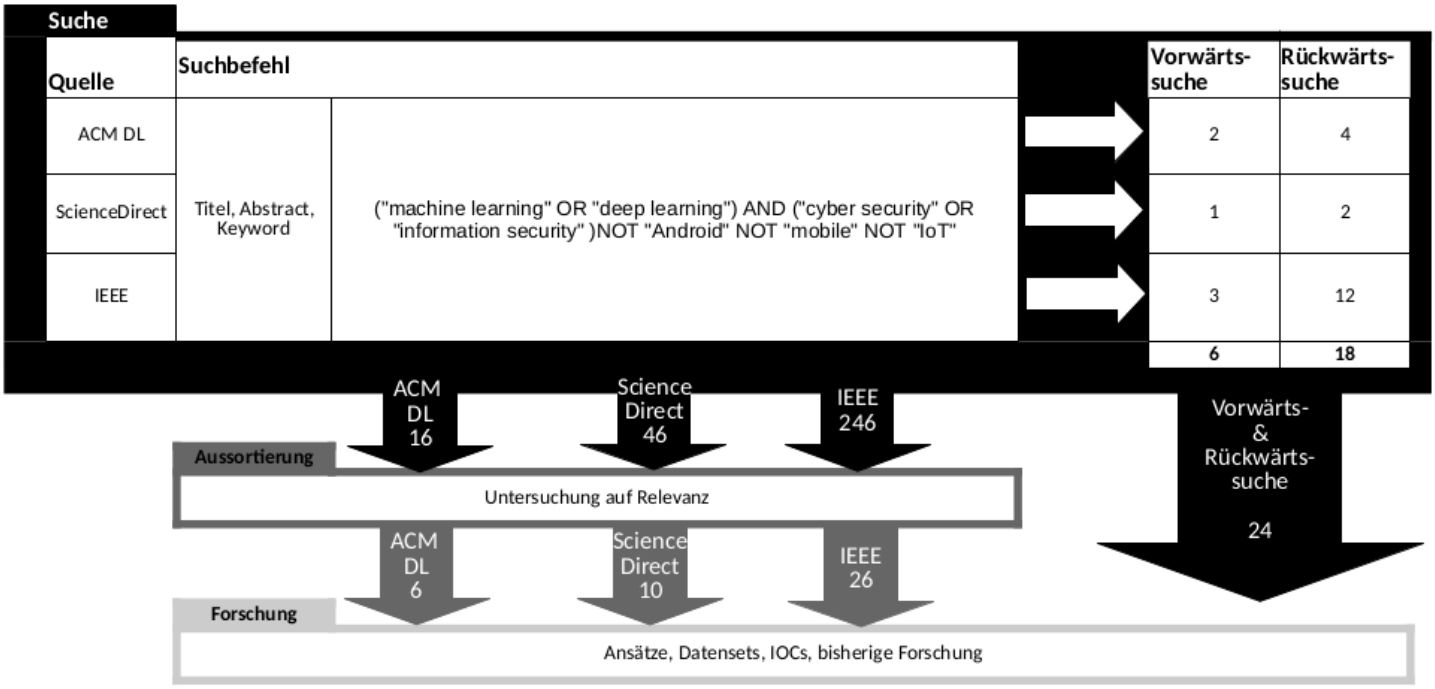
SmartVisionEurope (2015). *CRISP-DM Methodology*. URL: http://crisp-dm.eu/home/crisp-dm-methodology/ (besucht am 18. 10. 2019) (siehe S. 7, 8).

AV-TEST (2019). *Malware Statistics & Trends Report*. URL: https://www.av-test.org/en/statistics/malware/ (besucht am 08. 10. 2019) (siehe S. 2).

Victoria, U. of (2010). *ISOT Botnet Dataset*. URL: https://www.uvic.ca/engineering/ece/isot/datasets/ (besucht am 08. 11. 2019) (siehe S. 38).

A. Anhang

A.1. Research Model



Prozess der Literaturrecherche

A.2. Literaturrecherche

Legende:						
Related Work	Ansatz+Datensatz	Ansatz ohne Datensatz	Datensatz	IOCs	Motivation	
Autor	Titel	Jahr	Zitate Google Scholar	Publisher	Inhalt	Rating
Raff, Edward and Sylvester, Jared and Nicholas, Charles	Learning the PE Header, Malware Detection with Minima lDomain Knowledge	2017	19	ACM	Malware Detection mit minimalem Domänenwissen wobei ein Teil des PE headers extrahiert wird. Neuronale Netze lernen aus unformatierten Bytes ohne explizite Feature Extrahierung.	3
Sewak, Mohit and Sahay, Sanjay K. and Rathore, Hemant	An investigation of a deep learning based malware detection system	2018	5	ACM	Verbesserungen der Vorhersagen des Malicia Datensets durch Neuronale Netze. Allerdings wurde das Malicia Dataset eingestellt.	2
Hendler, Danny and Kels, Shay and Rubin, Amir	Detecting Malicious PowerShell Commands using Deep Neural Networks	2018	13	ACM	Erkennen bösartiger PowerShell Kommandos mit Hilfe Neuronaler Netze und NLPs	2
Ding, Yalei and Zhai, Yuqing	Intrusion Detection System for NSL-KDD Dataset Using Convolutional Neural Networks	2018	0	ACM	Neuronale Netze wurden auf das NSL-KDD Dataset agewandt welches aus rohen tcpdump Daten besteht	3
Brown, Andy and Tuor, Aaron and Hutchinson, Brian and Nichols, Nicole	Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection	2018	13	ACM	Anomalieerkennung in Systemprotokolle durch RNN (recurrent neural networks)	3
Chen, Xunxun and Li, Gaochao and Zhang, Yongzheng and Wu, Xiao and Tian, Changbo	A Deep Learning Based Fast-Flux and CDN Domain Names Recognition Method	2019	0	ACM	Differenzierung von Fast-Flux domain names und CDN (Content Distriution Network) domain names mit Hilfe von deep Learning	2
Lu Xiaofeng, Zhou Xiao, Jiang Fangshuo, Yi Shengwei, Sha Jing	ASSCA: API based Sequence and Statistics features Combined malware detection Architecture	2018	3	ScienceDirect	Malware Detection von system Files in Windows Systemen durch Machine Learning und Deep Learning. Daten von VirusShare und VirusTotal	3
Quan Le, Oisin Boydell, Brian Mac Namee, Mark Scanlon	Deep learning at the shallow end: Malware classification for non-domain experts	2018	13	ScienceDirect	Daten der Microsoft Malware Classification Challenge von Kaggle werden mit CNN bewertet.	3

Ausschnitt der Literaturrecherche Liste

A.3. Python Testskript

```
import ember
import lightgbm as lgb
import numpy as np

lgbm_model = lgb.Booster(model_file="Ember/Data/ember2018/
    ember_model_2018.txt")
print("Is Putty Malware?")
putty_data = open("Ember/putty.exe", "rb").read()
prog1 = ember.predict_sample(lgbm_model, putty_data)
prog1rounded = np.around(prog1, decimals=2)
print (prog1rounded," !Benign!")
print("Is Mimikatz Malware?")
putty_data = open("Ember/x64/mimikatz.exe", "rb").read()
prog2 = ember.predict_sample(lgbm_model, putty_data)
prog2rounded = np.around(prog2, decimals=2)
print (prog2rounded, "!Malicious!")
```