HOST-BASED INTRUSION DETECTION MIT LSTMS

TIM KAELBLE

UNIVERSITÄT
LEIPZIG

Institut für Informatik
Abteilung Datenbanken
Fakultät für Mathematik und Informatik
Universität Leipzig

April 2022

BEGUTACHTER:
Prof. Dr. Erhard Rahm
Dr. Jörn Hoffmann

BETREUER:
Martin Grimmer

ABSTRACT

The increase in zero-day attacks on computer systems poses new challenges for IT security research. Intrusion detection systems (IDSs) are designed to alert operators to any attacks in real time. Anomaly-based IDS is specifically targeted to detect zero-day attacks as well. In this thesis, an anomaly-based Host-based Intrusion Detection System (HIDS) based on Long-Short-Term-Memory neural networks (LSTMs) is developed. For this purpose, the sequence of system call data is investigated. In a further iteration, novel methods are presented with which the sequences are extended by additional system call parameters. This work investigates whether the LSTMs are successful in anomaly-based HIDS. Additionally, it will be investigated which extra parameters can be used to improve the results of the developed algorithm. It was shown in this work that the LSTM approach provides competitive results, but does so at a significant computational cost. The developed additional parameters could achieve a significant improvement of the results and could also find their use in connection with other algorithms.

ZUSAMMENFASSUNG

Die Zunahme von Zero-Day Angriffen auf Computer Systeme stellt die IT-Sicherheitsforschung vor neue Herausforderungen. Intrusion Detection Systems (IDSs) sollen die Betreibenden möglichst in Echtzeit auf jegliche Angriffe hinweisen. Mit anomaliebasierten IDS wird speziell darauf abgezielt auch Zero-Day Angriffe zu erkennen. In dieser Arbeit wird ein anomaliebasiertes Host-based Intrusion Detection System (HIDS) basierend auf Long-Short-Term-Memory neuronalen Netzwerken (LSTMs) entwickelt. Dafür wird die Sequenz von System Call Daten untersucht. In einer weiteren Iteration werden neuartige Verfahren präsentiert, mit welchen die Sequenzen durch weitere System Call Parameter erweitert werden. Es soll im Rahmen dieser Arbeit untersucht werden, ob die LSTMs in der anomaliebasierten HIDS erfolgreich sind. Zusätzlich soll untersucht werden, welche zusätzlichen Parameter verwendet werden können um die Ergebnisse des entwickelten Algorithmus zu verbessern. Es konnte im Rahmen dieser Arbeit gezeigt werden, dass der LSTM-Ansatz konkurrenzfähige Ergebnisse liefert, dies aber mit einem erheblichen Berechnungsaufwand. Die entwickelten zusätzlichen Parameter konnten eine deutliche Verbesserung der Ergebnisse erzielen und könnten auch im Zusammenhang mit anderen Algorithmen ihren Einsatz finden.

AKRONYME

ADFA-LD ADFA Linux Dataset

AE Auto Encoder

AI Artificial Intelligence

BOSC Bag of System Calls

BSI Bundesamt für Sicherheit in der Informationstechnik

CNN Convolutional Neural Net

CVE Common Vulnerabilities and Exposures

CWE Common Weakness Enumeration

DR Detektionsrate

FN False Negative

FP False Positive

HIDS Host-based Intrusion Detection System

IDS Intrusion Detection System

LID-DS Leipzig Intrusion Detection Dataset

LSTM Long-Short-Term-Memory neuronales Netzwerk

ML Maschinelles Lernen

MLP Multilayer Perceptron

NLP Natural Language Processing

NIDS Network Intrusion Detection System

OHE One-Hot-Encoding

RNN rekurrentes neuronales Netzwerk

SIDS Signature-based Intrusion Detection Systems

STIDE Sequence Time-Delay Embedding

SW Sliding Window

TCF Thread Change Flag

TIDE Time-Delay Embedding

TN True Negative

W2V Word2Vec

INHALTSVERZEICHNIS

TABELLENVERZEICHNIS

ABBILDUNGSVERZEICHNIS

1 EINFÜHRUNG

1.1 EINLEITUNG

Laut des *X-Force Thread Intelligence Index 2022* von IBM [69] gibt es eine klare Steigerung von Malware mit einzigartigem Quelltext. Dies gilt auch speziell für Linux Systeme, wobei der größte Anstieg bei den Banking-Trojanern zu beobachten ist. Dort konnte eine Verzehnfachung von neuartigen Angriffen verzeichnet werden. Die Zunahme der Angriffe auf Linux Systeme hängt laut IBM auch damit zusammen, dass Organisationen zunehmend auf Cloud-Umgebungen setzen, welche häufig auf Linux Systeme angewiesen sind. Einzigartige und damit meist unbekannte Angriffe abzuwehren stellt Unternehmen vor große Schwierigkeiten. Die häufig verwendeten, auf Signaturen basierenden Abwehrmechanismen reichen nicht aus diese drohende Gefahren abzuwenden. Im Gegensatz zu dem signaturbasierten Ansatz, müssen die Angriffe bei einem anomaliebasierten Ansatz nicht vorher bekannt sein. Stattdessen wird ein Modell des zu erwartenden Normalverhalten des Systems erstellt. Mit dem erstellten Modell sollen dann, möglichst in Echtzeit, Abweichungen bzw. Anomalien des erwarteten Verhalten signalisiert werden. Die Bedeutsamkeit der Erkennung von bisher unbekannten Angriffen wird ebenfalls durch das Bundesamt für Sicherheit in der Informationstechnik (BSI) bestätigt. Das BSI berichtet, dass im Berichtzeitraum die Schadprogramm-Varianten um rund 144 Millionen zugenommen haben, was einer Steigerung von 22% gegenüber dem Zeitraum des vorigen Berichts bedeutet [36]. Laut eines weiteren Berichts von IBM [12] konnten durch den Einsatz von Security-Artificial Intelligence (AI) und Automatisierung die durchschnittlichen Kosten eines erfolgreichen Angriffs von 6,71mio USD auf 2,9mio USD gesenkt werden. Dies spricht für die weitere Entwicklung von AI-basierter Angriffserkennung und insbesondere anomaliebasierte Angriffserkennung mit Unterstützung durch AI.

Juni 2020 bis Mai 2021

Das zugrunde liegende Verhalten von Computersystemen kann auf verschiedene Weisen beschreiben werden. Speziell werden in dieser Arbeit Host-based Intrusion Detection Systems (HIDSs) verwendet, da sie gegenüber den Network Intrusion Detection Systems (NIDSs) auch interne Attacken erkennen können. Häufig verwendete Informationsquelle dafür sind zum Beispiel Systemlogs [31]. In dieser Arbeit werden hingegen System Calls verwendet. Denn jegliche Nutzerprogramme lösen betriebssystemspezifische System Calls aus, welche über verschiedene Tools wie zum Beispiel Sysdig [74] ausgelesen werden können. Sie bieten somit eine sehr abstrakte Beschreibung des

Systems auf Betriebssystemebene. Zudem sind sie im Gegensatz zu den System Logs nicht durch die Einstellungen der Entwicklerinnen abhängig. Eine Schwierigkeit besteht jedoch mit dem Umgang der großen Datenmengen, die selbst bei kleineren Anwendungen entstehen.

Die Probleme in der Verarbeitung von sehr großen Datenmengen konnten allerdings, unter anderem durch die Verwendung selbst lernender Algorithmen, erfolgreich angegangen werden. Gerade in der Natural Language Processing (NLP) gab es große Fortschritte in der Verarbeitung von sehr großen Datenmengen. System Calls haben ebenso ein begrenztes Vokabular und sie besitzen einen semantischen Zusammenhang, womit ihnen eine gewisse Ähnlichkeiten zu natürlichen Sprachen zugesprochen werden kann. Die Übertragung von NLP-Methoden auf anomaliebasierte HIDS wurde bereits in einigen Arbeiten vorgenommen und soll auch hier weiter verfolgt werden.

Bereits in verschiedenen Arbeiten wurden die Abfolge von System Calls betrachtet. Eine der ersten Arbeiten, welche die Sequenzen von System Calls betrachtet, stammt aus dem Jahr 1996 und wurde von Forrest et al. [20] veröffentlicht. Maggi et al. verwenden zusätzlich auch Parameter und verweisen in ihrer Arbeit [49] auf verschiedene Ansätze. Mit der Verwendung von zusätzlichen Parametern, wie zum Beispiel Dateipfaden oder Rückgabewerten, können die Sequenzen der System Call Namen erweitert werden. Zwar wird der Informationsgehalt jedes System Calls erhöht, jedoch nimmt die Komplexität der Sequenzen damit wesentlich zu. Es muss also eine Abwägung zwischen Informationsgehalt und Komplexität bei der Entwicklung der zusätzlichen Parameter erfolgen. Im Folgenden wird zusammengefasst, wie auf der Grundlage der bereits durchgeführten Forschung ein neuer Ansatz zur Erkennung von Abweichungen vom normalen Systemverhalten entwickelt wird.

1.2 ZIELSETZUNG

Long-Short-Term-Memory neuronale Netzwerke (LSTMs) sind in der NLP aber auch in der Analyse von Zeitreihendaten ein etabliertes Verfahren. Sie haben den Vorteil auch Zusammenhänge mit größerer zeitlicher Verzögerung zu erkennen [33]. In dieser Arbeit soll ein Verfahren entwickelt werden, mit dem LSTMs genutzt werden um ein Sprachmodell der System Call Daten zu erstellen. Anhand dieses gelernten Modells, soll dann eine Einstufung in Normalverhalten und Angriffsverhalten erfolgen. Des Weiteren wird die Hinzunahme von Parametern, wie zum Beispiel den Rückgabewert, mit Hinblick auf die Erkennungsquote und die Anzahl der Fehlalarme des Intrusion Detection System (IDS) untersucht.

Zusammenfassen lässt sich diese Zielsetzung mit den folgenden Forschungsfragen:

- Kann der Erfolg von LSTM-Netzwerken auf die Erkennung von Anomalien in der Cyber-Sicherheit übertragen werden?
- Kann die Zunahme von Parametern bei der Anomalieerkennung mittels System Calls eine Verbesserung bringen?

  → Welche Parameter kommen dafür in Frage?

Um diese Forschungsfragen angemessen behandeln zu können, müssen zunächst Grundlagen aus verschiedenen Bereichen gelegt werden. Zum einen werden in Kapitel 2 unterschiedliche Herangehensweisen zur Überwachung von Systemen betrachtet. Dabei wird erläutert wieso es für diese Anwendung sinnvoll ist einen Host-basierten Ansatz zu wählen. Speziell soll auch beschrieben werden, warum sich System Calls zur Überwachung von Computersystemen eignen. Des Weiteren müssen Grundlagen für die in dem Algorithmus verwendeten Techniken gelegt werden. Dazu gehören hauptsächlich Grundlagen zu Rekurrente neuronale Netzwerke (RNNs) sowie die Erweiterungen dieser, die LSTMs. Der Kontext in welchem diese Arbeit steht wird in Kapitel 3 vorgestellt. Ein großer Teil der Implementierungsarbeit stellt die Vorverarbeitung der Daten dar. Diese soll mit der genaueren Untersuchung der Zusammensetzung der Techniken für den Algorithmus in Kapitel 4 dargestellt werden. In Kapitel 5 wird dann eine Auswertung auf dem Leipzig Intrusion Detection Dataset (LID-DS) [26] präsentiert. Dieser Datensatz bietet den Vorteil, dass zusätzlich die System Call Parameter, wie zum Beispiel die *Thread ID*, zur Verfügung gestellt werden.

In Kapitel 6 der Arbeit soll dann eine Schlussfolgerung aus den zuvor gewonnenen Ergebnissen gezogen werden. Hauptsächlich sollen die gestellten Forschungsfragen untersucht werden. Konnte mit einem hinzugezogenen Parameter ein Mehrwert erzielt werden? Bieten sich LSTM-Netzwerke auch für die Anomalieerkennung im IT-Sicherheitsbereich an?

Abschließend werden die Vorgehensweisen und Erkenntnisse in Kapitel 7 zusammengefasst. Auf Grundlage dieser Erkenntnisse wird dann ein Ausblick auf mögliche Entwicklungen und identifizierte Chancen gegeben.

2 GRUNDLAGEN

In den folgenden Abschnitten wird die Grundlagen betrachtet, welche für die Umsetzung eines IDS nötig sind. Zu Beginn soll in Unterabschnitt 2.1.1 besprochen werden, welche Analysetechniken für die Überwachung von Systemen zur Verfügung stehen. In Unterabschnitt 2.1.2 wird dann auf mögliche Orte der Erfassung und in Abschnitt 2.2 auf die eigentliche Daten für die Überwachung, die System Calls, eingegangen. Nachdem so eine allgemeine Herangehensweise an die Erkennung von Angriffen dargelegt wurde, soll im Anschluss die Grundlagen des in der Arbeit verwendeten Algorithmus untersucht werden. Dazu werden in Abschnitt 2.3 RNN, sowie die Erweiterung der RNNs die LSTM neuronalen Netzen beschrieben.

2.1 Intrusion Detection Systems (IDSs)

Eine *Intrusion* ist eine unautorisierte Aktivität welche zu Missbrauch eines Systems führen kann. Sie sorgt für ein zum Normalverhalten abweichendes, anomales Verhalten und kann durch diese Abweichung einem System großen Schaden zufügen. Ein IDS überwacht Ressourcen, wie zum Beispiel Netzwerkdaten und versucht automatisch Missbrauch und anomales Verhalten zu identifizieren, um dadurch die Sicherheit des Systems gewährleisten zu können. [15]

zu dt. Eindringen, Einbruch

Um solche unautorisierten Aktivitäten erkennen zu können, müssen Daten erfasst und anschliessend analysiert werden. In den folgenden beiden Abschnitten werden verschiedene Ansätze dieser beiden Schritte, also Datenanalyse und Datenerfassung, genauer untersucht.

2.1.1 *Datenanalyse*

Das Erkennen von Angriffen auf Computersystemen mittels IDS wird im wesentlichen in zwei Kategorien eingeteilt [39, 45, 56]. Dazu zählen die signaturbasierten und anomaliebasierten Verfahren auf die im Folgenden eingegangen wird.

SIGNATURBASIERT Signature-based Intrusion Detection Systems (SIDS) versuchen zuvor bekannte Muster von Angriffen in den zu überwachenden Systemen wiederzuerkennen. Dies basiert auf einer ausgeprägten Datenbank an Signaturen von bekannten Angriffen. Die aktuelle Signatur von zum Beispiel einer Datei wird dann mit einer Datenbank verglichen. Es gibt diverse Möglichkeiten wie diese Signaturen erstellt werden. In mehreren Arbeiten werden dafür zum

Beispiel Netzwerk Pakete betrachtet [39] und in wenigen auch *State Machines* erstellt [51].

zu dt. Zustandsautomaten

SIDS haben meist eine sehr hohe Genauigkeit, doch ein wesentlicher Nachteil der signaturbasierten IDS liegt in der Tatsache begründet, dass keinerlei neuartigen Angriffe, sowie viele Abwandlungen von schon bekannten Angriffen nicht erkannt werden. Das liegt daran, dass für diese Fälle noch keine Signaturen in der Datenbank vorhanden sind. [15] Doch wie in Abschnitt 1.1 bereits beschrieben, wird die Gefahr von Zero-Day Angriffen größer als sie bereits ist. Weswegen im Folgenden die anomaliebasierte Ermittlung von Intrusions beschrieben wird, die diese Probleme angehen kann.

ANOMALIEBASIERT Der wesentliche Vorteil Gegenüber den SIDS besteht darin, dass auch bisher unbekannte Angriffe erkannt werden können. Denn anstatt sich auf bekannte Angriffe zu berufen, werten anomaliebasierte IDSs lediglich eine Abweichung eines wohldefinierten Normalverhaltens als Angriff. Das Normalverhalten oder anders formuliert das Modell eines Systems im Normalzustand soll möglichst akkurat das zu erwartende und damit gewollte Systemverhalten beschreiben [35]. Anomalien oder auch *„outlier“* wurden unter anderem schon 1969 definiert.

zu dt. Ausreißer

„An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs.“ [29]

Also eine Anomalie oder wie hier beschrieben eine abweichende Beobachtung, ist eine Beobachtung, die deutlich von den anderen Datenpunkten innerhalb der Stichprobe abweicht. Anomalien speziell in Daten von Computersystemen können aus verschiedenen Gründen entstehen, zum Beispiel durch böswillige Aktivitäten oder aber auch durch Programmfehler. Anomalien dieses Verhaltens können also zu signifikanten und oft kritischen Veränderungen eines Systems führen. So kann eine Anomalie in Netzwerkdaten dafür stehen, dass ein gekaperter Computer sensitive Daten an ein unautorisiertes Ziel sendet [43]. Genau dieser signifikante Unterschied des aktuellen Systemzustand zu einem entworfenen Modell (wohldefiniertes Normalverhalten) zu einem bestimmten Zeitpunkt wird dann als Anomalie eingestuft. Jede Anomalie gilt dann wiederum als *Intrusion*. Zusammengefasst besteht also die Grundannahme dieser Methode darin, dass Intrusions von dem gelernten Normalverhalten des Systems unterschieden werden können.

Anomaliebasierte IDSs werden im Einsatz in zwei Phasen aufgeteilt, die *Trainingsphase* sowie die *Testphase*. Im ersten Schritt wird ein Modell des Normalverhaltens erstellt. Dieser Schritt wird in Maschinelles Lernen (ML)-basierte, Statistik-basierte [15] oder auch wie in manchen Arbeiten [39] zusätzlich in *Knowledge*-basierten Algorithmen untergliedert. In der zweiten Phase muss dieses Modell dann überprüft werden. Speziell soll dabei mit noch nicht betrachteten Daten getestet

zu dt. Wissens-basierte

werden ob Normalverhalten und Anomalien auch als solche eingestuft werden. Dabei liefert der Algorithmus für jede Eingabe entweder einen Anomaliescore oder direkt ein Label. Das Label gibt an ob es sich um eine tatsächliche Intrusion handelt oder nicht. Liegt der Score über einem festgelegten Schwellwert, gilt die Eingabe als anormal ansonsten als normal. [15]

Jedoch ergibt sich hier im Vergleich zu den SIDS eine Problematik. Der erwähnte Schwellwert entscheidet über die Ergebnisqualität der anomaliebasierten IDSs und sollte daher sorgfältig und am besten automatisch ermittelt werden, um anwendungsspezifische Justierungen zu vermeiden.

mehr dazu in Abschnitt 4.4

Die Trainingsphase wird von Chandola et al. [10] weiter unterteilt in *Supervised*, *Semi-supervised* sowie *Unsupervised Anomaly Detection*. Diese unterscheiden sich hauptsächlich in der Anforderung an die Trainingsdaten.

zu dt. überwacht, unüberwacht

Bei einer *Supervised* Anomalieerkennung werden Trainingsdaten verwendet, in welchen Anomalien sowie auch Normalverhalten gelabelt sind. Somit werden auch Muster von Angriffen gelernt. Doch kann es dadurch beim Lernprozess zu Schwierigkeiten kommen, da häufig ein Ungleichgewicht zwischen der Datenmenge an Angriffs- und Normalverhalten besteht. [67] Zusätzlich sind Anomalien meist unbekannt, womit das Erstellen eines sinnvollen Datensatzes erschwert wird. Auch deswegen ist diese Herangehensweise weniger verbreitet. [24]

Hingegen wird beim *Semi-supervised* Verfahren ein Trainingsdatensatz benötigt, welcher lediglich das Normalverhalten kennzeichnet. Somit werden auch keine Angriffe zum Trainingszeitraum gesehen. Die Grundidee besteht also darin Abweichungen eines Modells des Normalverhaltens zu erkennen.

Bei der *Unsupervised Anomaly Detection* werden Daten verwendet welche keine Labels beinhalten. Dabei wird davon ausgegangen, dass Normaldaten sehr viel umfangreicher in den Daten vertreten sind als Anomalien. Sonst kann es zur Testphase zu häufigen Fehlalarmen kommen, wenn Anomalien fälschlicherweise als Normalverhalten eingestuft werden. [65]. Jedoch sind auch *unsupervised* Trainingsverfahren umsetzbar, welche keine Anomalien in den Daten enthalten.[24]

Die Wahl des Algorithmus zur Erkennung von Anomalien in den Daten ist dabei entscheidend für den Ablauf der Trainingsphase und wird deshalb auch im Bezug auf das verwendete Verfahren in Unterabschnitt 4.4.2 beschrieben.

Insgesamt ergeben sich bei der Umsetzung der Anomaliedetektion allerdings auch einige Schwierigkeiten, welche im folgenden zusammengefasst werden:

- *Definition des Normalverhaltens*: Ziel ist es jegliches Verhalten, welches kein anomales beinhaltet, zu erfassen. So muss dafür gesorgt werden, dass das Normalverhalten auch tatsächlich in den Daten widergespiegelt wird.

- *Dynamik des Normalverhaltens*: Normalverhalten kann sich über die Zeit verändern und somit vom Algorithmus Gelerntes unbrauchbar machen [10].
- *Unzulängliche Datensätze*: Daten zur Erfassung des Normalverhaltens sind oft veraltet oder nicht sehr detailreich.
- *Findung eines sinnvollen Schwellwerts*: Festlegung eines Schwellwertes, welcher die eigentliche Unterscheidung zwischen Normalverhalten und Angriffsverhalten umsetzt.

Mehr dazu in Unterabschnitt 2.2.3

Die Funktionalität eines anomaliebasierten IDS steht und fällt also mit den für das Training bereitgestellten Daten. Daher ist eine entscheidende Frage bei Verwendung von anomaliebasierten IDSs: Woher kommen die Daten, die für das Training des Normalverhalten benötigt werden. Und kann gewährleistet werden, dass diese Daten auch das Normalverhalten genügend präzise beschreiben können.

2.1.2 *Datenerfassung*

Die Datenerhebung wird in verschiedenen Arbeiten in zwei Kategorien unterteilt [39, 45]. Zum einen die HIDS mit Datenerfassung auf Host-Ebene und zum anderen die NIDS bei welchen die Daten auf Netzwerkebene betrachtet werden. In den folgenden Abschnitten werden diese beiden Ansätze genauer untersucht.

NETWORK BASED INTRUSION DETECTION Die Grundidee hierbei besteht darin, dass ein Angriff immer den Zugriff auf die Computersysteme von außen erlangen möchte. Dafür wird der Netzwerkverkehr über Pakete, NetFlow oder andere Netzwerkdaten überwacht. Ein großer Vorteil daran ist, dass viele Computer in einem Netzwerk gleichzeitig überwacht werden. Ziel ist es dabei Angriffe möglichst früh zu erkennen und zu verhindern, dass sich die Gefahr weiter ausbreiten kann. Die Umsetzung von NIDS ist bei besonders großen Netzen erschwert, da der hohe Datendurchsatz das Erkennen von Angriffen erschwert. [4]

Ein weiterer Nachteil bei der Untersuchung von Netzwerkpaketen oder ähnlichem besteht darin, dass der Netzwerkverkehr meist verschlüsselt ist und somit nicht auf den Inhalt der Pakete eingegangen werden kann.

HOST BASED INTRUSION DETECTION Wie der Name bereits impliziert, konzentrieren sich HIDSs auf die Untersuchung von Daten, welche auf dem Host basieren. Es wird versucht das dynamische Verhalten, sowie den Zustand des Systems zu überwachen und dies nur mit Informationen, die auf dem Host zugänglich sind. In der Literatur werden hierfür verschiedene Informationsquellen genutzt.

Dazu gehören verschiedene Logs, z.B Firewall und Database Logs [39], oder aber auch Daten aus dem Kernel wie z.B. System Calls [49]. Im Gegensatz zur NIDS kann hier auf den Inhalt von jeder Information eingegangen werden, da die interne Kommunikation unverschlüsselt stattfindet. [7]

HOST-DATEN ZUR BESCHREIBUNG DES SYSTEMVERHALTENS Nach Bridges et al. [9] besteht im wesentlichen die Auswahl zwischen Log-Files und *System Calls*. In der Übersichtsarbeit wird auch noch Literatur besprochen, welche Windows Registry, File System, Programm Analysedaten untersuchen, welche im Folgenden aber nicht mit einbezogen werden.

zu dt. Systemaufrufe

Log-Files können weiter aufgeteilt werden in System-Logs und programmspezifischen Logs. Wobei System-Logs nur durch das Betriebssystem geschrieben werden. Beide Log Arten beschreiben das Systemverhalten, jedoch findet bei beiden bereits eine Filterung von Informationen des Normalverhaltens statt.Denn es muss manuell bestimmt werden welche Aktionen überhaupt geloggt werden und stellen deshalb auch keine vertrauenswürdige Quelle dar [14]. System Calls hingegen sind klar definiert und beschreiben das Systemverhalten zunächst ungefiltert, unterliegen dafür allerdings einer weitaus größeren Varianz. [9] Für die feingranulare Aufzeichnung von Log-Files sowie für System Calls gilt, dass die Aufzeichnung sehr rechenintensiv werden kann. Mit modernen Softwarelösungen wie zum Beispiel sysdig [74], kann der Berechnungsaufwand für viele Anwendungsbereiche in akzeptable Bereiche gebracht werden. In dieser Arbeit wird auf die Filterung durch Logs verzichtet und auf die detailliertere Darstellung des Systemverhaltens durch System Calls gesetzt.

mehr dazu in Abschnitt 2.2

Abbildung 2.1 soll einen Überblick über die in den vorigen Abschnitten gemachte Einstufung geben. Dabei gibt die Abbildung die Strukturierung der vorigen Abschnitte wieder, in welcher die Datenerfassung in die Host-basierte und Netzwerk-basierte Erfassung unterteilt wird, sowie die Datenanalyse in die anomaliebasierten und signaturbasierten Verfahren eingeteilt wird. Die dicker umrandeten Verfahren werden in dieser Arbeit verwendet. Also zur Datenerfassung werden nur Informationen, welche auf dem Host zugänglich sind verwendet, und die so erhaltenen Daten werden anomaliebasiert untersucht. Es stellen sich beim designen von anomaliebasierten HIDS zwei Hauptfragen:

- Mit welchen Daten kann das Systemverhalten möglichst präzise dargestellt werden?
  - Logs, System Calls, Registry ... [9]
- Wie wird die eigentliche Anomalie in den Daten erkannt?

Die letztere Frage soll speziell in Abschnitt 4.4 beleuchtet werden, doch wie das Systemverhalten präzise dargestellt werden kann, wird

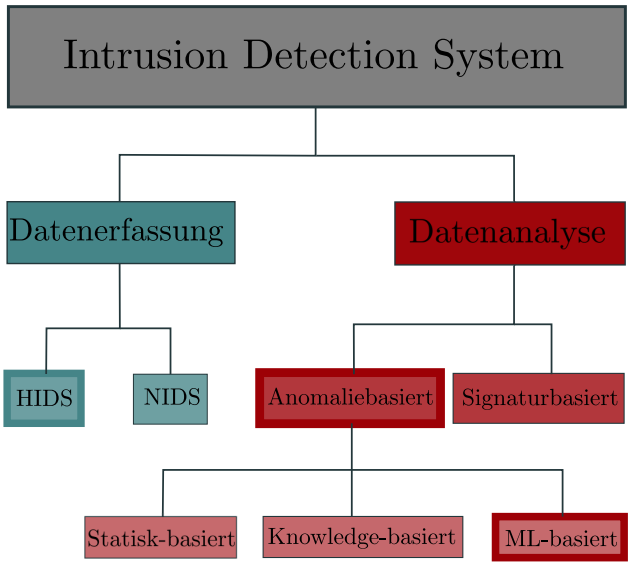

Abbildung 2.1: Einordnung des verwendeten IDS (Relevant für diese Arbeit dicker markiert).

nun behandelt. In dieser Arbeit werden System Calls als Beschreibung des Systemverhalten verwendet. Im Folgenden sollen Grundlagen der System Calls besprochen werden. Zusätzlich soll untersucht werden, welche Informationen neben der eigentlichen Sequenz der System Calls zur Verfügung stehen.

2.2 SYSTEM CALLS

Jegliche Programme die auf einem Rechner mit einem Betriebssystem laufen, müssen mit diesem interagieren um Veränderungen am System vornehmen zu können. Diese Interaktion findet in Form von *System Calls* statt und kann mit einer gewöhnlichen Programm Schnittstelle verglichen werden. Beispielhaft werden in Tabelle 2.1 zwei System Calls von Linux Betriebssystemen, ihre Argumente und die Art ihrer Rückgabewerte beschrieben.

zu dt. Systemaufrufe

System Calls			
Name	Beschreibung	Argumente	Rückgabewerte
open	Öffnet die in *path* spezifizierte File und gibt einen *file descriptor* zurück.	*path*, *flags*, *mode*	File descriptor oder Fehlerwert
write	Schreibt bis zu *count* Bytes aus dem Buffer (ab Stelle *buf*) in die File, welche über den *file descriptor* fd definiert wird.	fd, $*buf$, *count*	Geschriebene Bytes oder Fehlerwert

Tabelle 2.1: Kurzbeschreibung ausgewählter System Calls

2.2.1 *Allgemeines*

Generell werden System Calls verwendet, um vom Betriebssystem zur Verfügung gestellte Funktionalitäten auszuführen. Das Betriebssystem, oder noch genauer der *Kernel* des Betriebssystems, stellt verschiedene Services bereit, welche dann von Programmen genutzt werden können. Die System Calls stellen dabei die Kommunikation zwischen dem Kernel und den darauf laufenden Programmen dar. Zu den Services gehören verschiedene Bereiche der Prozesskontrolle, das Datei- und Geräte-Management, Informationspflege und Kommunikationsaufbau und zugehörige Aufgaben. Geschrieben werden diese Funktionalitäten in C, C++ oder auch in Assembly. Üblicherweise können System Calls nur von Nutzerprozessen, also aus dem *User space*, aufgerufen werden, diese besitzen eine eingeschränkte Berechtigungen. In Abbildung 2.2 wird der Ablauf dargestellt, in welchem ein Programm aus dem User space über einen System Call eine privilegierte Aktion ausführen lassen kann. Dies ermöglicht es Nutzerprozessen auf eine limitierte Auswahl an privilegierten Funktionen aus dem Kernel zugreifen zu können. [21]

zu dt. Betriebssystemkern

zu dt. Benutzer-Modus/Benutzerprozesse

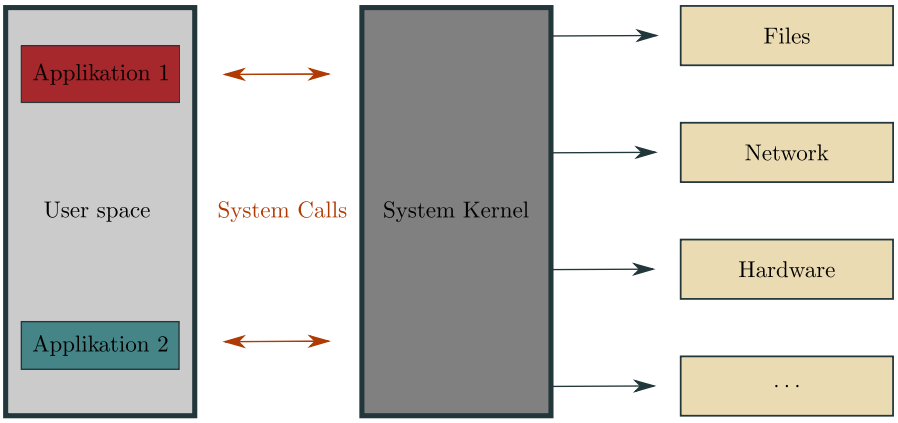

Abbildung 2.2: Darstellung der Kommunikation zwischen Applikationen aus dem User space mit dem System Kernel und dem Kernel mit Dateien/Netzwerk/Hardware/...

Wie in Abbildung 2.2 ebenfalls angedeutet, besteht ein System Call aus einem Aufruf an das Betriebssystem und der Antwort des Betriebssystems. In Tabelle 2.1 wird an zwei Beispielen beschrieben welche System Calls Argumente beim Aufrufen mitgegeben werden können. Das können neben bestimmten Flags, welche die Funktion der System Calls bestimmen, auch zum Beispiel Dateipfade oder IP-Adressen sein. [54] Die Betriebssystemantworten auf einen System Call haben zusätzlich in den Argumenten einen Rückgabewert . Zu den Rückgabewerten gehören unter anderem diverse Fehlerwerte, aber auch zum Beispiel bei *read* oder *write* System Calls die Menge an gelesenen oder geschriebenen Bytes. Generell wird bei erfolgreicher Durchführung ein nicht-negativer und bei einem Fehler ein negativer

Gekennzeichnet mit „res“

Wert zurückgegeben [57]. Zu einem System Call gehört also mehr als nur der Funktionsname.

Ein Angreifer muss also, um Schaden anzurichten, Veränderungen an den System Calls vornehmen. Der folgende Abschnitt soll einen Überblick verschaffen, wie die System Calls für ein IDS verwendet werden können.

2.2.2 *System Calls für IDSs*

Viele verschiedene IDS Ansätze betrachten die Sequenz von System Calls und erzielen vielversprechende Ergebnisse. [49] Häufig werden dabei aber nicht die in den Argumente der System Calls enthaltene Information mit einbezogen und bieten damit einen Spielraum für Angriffe. Verschiedene Arbeiten [75, 76, 79] zeigen, wie dieser Spielraum ausgenutzt werden kann um unerkannt Angriffe durchzuführen. Tan et al. [76] erreichen dies durch die Veränderung eines zuvor von der IDS erkannten Angriffes. Dabei werden die dem IDS fremden Sequenzen auseinander gezogen und mit bekannten Sequenzen aufgefüllt. Zur Auswertung wurde der Sequence Time-Delay Embedding (STIDE) Algorithmus verwendet. Ein weiterer Ansatz versucht lediglich die System Call Argumente zu verändern, ohne dabei die Sequenz zu beeinflussen [79]. Was diese Beispiele jedoch zeigen, ist, dass wenigstens einer dieser Faktoren, also entweder die Sequenz von System Calls, verändert werden muss oder es werden die Argumente der System Calls verändert. So kann zum Beispiel anstatt auf den Pfad „/tmp/some/file“ auf „/etc/passwd“ zugegriffen werden. Welche dieser Argumente und wie diese genutzt werden können um die Anomalieerkennung zu verbessern soll in Unterabschnitt 4.2.2 untersucht werden. In dem nachfolgenden Kapitel wird nun auf verschiedene Datensätze, welche System Call Sequenzen und teilweise Argumente und weiter Metadaten enthalten, eingegangen.

2.2.3 *Datensätze*

Seit 1998 einer der ersten System Call Datensätze für HIDS veröffentlicht wurde [46, 48], kamen über die Jahre verschiedene weitere Datensätze hinzu. Auf diese wird in den kommenden Abschnitten kurz eingegangen. Dabei soll auch auf die Nutzbarkeit und die entstehenden Problematiken dieser für die HIDS über System Calls eingegangen werden.

DARPA Der unter anderen von der *Defence Advanced Research Project Agency*, kurz DARPA, erstellte Datensatz KDD-99 [48] lieferte einen der ersten Benchmark Datensätze für HIDS. Er simuliert ein militärisches Netzwerk bestehend aus drei Systemen mit unterschiedlichen Betriebssystemen und Services. Diese Systeme erzeugen mit wech-

selnden IP-Adressen Traffic, welcher insgesamt fünf Wochen über TCP-Dump aufgezeichnet wurde. Dabei werden verschiedene Angriffe ausgeführt, darunter sind *Denial of Service* und *User to Root*. Der Datensatz steht auf Grund verschiedener Unzulänglichkeiten schon länger in der Kritik [19, 77, 78]. Unter anderem beschreiben McHugh et al. [50], dass eine Unausgewogenheit zwischen Angriffs- und Normaldaten bestehen. Zum Beispiel gibt es Aufnahmetage, an welchen bis zu 76% der Daten Angriffsdaten entsprechen, was laut McHugh keine realistische Verteilung ist. Eine der größten Kritikpunkte, welcher auch von Tavallaee et al. [77] und Engen [18] aufgefasst wird, besteht in der Redundanz der Aufnahmen. So haben Tavallaee et al. alle mehrfach vorkommenden Aufnahmen entfernt und damit 78.05% der Trainingsdaten und 17.15% der Testdaten entfernt. Gerade bei selbstlernenden Systeme kann dadurch ein ungewollter Bias entstehen. Des Weiteren ist der Datensatz mittlerweile stark veraltet (1999/1998).

Auch als Priviledge Escalation bezeichnet

UNM Der *University of New Mexico* Datensatz stammt aus dem Jahr 2004 und beinhaltet die Aufzeichnung von System Calls diverser Programme, welche alle Administratorenrechte besitzen. Dabei wurden verschiedene Angriffe, wie zum Beispiel *Buffer Overflows* ausgeführt. Auch dieser Datensatz [19] ist mittlerweile veraltet und kommt für eine weitere Betrachtung nicht in Frage, da er zusätzlich auch keine weiteren Kontextinformationen wie Thread IDs enthält [13].

ADFA-LD Der ADFA Linux Dataset (ADFA-LD) wurde von Creech et al. [13] im Jahre 2013 erstellt und ist damit wesentlich aktueller als die zuvor genannten. Dieser wurde auf einem Linuxsystem aufgezeichnet, dessen Schwachstellen von verschiedenen *Penetration Testing* Tools ausgenutzt werden. Aufzeichnungen wurden auf dem Betriebssystem Ubuntu 11.04 durchgeführt, allerdings wurden diese nicht gut dokumentiert was ein Bearbeiten erschwert [1]. Hinzu kommt, dass lediglich Sequenzen von System Call IDs aufgezeichnet wurden und damit keine Metadaten im Datensatz enthalten sind.

NGIDS-DS Der 2017 erstellte Datensatz NGIDS [30] wurde mit Hilfe der dedizierten Security Hardware *IXIA Perfect Storm* aufgezeichnet. Er beinhaltet Thread Informationen, aber auch hier fehlen weitere Daten wie zum Beispiel System Call Argumente. Ein weiteres großes Problem liegt in der Ungenauigkeit der Zeitstempel, welche nur auf die Sekunde genau sind und es ergeben sich Schwierigkeiten in der Zuordnung der beschriebenen Event ID und den Zeitstempeln [28].

LID-DS 2019 veröffentlichten Grimmer et al. [28] das *Leipzig Intrusion Detection-Data Set* (LID-DS). Sie erkannten, dass die bisherigen Datensätze entweder veraltet oder nicht ausreichend waren um zum Beispiel Thread IDs oder System Call Argumente für ein IDS zu ver-

Szenarien	
Name	Beschreibung
Bruteforce-CWE-307	**Setup:** Simple Wordpress Web-Applikation **Schwachstelle:** Ungeeignete Einschränkung von übermäßigen Authentifizierungsversuchen
CVE-2012-2122	**Setup:** Oracle MySQL Datenbank **Schwachstelle:** Mehrfacher falscher Loginversuch führt zu erfolgreichem Login
CVE-2014-0160	**Setup:** Simple Web-Applikation **Schwachstelle:** Fehler in OpenSSL Implementierung, Heartbleed
CVE-2017-7529	**Setup:** Nginx Web-Applikation **Schwachstelle:** Datenleck durch *Integer Overflow*
CVE-2018-3760	**Setup:** Rails Web-Applikation **Schwachstelle:** Informationsleck durch *Sprockets*
CVE-2019-5418	**Setup:** Rails Web-Applikation **Schwachstelle:** Schwachstelle in Applikations-Controller
EPS_CWE-434	**Setup:** Upload Service **Schwachstelle:** Keine Upload-Beschränkungen
PHP_CWE-434	**Setup:** Web-Applikation **Schwachstelle:** Keine Upload-Beschränkungen
SQL Injection	**Setup:** Web-Applikation mit SQL-Datenbank **Schwachstelle:** SQL-Injection
ZipSlip	**Setup:** Uploading Portal für Zip Dateien **Schwachstelle:** Dateien werden ohne Überprüfung dekomprimiert

Tabelle 2.2: Kurzbeschreibung der in dem Datensatz vorkommenden Szenarien und den dazugehörigen Sicherheitslücken [28]

wenden. Er wurde auf einem modernen Betriebssystem Ubuntu 18.04 aufgenommen und besteht aus 10 verschiedenen Szenarien auf welchen jeweils ein Angriff ausgeführt wird. Jedes Szenario repräsentiert damit also eine bekannte Schwachstelle eines Systems. In Tabelle 2.2 werden die verschiedenen Schwachstellen als Common Vulnerabilities and Exposures (CVE) oder Common Weakness Enumeration (CWE) bezeichnet. CVE ist ein Industriestandard der für eine einheitliche Namenskonvention für Sicherheitslücken verwendet wird. Bei den CWEs handelt es sich um eine von der Community gepflegte und von der MITRE Corporation veröffentlichte Auflistung verschiedener Typen von Schwachstellen in Soft- und Hardware. Aufgezeichnet werden für jedes Szenario neben den Namen der System Calls auch deren Metadaten. Dazu zählen unter anderem die Parameter, Rückgabewerte, Zeitstempel sowie User-, Prozess- und Thread-IDs.

zu dt. Allgemeine Schwachstellen und Gefährdungen

zu dt. Aufzählung gemeinsamer Schwachstellen

Jedes Szenario beinhaltet insgesamt ca. 1000 30-60 Sekunden lange Aufzeichnungen. Die ersten 200 Aufzeichnungen dienen als Trainingsdaten, die darauffolgenden 50 als Validierungsdaten und die restlichen als Testdaten. Dabei sind mindestens in 100 Aufzeichnungen der Testdaten Angriffe enthalten, für welche der Angriffszeitpunkt gegeben ist. Jede dieser Aufzeichnungen ist in einer Datei gespeichert, welche in einer *runs.csv* beschrieben und zusammengefasst werden.

In Tabelle 2.3 wird ein Ausschnitt aus der beschreibenen *runs.csv* dargestellt und in Tabelle 2.4 eine Beispieldatei einer tatsächliche Aufzeichnung von System Calls.

runs.csv		
Eintrag	Beispieldatei 1	Beispieldatei 2
image_name	file_1	file_2
scenario_name	scenario_1	scenario_1
is_executing_exploit	False	True
warmup_time	10	10
recording_time	53	40
exploit_start_time	-1	15

Tabelle 2.3: Ausschnitt aus der runs.csv des LID-DS, welche unter anderem die Labels für Angriffsfiles und Normalfiles enthalten. [28]

System Call			
Eintrag	System Call 1		System Call x
evtent number	26		4012
evtent time	11 : 09 : 47.592865922		10 : 18 : 20.1231231231
user uid	101		33
process name	nginx	...	apache2
thread id	21822		1425
evtent dir	<		>
evtent type	sendfile		writev
evtent args	$res = 612, offset = 1225$		$fd = 12(< 4t > 172.131.12.1 : 123 \rightarrow 172.13.231.2 : 123), size = 2392$

Tabelle 2.4: Ausschnitt aus den eigentlichen Aufzeichnungen von System Calls aus dem LID-DS [28]

PROBLEM DES LID-DS In den Testdaten mit enthaltenen Angriffen wird ein Angriffszeitpunkt angegeben. Jedoch gibt es bei einer Datei mit Angriff im Gegensatz zu den normalen Dateien vier statt zwei mögliche Zuordnungen. In den normalen sind das zum einen True Negative (TN), falls kein Alarm vorliegt, da in den normalen Dateien keine Angriffe stattfinden. Oder falls fälschlicherweise ein Angriff erkannt wird, die Zuordnung als False Positive (FP). Die Zuordnungen der Dateien mit Angriff werden in Abbildung 2.3 dargestellt.

Befindet sich der Anomaliescore vor dem Angriffszeitpunkt $t_{Angriff}$ unter dem Schwellwert, also im Quadrant 3 der Abbildung, kann von einem TN ausgegangen werden. Also es wurde korrekterweise kein Alarm vorhergesagt. Befindet sich der Anomaliescore vor dem Angriffszeitpunkt $t_{Angriff}$ über dem Schwellwert, also im Quadrant 2, liegt ein FP vor. Es wurde ein Alarm gemeldet an einer Stelle an dem (noch) kein Angriff stattfand. Nach dem angegebenen Angriffs-

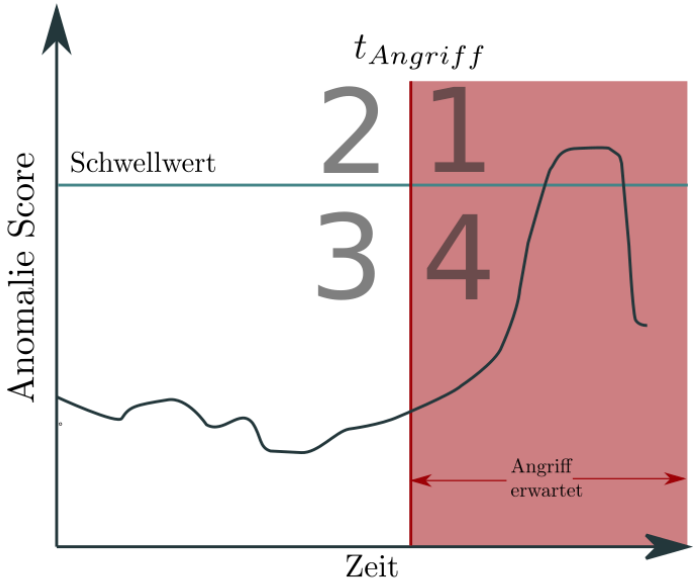

Abbildung 2.3: Illustration des Quadrantenproblems (Abwandlung von [25]). Es beschreibt die Problematik, dass Alle System Calls nach $t_{Angriff}$ als bösartig eingestuft werden müssen, obwohl auch Normalverhalten stattfindet.

zeitpunkt $t_{Angriff}$ wird es allerdings schwieriger. Denn liegt der Anomaliescore nach dem Angriffszeitpunkt über dem Schwellwert, also im Quadranten 1, wird von einem korrektem Alarm ausgegangen. Jedoch kann der Angriff zu diesem Zeitpunkt schon vorbei sein. Also kann ein Alarm nach $t_{Angriff}$ theoretisch auch ein FP sein. Genauso könnte umgekehrt ein Anomaliescore unter dem Schwellwert, also im Quadranten 4 korrekt sein obwohl er als False Negative (FN) gewertet werden müsste. Diese Problematik wird auch von Grimmer et al. [25] aufgegriffen und muss in der Auswertung beachtet werden. In Abschnitt 4.5 wird eine Lösungsmöglichkeit für die Problemstellung betrachtet.

Nachdem nun verschiedene allgemeine Ansätze zur Erkennung von schädlichem Verhalten eines Systems und mögliche Datensätze zur Auswertung untersucht wurden, soll im Folgenden die Frage geklärt werden, wie mit Hilfe von künstlichen neuronalen Netzen Muster in einem Datensatz erkannt werden können.

2.3 KÜNSTLICHE NEURONALE NETZE

Das Übertragen von bestehenden und in der Natur vorkommenden Strukturen und Prozessen kommt in verschiedenen Forschungsbereichen zum Einsatz. Auch zum Beispiel in direkter Umsetzung für Oberflächenstrukturen welche häufig in der Raumfahrt eingesetzt werden [82], oder auch Abstrakter in der Umsetzung von Verhal-

tensweisen von Ameisen [53]. Ein in den letzten Jahren immer weiter verbreiteter Ansatz in der elektronischen Datenverarbeitung ist die Verwendung von künstlichen neuronalen Netzen. Diese haben Neuronen und neuronale Netze als biologisches Vorbild. Dabei ist allerdings die abstrakte Modellierung der Informationsverarbeitung im Vordergrund und nicht das Nachbilden der biologischen neuronalen Netze. Man verspricht sich mit dem Einsatz von künstlichen neuronalen Netzen, welche eine Varietät an verschiedenen Architekturen beinhalten, diverse Optimierungsprobleme zu lösen. Zu aktuellen Beispielen zählen dabei auch Vorhersagen die im Zusammenhang mit der Verbreitung des COVID-19 Virus stehen [52, 66, 80].

Algorithmen in welchen neuronale Netze zum Einsatz kommen bestehen generell aus zwei Phasen. In der ersten Phase, der Trainingsphase, werden vordefinierte Trainingsdaten in das Netz gegeben. Dieses versucht Merkmale in den Daten zu ermitteln, mit welchen dann in der Testphase Voraussagen gemacht werden können. Dabei haben sich spezialisierte Herangehensweisen entwickelt, wie zum Beispiel die Convolutional Neural Net (CNN) für die Bilderkennung. Für sequentielle Daten wie Audio, Text und Video, bei welchen eine zeitliche Komponente entscheidend ist, eignen sich vor allem die RNN. Da in dieser Arbeit sequentielle Daten betrachtet werden, die sequentielle Abfolge von System Calls, soll im Folgenden nur auf die RNN und eine besondere Abwandlungen dieser eingegangen werden.

2.3.1 *Rekurrente neuronale Netzwerke*

Der entscheidende Unterschied von RNNs zu herkömmlichen neuronalen Netzen ist, dass der Ausgang eines Knotens auf einer *Layer* mit einer vorherigen oder derselben Layer verbunden ist. Ist dies der Fall spricht man von einem *Feedback* oder *Recurrent Neural Network*, sonst von einem *Feedforward Neural Network*. Mit Knoten, die eine extra Verbindung zu sich selbst haben, können frühere Eingaben Einfluss auf die Behandlung der nächsten Eingabe haben. Der einzelne Knoten merkt sich seine Ausgabe, welche im nächsten Zeitschritt als weiteres Eingabesignal dient. Dadurch wird es ermöglicht auch zeitlich abhängige Sequenzen zu erlernen, da die Signalverarbeitung der RNNs auch vorherige Geschehnisse mit einbezieht. Im Gegensatz dazu steht zum Beispiel die Bilderkennung, bei welchem das vorherige Bild keinen Einfluss auf die Einschätzung des aktuellen Bildes hat. Dieser Zusammenhang lässt sich in der folgenden Gleichung darstellen.

zu dt. Ebene

$$\begin{aligned} h_t &= \sigma\left(W_h h_{t-1} + W_x x_t + b\right) \\ y_t &= h_t \end{aligned} \tag{2.1}$$

Dabei beschreiben x_t, h_t und y_t den Eingang des Neurons, die rekurrente Information und den Ausgang des RNN. W_h und W_x beschreiben die Gewichte und b den Bias. Ein einzelner Knoten wird ohne die

Gewichte in Abbildung 2.4 dargestellt, sowie in Abbildung 2.5 in einer alternativen Darstellung, wie sich dieser Knoten A über t Zeitpunkte verhält. Dabei werden für die Übersichtlichkeit Gewichte sowie der Bias nicht dargestellt. Doch auch die RNNs haben ein Problem bei

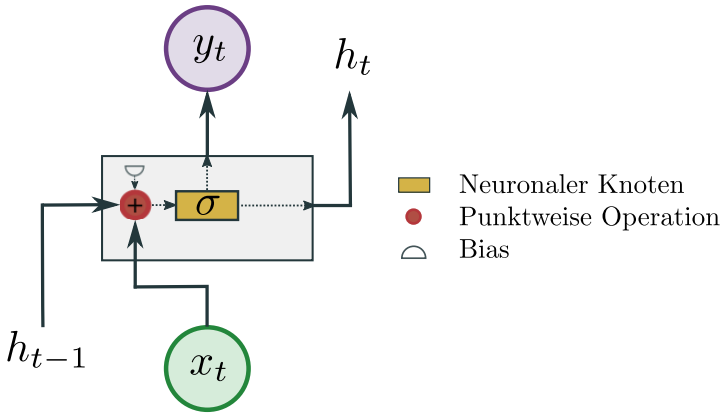

Abbildung 2.4: Darstellung einer RNN Zelle

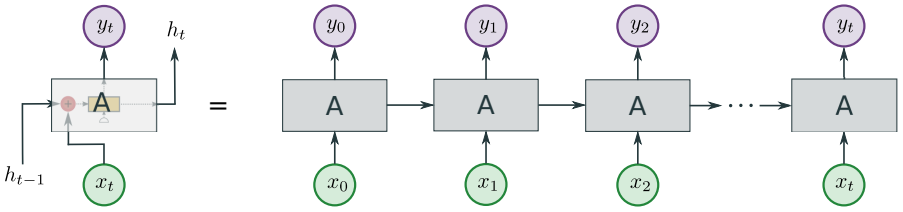

Abbildung 2.5: Darstellung einer „ausgerollten“ und vereinfachten RNN Zelle

Merkmalen, die sich über einen längeren Zeitraum strecken. Denn dabei kommt es häufig vor, dass durch die *Backpropagation* die berechneten Gradienten entweder verschwindend klein, oder sehr groß werden. Gerade bei Abhängigkeiten über einen größeren zeitlichen Abstand tendieren die Fehlersignale, die durch die Backpropagation durch das Netz gegeben werden, zu geringe Gewichtsänderungen auszulösen. Traditionelle Aktivierungsfunktionen wie die hyperbolische Tangensfunktion haben Gradienten im Bereich $(-1, 1)$ oder $[0, 1)$ und Backpropagation berechnet Gradienten durch die Kettenregel. Dies hat den Effekt, dass n dieser kleinen Zahlen multipliziert werden, um die Gradienten der „vorderen“ Schichten in einem n-Schichten-Netzwerk zu berechnen, was bedeutet, dass der Gradient (Fehlersignal) exponentiell mit n abnimmt und die vorderen Schichten sehr langsam trainieren. Die von Sepp Hochreiter erstmals erwähnte LSTM Zellen ermöglichen durch verbesserte Fehlerkorrektur stabilere Lernergebnisse sowie auch das Lernen von Mustern mit noch größeren zeitlichen Abständen. [32]

Diese Sonderform der RNNs, die auch in dieser Arbeit verwendet werden, sollen deshalb genauer untersucht werden.

2.3.2 *Long-Short-Term-Memory neuronale Netzwerke (LSTMs)*

Hauptziel der LSTMs ist es, das Lernen der zeitlich abhängigen Muster zu verbessern. Entscheidend dafür ist die Einschätzung welche zuvor gesehenen Informationen für die aktuelle Eingabe relevant ist.

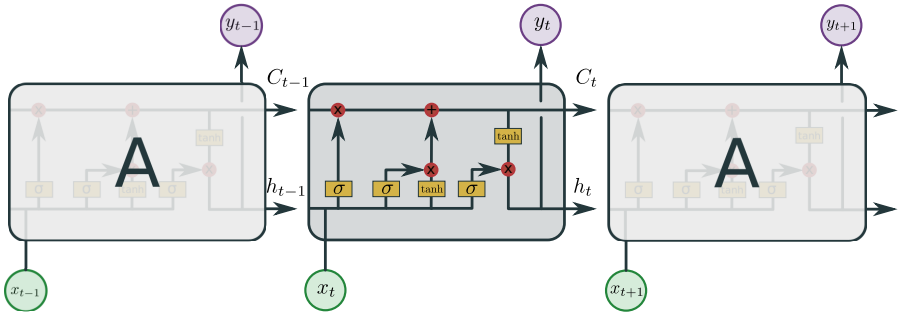

Abbildung 2.6: Schematische Darstellung eine Memory Cell A in einem LSTM NN, mit Input, Output und Forget Gate (Abwandlung von [61]).

Und um zu erlernen welche früheren Ausgaben für die Ermittlung nächster Datenpunkte entscheidend sind, wird an jedem Knoten eine mit A in der Abbildung 2.6 gekennzeichnete *Memory Cell* angebracht. Sie ist mit sich selbst verbunden, kennt also die vorherigen Ausgaben und gibt den Zellstatus an. Mit Hilfe dieser Information soll eine Abhängigkeit auch über einen längeren Zeitraum gefunden werden. Der Zellstatus C_{t-1} zum Zeitpunkt $t-1$ hat im nächsten Zeitschritt t einen Einfluss auf den Zellstatus C_t und somit auch auf die Ausgabe y_t. Die Weitergabe des Status wird in Abbildung 2.7 dargestellt.

zu dt. Gedächtniszelle

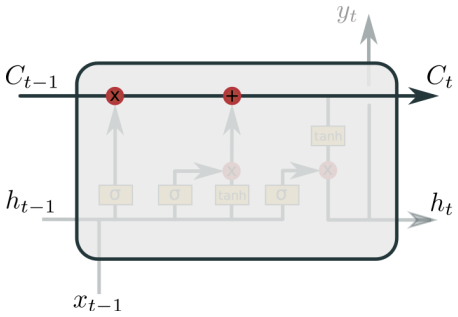

Abbildung 2.7: Weitergabe des Zellstatus innerhalb eines Knotens (Abwandlung von [61]).

Einfluss auf den Zellstatus haben zwei verschiedene *Gates* . Im ersten Schritt wird entschieden, welche Information aus dem vorherigen Zeitschritt keinen Einfluss mehr auf den Zellstatus haben sollen. Dies wird mit dem *Forget Gate* umgesetzt und ist in Abbildung 2.8 zu sehen und kann analog zur RNN Zelle folgendermaßen hergeleitet werden.

zu dt. Gatter/Tore

$$f_t = \sigma\left(W_{fh} h_{t-1} + W_{fx} x_t + b_f\right) \tag{2.2}$$

W_{fh} und W_{fx} beschreiben die Gewichte b_f den Bias des *Forget Gate*. Es wird also die vorherige Eingabe h_{t-1} sowie die aktuelle Eingabe x_t gewichtet und mit Bias an die Aktivierungsfunktion σ übergeben. Damit sollen Informationen aus dem Speicher, die keinen Einfluss mehr haben sollen, entfernt werden. In dem Sprachbeispiel ist vorstellbar, dass das Genus (*grammatikalisches Geschlecht*) gespeichert wird, um so eine grammatikalisch korrekte Vorhersage zu machen. Kommt nun allerdings ein neues Pronomen in der Eingabe x_t, sollte das bisher gespeicherte Genus keinen Einfluss mehr haben.

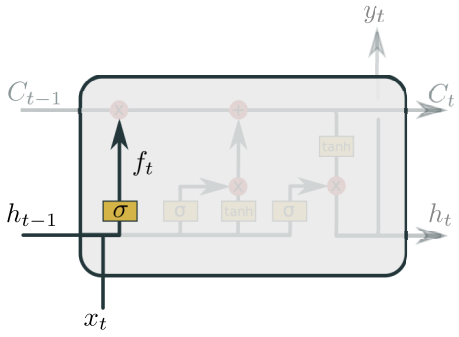

Abbildung 2.8: Einfluss des Forget Gates auf den Zellstatus (Abwandlung von [61]).

Das *Input Gate* soll im nächsten Schritt angeben, welche neuen Informationen in den Zellstatus C_t aufgenommen werden. Dies erfolgt in zwei Schritten, zunächst wird mit i_t ermittelt, welche Information geupdated werden soll.

$$\begin{aligned} i_t &= \sigma\left(W_{ih}h_{t-1} + W_{ix}x_t + b_i\right), \\ \tilde{C}_t &= tanh\left(W_{\tilde{C}h}h_{t-1} + W_{\tilde{C}x}x_t + b_{\tilde{C}}\right), \end{aligned} \tag{2.3}$$

Im Vektor $\tilde{C}$ sind mögliche Kandidaten enthalten (wie z.B. das Genus), welcher den zuvor vergessenen Wert ersetzen soll (vgl. Abbildung 2.9). Der gesamte Zellstatus C_t wird dann zusammen mit Werten aus dem *Forget Get* verrechnet.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t, \tag{2.4}$$

Wie der Zellstatus C_t nun die Ausgabe beeinflusst, wird über das *Output Gate* geregelt (siehe Abbildung 2.10).

$$\begin{aligned} o_t &= \sigma\left(W_{oh}h_{t-1} + W_{ox}x_t + b_o\right), \\ h_t &= o_t tanh\left(c_t\right) \end{aligned} \tag{2.5}$$

Dies soll in unserem Sprachbeispiel entscheiden, ob die Information des Genus für die Vorhersage des nächsten Wortes eine Rolle spielt. [22, 61]

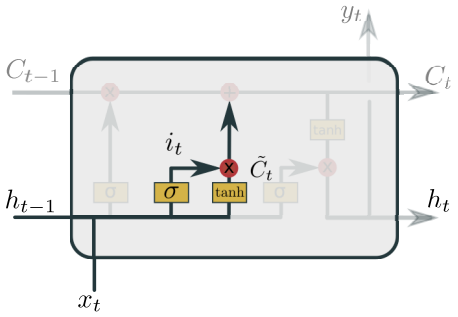

Abbildung 2.9: Einfluss des Input Gates auf den Zellstatus (Abwandlung von [61]).

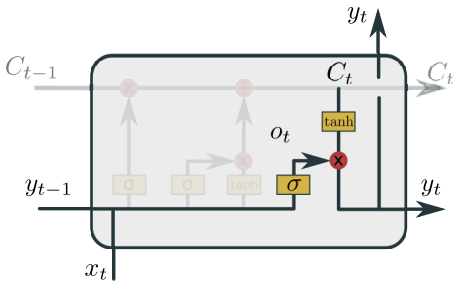

Abbildung 2.10: Das Output Gate regelt den Einfluss des Zellstatus auf die Ausgabe des Neurons (Abwandlung von [61]).

Die verschiedenen Gates können so als ein weiteres kleines NN in jedem Knoten der LSTM Netze betrachtet werden, welche einen zeitlichen Zusammenhang besser erkennen sollen. Gesamt lässt sich eine LSTM Zelle mit den folgenden Formeln beschreiben:

$$
\begin{aligned}
f_t &= \sigma\left(W_{fh}h_{t-1} + W_{fx}x_t + b_f\right) \\
i_t &= \sigma\left(W_{ih}h_{t-1} + W_{ix}x_t + b_i\right), \\
\tilde{C}_t &= tanh\left(W_{\tilde{C}h}h_{t-1} + W_{\tilde{C}x}x_t + b_{\tilde{C}}\right), \\
C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t, \\
o_t &= \sigma\left(W_{oh}h_{t-1} + W_{ox}x_t + b_o\right), \\
h_t &= o_t tanh\left(c_t\right) \\
y_t &= h_t
\end{aligned}
\tag{2.6}
$$

3 VERWANDTE ARBEITEN

Die Forschungsfrage mit der sich diese Arbeit beschäftigt, kann wie in Abschnitt 1.2 beschrieben, weiter unterteilt werden. Zum einen soll untersucht werden inwiefern LSTMs in der Domäne der anomaliebasierten HIDS Vorteile bringen können. Und zum anderen wie die Sequenzen von System Calls angereichert werden können um die FP-Rate sowie die Detektionsrate (DR) zu verbessern. Um den Überblick über die verwandten Arbeiten zu bewahren, sind diese im Folgenden untergliedert. Zunächst sollen Grundlagen der Anomalieerkennung und erste Ansätze der Anomalieerkennung mit System Calls aufgeführt werden, auf welchen diese Arbeit indirekt fußt. Im nächsten Schritt werden dann Arbeiten betrachtet, welche sich auf die Argumente der System Calls konzentrieren. Abschließend werden Arbeiten untersucht welche sich speziell mit der Übertragung von NLP Verfahren in die HIDS auseinandersetzen. Dazu zählen auch Ansätze mit LSTMs.

3.1 GRUNDLAGEN ANOMALIEDETEKTION

Die Anfänge der Anomalieerkennug werden von Huang et al. [35] auf die Arbeit von Grubbs [29] zurückgeführt, in welcher sogenannte Ausreißer in Sample-Daten gefunden und entfernt werden sollen. Hingegen berufen sich Chandola et al. [10] bei den Anfängen der Anomalieerkennung auf eine Arbeit aus dem *Dublin Philosophical Magazine of Journal and Science* von 1887 [17]. Dort werden *discordant observations* anhand einer abweichenden gesetzmäßigen Frequenz isoliert. Speziell im Kontext der Angriffserkennung wird wie in Unterabschnitt 2.1.2 beschrieben die Anomalieerkennung in NIDS und HIDS eingeteilt. Da es wie in der Anomalieerkennung in diesen Bereichen eine große Anzahl an Arbeiten und auch Übersichtsarbeiten gibt [10, 37, 65], soll im Folgenden nur auf bestehende Arbeiten im Bereich der HIDS eingegangen werden, welche speziell System Calls verwenden.

zu dt. nicht stimmige Beobachtung

3.2 ANOMALIEBASIERTE HOST-BASED INTRUSION DETECTION SYSTEMS (HIDSS) MIT SYSTEM CALLS

Zunächst soll im Folgenden HIDS betrachtet werden die explizit nur die Sequenzen der System Call Namen betrachten und alle weiteren Informationen nicht beachten.

OHNE BETRACHTUNG DER SYSTEM CALL ARGUMENTE Bereits 1996 stellten Stephanie Forrest et al. [20] die erste Arbeit vor, in welcher sie mit ihrem Time-Delay Embedding (TIDE) Algorithmus Anomalien in System Call Daten ermitteln. Dabei wird anhand einer Datenbank gültiger System Call Paare, *lookahead pairs*, ermittelt ob eine System Call Sequenz eine Anomalie darstellt.

In einer späteren Arbeit erweitern sie diesen Ansatz, indem sie die Datenbank mit zusammenhängenden Sequenzen von System Calls befüllen. Kommt eine Sequenz nicht in der Datenbank vor, wird diese als Anomalie eingestuft. So wird TIDE zu STIDE. [34]

Ein anderen Ansatz wählten 1997 Lee et al. [44], wobei sie sich auch auf die Pionierarbeit von [20] berufen. Sie versuchen mit dem ML-Programm RIPPER Regeln aus den System Call Daten abzuleiten. Im Gegensatz zu [20] befinden sich bei diesem Ansatz normale sowie anomale Sequenzen in den Trainingsdaten.

1999 untersuchten Warrender et al. [81] wie verschiedene Algorithmen auf System Call Daten abschneiden. Dazu gehören die bereits erwähnten Verfahren TIDE, STIDE, RIPPER, sowie ein Hidden Markov Modell. Dabei schienen alle Verfahren erfolgreich wobei das Hidden Markov Modell als sehr rechenintensiv hervorgehoben wird.

Auch aktuellere Arbeiten befassen sich mit der Anomalieerkennung mit System Calls und verwenden dabei speziell nur die Namen von System Calls, ohne die Einbindung der System Call Argumente.

2005 betrachten Kang et al. [38] nicht die Sequenzen sondern die Frequenzen der auftretenden System Calls. Dabei zählen sie das Vorkommen von System Calls in einem bestimmten Zeitfenster und verwenden diese sogenannten *bag of system calls* für ihre Beschreibung des Normalverhaltens. Ähnlich zu [20] und [34] bauen sie mit diesen Bündelungen eine Datenbank für das Normalverhalten auf.

zu dt. Bündel von Systemaufrufen

2013 stellen Murtaza et al. [58] System Calls als Zustände von Kernelmodulen dar, indem sie System Calls einem bestimmten Modul zuschreiben. Dazu gehört zum Beispiel das Modul *File System* mit den System Calls *read write* etc., oder auch *Architecture*, *Memory Management*. Die Wahrscheinlichkeiten für das Auftreten von Zustandssequenzen wird dann zur Identifizierung von Anomalien genutzt.

2018 interpretieren Grimmer et al. [27] System Call Sequenzen als einen sogenannten *System Call Graph*. Dabei werden wie in [81] N-Gramme verwendet. Die N-Gramme stellen einen Knoten dar und der Übergang eines N-Gramms zu einem weiteren wird mit einer gerichteten Kante dargestellt. Zusammen mit den Häufigkeiten des Auftretens eines Überganges und dem Ausgangsgrad eines Knotens ergeben sich die Übergangswahrscheinlichkeiten für alle Knoten. Der Anomaliescore eines Fensters von N-Grammen aus den Testdaten wird anhand der Übergangswahrscheinlichkeiten aus dem im Training aufgebauten Graphen abgelesen.

Doch es gibt auch mehrere Arbeiten die Schwachstellen der Angriffserkennung mittels Sequenzen von System Calls aufzeigen. In der Arbeit von Wagner et al. [79] werden verschieden Methoden untersucht mit welchen Angriffe nicht durch das IDS von Somayaji et al. [70] erkannt wurden. Bei ihren theoretischen Ansätzen berufen sie sich unter anderem auf das Abändern von System Call Argumenten, ohne dabei auf die Sequenz der System Calls einfluss zu nehmen. Und

beruht auf STIDE [34]

Tan et al. [76] untersuchen speziell die von Forrest et al. [20] ins Spiel gebrachte Fensterlänge von 6 für den STIDE Algorithmus. Dabei umgehen sie die Angriffserkennung in dem sie die Angriffssequenzen auseinanderziehen und mit Normalsequenzen auffüllen.

MIT BETRACHTUNG DER SYSTEM CALL ARGUMENTE Dies zeigt, dass es sinnvoll sein kann auch noch weitere Informationen aus den System Calls einzubinden. Seien es Metadaten wie die Thread Information der System Calls, oder auch die eigentlichen Argumente der Aufrufe. Inwiefern diese zusätzlichen Informationen bereits in der Literatur verwendet wurde soll nun behandelt werden.

2003 wählen Kruegel et al. [42] einen zu den bisherigen Arbeiten konträren Weg. Sie missachten die Sequenz und betrachten nur die Rückgabewerte und Argumente der System Calls. Sie erstellen in der Trainingsphase für jeden System Call verschiedene Modelle, welche in der Testphase die Wahrscheinlichkeit eines anomalen Verhaltens bestimmen. Speziell nutzen sie aber auch zuvor gesammelte Informationen über mögliche Angriffe für die Erstellung der Modelle. Spannend dabei sind vor allem die vorgestellten Modelle, welche nun im Folgenden genauer beschrieben werden.

String Length: Die Annahme dieses Features besteht darin, dass sich bei einem Angriff die String-Länge der Argumente signifikant ändert. Dafür wird in der Trainingsphase versucht die Verteilung der String-Längen zu approximieren. In der Testphase wird dann die Wahrscheinlichkeit dafür, dass die aktuelle String-Länge aus der Verteilung stammt mit der *Tschebyscheffschen Ungleichheit* berechnet.

String Character Distribution: Hier wird angenommen, dass es unter legitimen System Calls Ähnlichkeiten unter den Frequenzen der auftretenden Zeichen eines Strings gibt. In der Trainingsphase wird für jedes gesehene Argument die Zeichenverteilung hinterlegt. Ähnlich zu der String-Länge zuvor wird in der Testphase nun überprüft mit welcher Wahrscheinlichkeit die aktuelle Zeichenverteilung aus der gespeicherten Verteilung gezogen werden kann.

Structural Inference: Da Angriffe laut Kruegel et al. [42] aber auch besonders lange oder auffällige Verteilungen von Argumenten umgehen können, wird versucht auch die Struktur der Argumente zu untersuchen. Um diese Struktur zu erlernen, vereinfachen sie die Argumente zunächst wie auch von Maggi et al. [59] beschrieben. Beispielhaft wird aus dem Pfad `/usr/lib/libc.so` $\longrightarrow$ `/a/a/a.a`. Für

das Erlernen verwenden sie ein Markov Modell. In der Testphase wird dann untersucht ob die aktuelle Struktur des System Call Arguments durch das Markov Modell erstellt werden kann oder nicht.

Token Finder: Es soll ermittelt werden ob die Werte eines bestimmten System Call Arguments aus einer endlichen Menge von Werten stammt. Laut Kruegel et al. [42] werden häufig System Calls mit zum Beispiel den selben Flags aufgerufen, allerdings gibt es auch Argumente bei welchen so eine klare Aufteilung unbrauchbar ist. Mit einer statistischen Analyse wird deswegen in der Trainingsphase ermittelt ob ein Argument einer zufälligen Verteilung folgt. Falls dies nicht der Fall ist, werden diese Werte in die Normaldatenbank aufgenommen. Für diese Werte wird in der Testphase überprüft ob der aktuelle Wert in der Normaldatenbank vorkommt. Trifft dies zu, wird eine 1 zurückgegeben und falls nicht eine 0. Folgt das Argument einer Zufallsverteilung, wird immer eine 1 zurückgegeben. Für die erwähnten Modelle der System Call Argumente wurde von Kruegel et al. ein LibAnomlay Framework entworfen um die Nutzbarkeit auch für andere Arbeiten zu ermöglichen. [42] Leider scheint das Framework aktuell nicht mehr nutzbar zu sein, da es online nicht mehr abrufbar ist.

Auch in [59] kommen diese Modelle der Argumente zum Einsatz.

Ebenso werden sie auch von Maggi et al. [49] verwendet und mit Hilfe des LibAnomaly Framework bauten sie eine Alternative Verarbeitung dieser Argument-Modelle. Doch bei allen zuvor erwähnten Arbeiten mit Verwendung von System Call Argumenten wird die Sequenz der System Calls nicht mehr betrachtet.

Koucham et al. wollen nun im Gegensatz dazu sowohl die Sequenz als auch die Argumente beachten. Dafür clustern sie die Argumente jedes System Calls für jeden einzelnen Prozess. So wird der System Call in der Testphase dem nächsten Cluster zugeordnet und dieses dient dann als Eingabe. Für das Clustern fließen in dieser Arbeit verschiedene Argumente und Kontextinformationen ein. Zu den Argumenten gehören zum Beispiel Pfadnamen, Benutzerkennung, die Menge der möglichen Werte, wozu unter anderem Flags zählen. Kontextinformationen beinhalten Rückgabewerte, Dateirechte oder auch Dateimodi. [41]

Luckett et al. versuchen in ihrer Arbeit sogenannte *Rootkits* in System Call Daten zu ermitteln. *Rootkits* sind eine Klasse von Malware, welche die Fähigkeit haben Root-Zugriff zu erhalten und dabei unentdeckt zu bleiben [7]. Sie verwenden dafür nur das Timing von System Calls um mit neuronalen Netzen die Daten in normal oder anomal zu klassifizieren. Dabei beschreiben sie aber nicht genauer wie sie das Timing der System Calls verwenden. [47]

Einen anderen Ansatz wählen Grimmer et al. indem sie die vorhandenen Informationen nicht direkt kodieren, sondern die Thread ID verwenden um die Streamverarbeitung anzupassen. Der Ausgangs-

punkt ihrer Überlegungen liegt darin, dass moderne Prozesse Multi-Threaded sind. Das heißt die Sequenz der System Calls beschreibt die Aktionen mehrere Threads gleichzeitig. Um diese Vermischung der Threads in den N-Grammen zu verhindern, bilden sie sie nur aus System Calls desselben Threads. So konnten sie in den meisten Fällen eine Verringerung der Fehlalarme und eine Erhöhung der Erkennungsrate erreichen.[25]

3.3 Natural Language Processing (NLP) in der Anomaliedetektion und HIDSs

In den letzten Jahren wurden auch diverse Ansätze vorgestellt, in welchen ein HIDS mit Hilfe von Verfahren aus der NLP designt wurden. Einen Überblick der verschiedenen Transfers wurde in der Arbeit von Sworna et al. [73] bereitgestellt. Besonders hervorzuheben ist dabei unter anderem die Studie von Wunderlich et al. [83] in welcher sie Sequenzen von System Calls als Text betrachten und verschiedene aus der NLP bekannte Verfahren in der Domäne der HIDS auswerten. Dazu gehören One-Hot-Encoding (OHE), Word2Vec (W2V) sowie GloVe und fastText. Zusätzlich untersuchen sie wie sich die Darstellung der System Calls als Kernelmodule, ähnlich zu [58], als Sequenz der Namen oder die Kombination beider auf die Ergebnisqualität auswirkt. In der Auswertung bevorzugen sie das OHE, wobei in dieser Arbeit nur geringfügig auf die Vorteile der Dimensionsreduktion durch das W2V-Verfahren eingegangen wird.

Ein in der NLP verbreitetes Konzept sind die LSTM [23, 84]. Diese Netzwerke finden ihre Anwendung auch in der HIDS, dazu gehören zum Beispiel [6, 11, 40, 60, 64, 72]. Doch die meisten Arbeiten verwenden entweder sehr alte oder anderweitig kritisch zu betrachtende Datensätze, wie in Unterabschnitt 2.2.3 beschrieben. Leider haben sie damit eine geringere Aussagekraft für aktuelle Systeme, als es mit aktuelleren Datensätzen der Fall wäre. So zum Beispiel Kim et al. [40], sie beschreiben wie sie Sprachmodelle der System Calls mithilfe von LSTMs erstellen. Mit diesen Modellen wollen sie dann für eine Sequenz von System Calls eine Vorhersage treffen und die Ergebnisse der verschiedenen Modelle kombiniert ergeben einen Anomaliescore. Neben dem verwendeten Datensatz unterscheidet sich die Arbeit von Kim et al. von dieser Arbeit, indem sie mehrere Sprachmodelle gleichzeitig aufzubauen und keine zusätzlichen Informationen der System Calls verwenden.

Park et al. hingegen verwenden denselben Datensatz wie diese Arbeit, greifen bei der Auswertung aber auf Metriken zurück, welche in diesem Kontext nur wenig Aussagekraft besitzen, wie in Abschnitt 4.5 beschrieben wird. Zusätzlich ist die weitere Bewertung der Arbeiten erschwert, da sie nur auf koreanisch vorliegen. [63]

2017 überwachen Dymshits et al. mit ihrer LSTM-Herangehensweise mehrere Hosts gleichzeitig. Sie erstellen aber aufgrund der großen Datenmengen normalisierte *Bag of System Call*-Vektoren. Zusätzlich legen sie ein fest definiertes Zeitfenster fest für welche diese Bündelungen vorgenommen werden. [16]

Es scheint also ein mittlerweile in der HIDS weit verbreitet zu sein, einen Transfer von NLP-Verfahren vorzunehmen. Dies gilt für Vorverarbeitungsschritte wie das W2V-Verfahren sowie auch verarbeitende Algorithmen wie den LSTMs. Diese Arbeit will speziell bekannte Ansätze sinnvoll verbinden. Zusätzlich soll durch das Entwickeln neuer Verfahren die Argumente der System Calls zu verwenden einen Fortschritt in FP-Rate und DR-Rate erzielen.

4

REALISIERUNG

Bei der Realisierung eines anomaliebasierten IDS müssen wie in Kapitel 2 beschrieben zwei Phasen durchlaufen werden. Die Trainingsphase und die Testphase. Diesen beiden Phasen muss ein präparierter Datensatz und die Definition der exakten Eingaben und Ausgaben des LSTMs zu Grunde liegen. Welche Datensätze überhaupt in Frage kommen wurde bereits in Unterabschnitt 2.2.3 untersucht. In Abschnitt 4.1 werden verschiedene Tools und Ressourcen betrachtet, die für die Vorverarbeitung des Datensatzes und weitere Implementierungen nötig sind. Wie dieser weiterverarbeitet, für das LSTM vorbereitet wird, und welche Informationen neben dem Namen des System Calls verwendet werden, kann soll in Abschnitt 4.2 betrachtet werden. Der eigentliche Algorithmus, also das Finden von Anomalien in den Daten wird in Abschnitt 4.4 beschrieben. Dabei wird erst eine allgemeine Beschreibung des Ablaufes gegeben und anschließend detailliert auf Training und Anomalieerkennung eingegangen. In Unterabschnitt 4.4.4 werden die festgelegten Parameter und die Parameter welche in den Experimenten untersucht werden vorgestellt. Dieses Kapitel wird dann in Abschnitt 4.5 mit einer Untersuchung der Metriken abgeschlossen, die für die Auswertung der Experimente benötigt werden.

4.1 VERWENDETE BIBLIOTHEKEN UND RESSOURCEN

Für die Implementierung der folgenden beschriebenen Verfahren kommen verschiedene Bibliotheken zum Einsatz. Die Gesamte Implementierung wurde in Python 3.7 umgesetzt. Um das W2V-Verfahren einzubinden wird die W2V Implementierung der GENSIM Bibliothek [85] genutzt. Für die Implementierung des LSTMs wird die PyTorch Bibliothek [62] angewandt. Berechnungen für diese Arbeit wurden (z.T.) mit Ressourcen des Universitätsrechenzentrums Leipzig durchgeführt.

4.2 VORVERARBEITUNG

Im kommenden Unterabschnitt 4.2.1 soll untersucht werden, welche Darstellungsformen für System Calls im Rahmen dieser Arbeit interessant und sinnvoll sind. Dabei werden die System Call Daten praxisnah als Datenstream betrachtet. Weshalb in Unterabschnitt 4.2.3 die Frage geprüft, wie dieser für das LSTM dargestellt wird. Ein weiterer wichtiger Teil der Arbeit besteht darin, zu klären welche Metadaten neben dem Namen des System Calls noch verwendet werden können um die Erkennungsrate der Angriffe zu erhöhen, bzw. die Fehlerrate zu

verringern. Die Frage welche Informationen dafür verwendet werden können und wie diese dargestellt werden, wird in Unterabschnitt 4.2.2 behandelt.

4.2.1 *Darstellung eines System Calls*

Neuronale Netze benötigen numerische Werte, weswegen eine Umwandlung der Namen der System Calls stattfinden muss, sodass die Sequenz dem neuronalen Netz als Eingabe dienen kann. Eine einfache Kodierung dieser Strings bestünde darin, die System Calls in Integer Werte umzuwandeln. Allerdings entstehen dabei künstliche Zusammenhänge und Ordnungen, welche für den Lernvorgang unvorteilhaft sein können [5]. Doch es bieten sich verschiedene Alternativen, um diese künstlichen Zusammenhänge zu verhindern. Zwei mögliche Verfahren werden im folgenden behandelt.

ONE-HOT-ENCODING Für die Darstellung eines System Calls mit dem OHE, muss zunächst die Anzahl k der unterschiedlichen System Calls ermittelt werden. Der System Call sc_i aus der Menge der möglichen System Calls $SC = \{sc_1, sc_2, \ldots, sc_k\}$ wird dann als Bit-Vektor v_i der Länge k kodiert. Dabei nehmen alle Stellen bis auf i den Wert 0 an und sonst den Wert 1. So wird aus dem System Call *open* aus der Menge $SC = \{open, close, read\}$ der Vektor $v_1 = (1, 0, 0)$. Und für *close* gilt dann $v_2 = (0, 1, 0)$. Da die Anzahl der möglichen System Calls, also k begrenzt ist, scheint diese Darstellung für einen System Call denkbar. Allerdings bringt sie auch zwei neue Probleme mit sich. Zum einen führt das OHE eines System Calls bei großem k zu einem langen und spärlich besetzten Vektor v_i. Gerade bei neuronalen Netzen führt das zu längeren Rechenzeiten. Und der neu gewonnen Vorteil, dass keine künstlichen Ordnungen vorhanden ist birgt den Nachteil, dass ebenso tatsächlich vorhandene Zusammenhänge verloren gehen. So besteht ein Zusammenhang zwischen *open* und *close* da zum Beispiel Dateien welche geöffnet werden in der Regel auch wieder geschlossen werden müssen Dieser Zusammenhang ist in dem OHE verloren gegangen. Optimaler Weise gilt es also eine Darstellung zu finden die kurz ist und in welcher nur gewollte Ordnungen vorhanden sind.

WORD2VEC In der NLP gilt es ähnliche Probleme durch die Darstellungen der Wörter zu lösen. Das W2V Verfahren ist ein auf feedforward neuronalen Netzen basierender Ansatz, welcher in der NLP häufig eingesetzt wird [3]. Dabei werden Wörter aus einem gegebenen Satz an möglichen Wörtern anhand eines Trainingsdatensatzes in dichte Vektoren v_{w2v} fester Länge e kodiert beziehungsweise *embedded*. Also ein System Call sc_i aus der Menge $SC = \{sc_1, sc_2, \ldots, sc_k\}$ wird umgewandelt in einen Vektor v_{w2v} mit gegebener Länge e. Ziel des von [55] eingeführten W2V Verfahrens ist es eine Dimensionsreduktion durch-

zu dt. eingebettet

zuführen, bei welcher möglichst viel Kontextinformationen erhalten bleiben. So sollen also die Vektordarstellungen von ähnlichen Wörtern ebenso ähnlich sein. Ähnlich meint in diesem Zusammenhang eine semantische Ähnlichkeit. Ziel ist es also, dass die Vektordarstellungen von zwei schreibenden System Calls näher beieinander liegen, als die Darstellung eines schreibenden System Calls und einem System Call zur Prozesskontrolle. Dies wird erreicht, indem Worte für das Erstellen der Kodierung nicht einzeln betrachtet werden, sondern immer in einem Kontext. Es wird angenommen, dass Wörter die häufig in einem ähnlichen Kontext auftreten, auch ähnlich sind. Dabei ist mit Kontext die Wörter vor und nach dem aktuellen Wort gemeint. Wie umfangreich dieser Kontext für jedes Wort sein soll, wird mit der Fenstergröße w festgelegt. In der Trainingsphase gibt es zwei verschiedene Architekturen. *Continuous Bag-Of-Words* versucht das feedforward Netzwerk darauf zu trainieren eine Vorhersage über ein Wort anhand des Kontextes zu machen. Hingegen wird unter Verwendung von *Skip-gram*, eine Vorhersage des Kontextes aufgrund des Wortes trainiert. [68] Schwierigkeiten wie das häufige Auftreten von Wörtern in der englischen Sprache wie *the*, treten in der System Call Domäne nicht auf. Auch Grimmer et al. [25] nutzen bereits die Vorteile der dicht besetzten und dimensionsreduzierten Vektoren in anomaliebasierten HIDS mit System Calls.

In [83] wird beschrieben dass das OHE laut Ergebnissen bevorzugt werden sollte, wobei auch das W2V geeignet ist. In dieser Arbeit überwiegt allerdings der zeitliche Vorteil durch die Dimensionsreduktion des W2V. Neben den eigentlichen Namen der System Call enthalten tatsächliche System Calls wie in Abschnitt 2.2 beschrieben noch wesentlich mehr Informationen. Wie zumindest Teile dieser die Embeddings der System Calls erweitern können soll im folgenden Abschnitt untersucht werden.

mehr dazu in Abschnitt 5.2

4.2.2 *Darstellung weiterer Parameter*

Wie zusätzliche Informationen der System Calls genutzt werden können ist auch Gegenstand bestehender Forschung. In Absatz 3.2 wurden die verschiedenen bestehenden Ansätze untersucht.

Im Folgenden soll es speziell darum gehen neue Darstellungsformen der System Call Argumente zu finden. Dabei soll noch einmal kurz auf die in Abschnitt 2.2 besprochenen Grundlagen zurückgegriffen werden um einen ersten Anhaltspunkt für wichtige Informationsquellen zu finden.

Um die Unterscheidung von Normalverhalten und einer Anomalie zu erleichtern, müssen Faktoren betrachtet werden, welche sensibel auf eine Veränderung des Normalverhaltens reagieren. Gleichzeitig sollen dabei aber nicht einzelne Angriffe genutzt werden um diese Faktoren zu ermitteln. Denn dabei besteht die Gefahr, Signaturen

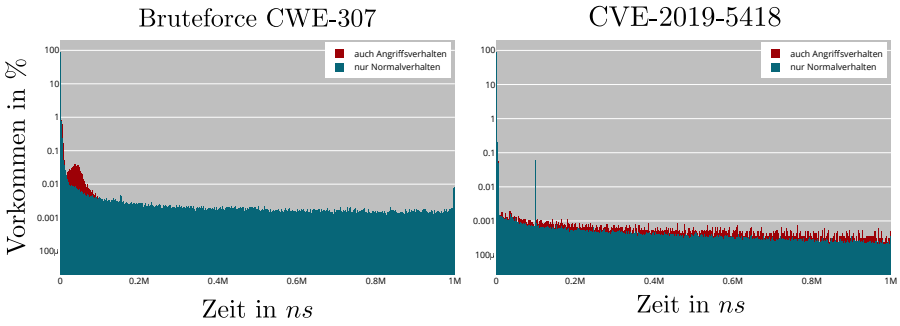

Abbildung 4.1: Dargestellt ist der zeitliche Abstand zwischen zwei System Calls aus dem selben Thread. Diese werden in ihrer Häufigkeit in Prozent an allen auftretenden Abstände in dem Plot eingetragen. Verwendet wurden dafür nur die Testdaten. Links für das Bruteforce Szenario und rechts für das CVE-2019-5418 Szenario. Blau: Nur Normalverhalten, Rot: Normalverhalten und Angriffsverhalten

einzelner Angriffe abbilden zu wollen, was wie in Unterabschnitt 2.1.1 ungewollte Nachteile hat. Ziel dieses Abschnitts ist es, die Überlegungen und Umsetzungen von zwei Verfahren zur Darstellung von zusätzlichen System Call Informationen zu präsentieren.

ZEITLICHE ABSTÄNDE VON SYSTEM CALLS Wie bereits in Absatz 3.2 beschrieben verwenden Lucket et al. [47] die Timing-Information von System Calls für das Aufspüren von Rootkits. In diesem Abschnitt wird diese Idee wieder aufgegriffen. Die Grundidee besteht darin, dass durch ein Angriffsverhalten eine zumindest kurzzeitige Veränderung im Timing der auftretenden System Calls eintritt. Dabei gelten System Calls in diesem Ansatz nur als aufeinanderfolgend, sofern sie auch aus dem selben Thread stammen. Eine genauere Erläuterung dazu erfolgt in Unterabschnitt 4.2.3. Um den Unterschied in den Normaldaten und Angriffsdaten zu untersuchen wurden in Abbildung 4.1 die Häufigkeit zeitlicher Abstände zwischen aufeinanderfolgenden System Calls aus dem Bruteforce und dem CVE-2019-5418 Szenario des LID-DS geplottet. In Blau sind die Abstände der System Calls während des Normalverhaltens eingetragen und in Rot die des Normalverhaltens mit zusätzlichen Angriffsverhalten. Im rechten Plot ist kein sichtlicher Unterschied zwischen Angriffsverhalten und dem Normalverhalten zu erkennen. Der in diesem Szenario durchgeführte Angriff scheint nicht zu einer Abweichung der Zeitabstände zwischen System Calls zu führen. Im linken Plot hingegen ist eine Abweichung im Bereich ab ca. $0.05 * 10^6 ns$ zu erkennen.

Die Umsetzung der beschriebenen Idee wird erreicht indem der zeitlichen Abstand τ zwischen zwei aufeinanderfolgenden System Calls berechnet wird. Dieser wird normalisiert als weiterer Eingabeparameter für den verarbeitenden Algorithmus verwendet. In der Trainingsphase wird zunächst nur der größte Abstand τ_{max} aus den

Trainingsdaten ermittelt. Mithilfe dieses Wertes werden dann in der Testphase alle Werte wie in Gleichung 4.1 beschrieben normalisiert.

$$\tau_{norm} = \frac{\tau}{\tau_{max}} \tag{4.1}$$

So gilt für die meisten Werte $\tau_{norm} = [0;1]$. Falls $\tau \geq \tau_{max}$ gilt, kann dieser Wert auch größer als 1 werden.

Doch treten mit der Verwendung dieser Information als Extraparameter zwei Probleme auf. Zum einen stellt sich die Frage, ob eine Verbesserung eines Szenarios mit abweichenden Abständen eine Verschlechterung in Szenarien in welchen dies nicht der Fall ist zur Folge hat. Und zum anderen stellt sich die generelle Frage ob diese Information den möglichen Ergebnisraum wesentlich vergrößert worunter die Ergebnisqualität im Gesamten leidet. Denn das Betriebssystem selbst beeinflusst die Timings der System Calls. So können Informationen eingebettet werden, welche wenig Aussagekraft über das Normalverhalten eines Prozesses haben. Ob dies das Lernen des Normalverhaltens verbessert oder verschlechtert wird in Abschnitt 5.4 untersucht.

zw. Normal- und Angriffsverhalten

RETURN WERTE Wie in Unterabschnitt 2.2.1 angemerkt, besteht ein System Call aus zwei „Phasen“. Wobei die zweite Phase den Rückgabewert des Betriebssystemskernel beinhaltet. Bei erfolgreicher Durchführung besteht dieser Rückgabewert bei verschiedenen System Calls aus einem positiven Integerwert. Ist die Durchführung jedoch nicht erfolgreich, wird ein Fehler zurückgegeben. Dieser besteht aus einem negativen Integerwert und in manchen Fällen noch einem zusätzlichem Fehlercode. So gibt der Rückgabewert eines *write* System Calls an, wieviele Bytes gelesen wurden. Auch weitere schreibende und lesende System Calls geben die gelesene oder geschriebene Bytes zurück. Für manche System Calls, die im Zusammenhang mit Sockets stehen, gilt dies ebenfalls. Der Rückgabewert enthält in diesen Beispielen also Informationen über den tatsächlichen Ablauf des spezifischen System Calls. Im Folgenden wird beschrieben wie diese Zusatzinformation der Rückgabewerte kodiert werden kann. Dabei werden nur System Calls betrachtet, welche Daten schreiben oder lesen und Daten über Sockets empfangen oder senden. Zusätzlich wird dies nur für System Calls, welche auch im Datensatz vorkommen und einen Byte Rückgabewert haben, untersucht. Leider fallen dabei System Calls wie *send* heraus, da der Rückgabewert die Anzahl der gesendeten Charaktere angibt und nicht die Anzahl an geschriebenen Bytes. Es kann also nicht davon ausgegangen werden, dass jegliche schreibende oder lesende Aktion damit abgedeckt ist. In Tabelle 4.1 werden die den Anforderungen entsprechenden System Calls mit einer Kurzbeschreibung vorgestellt.

Diese werden in ihrer Häufigkeit in Prozent an allen auftretenden Rückgabewertgrößen in dem Plot eingetragen. Für die beschriebenen

System Calls		
Name	Beschreibung	Rückgabewerte
write	Schreibt angegebene Anzahl an Bytes aus dem Buffer in die Datei, welche über den Filedeskriptor fd definiert wird.	Geschriebene Bytes oder -1 bei Fehler
pwrite	Schreibt angegebene Anzahl an Bytes aus dem Buffer in die Datei, welche über den Filedeskriptor fd definiert wird. Dabei wird der Datei-Offset nicht geändert.	Geschriebene Bytes oder -1 bei Fehler
writev	Schreibt *iovcnt* Vektor in die Datei, welche über den Filedeskriptor fd definiert wird.	Geschriebene Bytes oder -1 bei Fehler
read	Versucht angegebene Anzahl an Bytes von Filedeskriptor *fd* zu lesen und passt den Datei-Offset an.	Gelesene Bytes oder -1 bei Fehler
pread	Versucht angegebene Anzahl an Bytes von Filedeskriptor *fd* zu lesen und passt den Datei-Offset nicht an.	Gelesene Bytes oder -1 bei Fehler
readv	Liest spezifizierten *iovcnt* Vektor aus Filedeskriptor *fd*	Gelesene Bytes oder -1 Fehler
sendfile	Kopiert Daten zwischen zwei Filedeskriptoren. Kopiervorgang findet im Kernel statt.	Geschriebene Bytes oder -1 bei Fehler
sendmsg	Übermitteln einer Nachricht an einen anderen Socket.	Gesendete Bytes oder -1 bei Fehler
recv	Empfängt Daten von einem Socket.	Empfangene Bytes oder -1 bei Fehler
recvfrom, recvmsg	Empfängt Daten auf einem Socket welcher nicht zwingend verbindungsorientiert sein muss.	Empfangene Bytes oder -1 bei Fehler

Tabelle 4.1: Kurzbeschreibung ausgewählter System Calls. [54]

System Calls werden in Abbildung 4.2 die normalisierten Byte Rückgabewerte dargestellt. Beispielhaft wird dafür das $CVE-2017-7529$ Szenario genutzt. Für die System Calls read, pread und readv sind die Rückgabewertgrößen im Normalverhalten wie im Normalverhalten mit zusätzlichem Angriffsverhalten gleich und zwar ca. 615 Bytes. Bei allen anderen betrachteten Rückgabewerten ist ein Unterschied in den auftretenden Größen bei zusätzlichem Angriffsverhalten zu erkennen. So zum Beispiel bei den write, pwrite, und writev System Calls. Im Normalverhalten werden entweder 100 oder ca. 250 Bytes geschrieben, im Angriffsverhalten kommen noch weitere sehr selten auftretende Größen vor. Auch hier gilt analog zu den Zeitabständen

für die Normalisierung ρ_{norm} mit einem Rückgabewert ρ und dem Maximalwert aus den Trainingsdaten ρ_{max} aus den Trainingsdaten:

$$\rho_{norm} = \frac{\rho}{\rho_{max}} \tag{4.2}$$

Neben dem normalisierten Rückgabewert kann aber auch ein Fehlerwert Details über den Ablauf, eines System Calls liefern. Wie in Tabelle 4.1 beschrieben, wird bei diesen System Calls bei nicht erfolgreicher Durchführung eine -1 zurückgegeben. Weshalb $\rho_{norm} = -1$ gilt, falls ein System Call aus Tabelle 4.1 einen Fehlerwert liefert. Wie sich die Verwendung der normalisierten Rückgabewerte spezieller System Calls auf die Ergebnisse auswirkt wird in Abschnitt 5.4 behandelt.

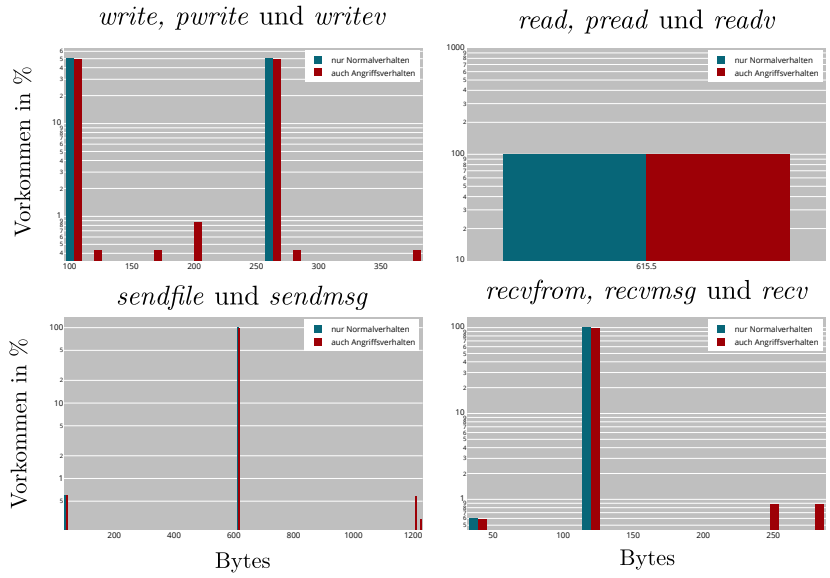

Abbildung 4.2: Histogramm der gelesenen/erhaltenen Bytes für die Testdaten des LID-DS [26] $CVE-2017-7529$ Szenarios. Dargestellt wird dabei immer der Anteil einer Rückgabewertgröße in Byte an allen Rückgabewertgrößen der besagten System Calls. Links oben für die von den System Calls *write, pwrite* und *writev* geschriebenen Bytes. Rechts oben für die von *read, pread* und *readv* gelesenen Bytes. Links unten für die von *sendfile* und *sendmsg* über Sockets gesendete Bytes. Rechts unten für die von *recvfrom, recv* und *recvmsg* über Sockets erhaltenen Bytes. Blau: Nur Normalverhalten, Rot:Normalverhalten und Angriffsverhalten

4.2.3 *Darstellung eines System Call Streams*

Nachdem die Kodierung eines System Calls und zwei weiterer Parameter neben dem Namen selbst besprochen wurde, soll in diesem

Abschnitt die Abarbeitung mehrerer System Calls für den verarbeitenden Algorithmus behandelt werden. Dabei werden die Abfolge der System Calls des Datensatzes als ein kontinuierlicher Stream betrachtet. Dies ermöglicht den Einsatz der entwickelten Vorgehensweise in der Praxis auch im Live-Betrieb, sofern die Verarbeitung entsprechend schnell stattfindet. Für viele Algorithmen wie zum Beispiel auch neuronale Netze ist es sinnvoll und teilweise unausweichlich eine feste Eingangsgröße festzulegen. Um dies für einen Stream zu ermöglichen werden in der NLP schon seit langer Zeit N-Gramme verwendet [71]. Auch in der auf System Call basierten Anomalieerkennung kommen N-Gramme zum Einsatz [25, 27, 81]. Ein N-Gramm ist eine zusammenhängende Folge von n Elementen aus einer gegebenen Eingabe. Diese werden wie in der linken Grafik in Abbildung 4.3 Beispielhaft dargestellt aufgebaut.

Dabei wird ein Buffer der Länge n erzeugt, welcher das erste n-gram liefert sofern sich n Elemente in dem Buffer befinden. Kommt ein neues Element in den Buffer fällt das älteste heraus.

Zu beachten ist dabei, dass die Abfolge der System Calls mehrere logische Abfolgen zusammenführt. Denn moderne Computer Systeme verarbeiten mehrere Threads parallel. [21] Die System Calls aller Threads werden dann je nach dem wie sie vom Betriebssystemskernel verarbeitet werden an die Sequenz angefügt.

THREAD AWARENESS Grimmer et al. [25] beschreiben in ihrer Arbeit wie sie die Thread Information der System Calls verwenden um die verwundenen Sequenzen zu entwirren. Dabei werden für jeden Thread ein eigener Buffer erzeugt. Die Ausgabe dieser Buffer wird in der rechten Grafik in Abbildung 4.3 veranschaulicht. Also nur System Calls aus demselben Thread bilden ein n-gram, weshalb sie von *Thread aware* N-Grammen sprechen. Grimmer et al. konnten zeigen, dass dies speziell auch für diesen Datensatz eine Verbesserung der Ergebnisse erzielt. Darüber hinaus zeigen sie, dass bei der Verwendung von *Thread aware* N-Grammen im Vergleich zu n-grammen ohne Berücksichtigung von Threads in keinem Szenario eine Verschlechterung der Ergebnisse auftritt.

zu dt. Thread bewusst

THREAD CHANGE FLAG Sie verwenden in der Auswertung aber ausschließlich Algorithmen welche kontextfrei arbeiten. Dies bedeutet, dass zuvor gesehene N-Gramme keinen Einfluss auf das aktuelle haben. Wie in Unterabschnitt 2.3.2 beschrieben wird, ist das bei LSTM Netzwerken nicht der Fall. Es gilt also den benötigten Kontext in die von Grimmer et al. beschriebene Methodik zu integrieren.

Eine Möglichkeit bestünde darin, die ThreadID zu kodieren und die Information aus welchem Thread das n-gram stammt mitzugeben. Doch das Kodieren der tatsächlichen Thread ID ist sehr unpraktisch, da Thread IDs wiederverwendet werden können. Stattdessen wird

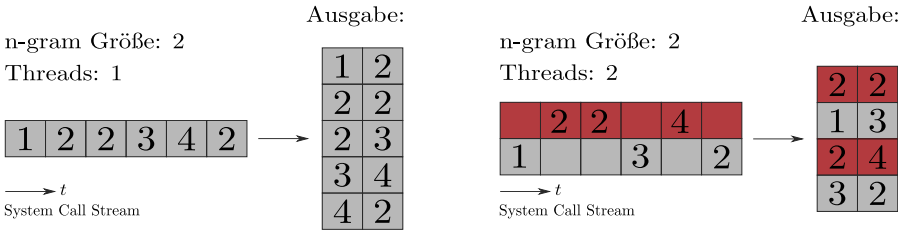

Abbildung 4.3: Erstellung der N-Gramme mit $n = 2$ aus einem Datenstream von System Calls. Links ohne und rechts mit der Beachtung der Threadinformation. Die dabei entstehenden N-Gramme dienen als Eingabe für den bewertenden Algorithmus. Die Reihenfolge der entstehenden N-Gramme als Eingaben ist dabei von oben nach unten.

in dieser Arbeit ein n-gram mit der Informatione über einen Kontextwechsel angereichert. Dieser Kontextwechsel soll dann über die Thread Change Flag (TCF) an das LSTM übergeben werden. Initial wird, weil kein Kontextwechsel stattfindet, für die TCF eine 0 angegeben. Kommt das aktuelle n-gram aus demselben Thread wie das n-gram zuvor, ist die TCF ebenfalls 0. Ist das aktuelle n-gram allerdings aus einem anderen Thread wird ein Kontextwechsel durch das Setzen der TCF auf 1 angezeigt. Visualisiert wird dies in Abbildung 4.4.

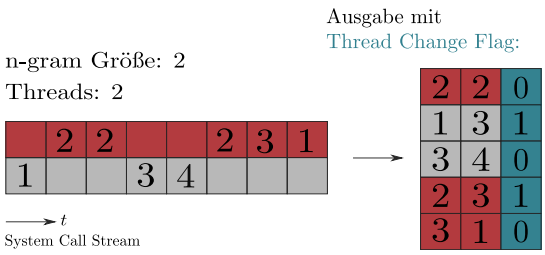

Abbildung 4.4: Analog zu Abbildung 4.3 werden N-Gramme erzeugt. Diese werden nun mit der TCF angereichert. Initial ist die TCF 0. Ist das zuvor ausgegeben n-gram aus demselben Thread, ist die TCF ebenfalls 0. Findet ein Wechsel des Threads statt, wird dieser Kontextwechsel durch das Setzen der TCF auf 1 signalisiert.

Die Vorstellung dabei ist, dass eine TCF= 1 dem LSTM die Information gibt, dass der zuvor gesehene N-Gramme nur eine untergeordnete Rolle für den Kontext spielen, da diese aus einem anderen Thread stammen. Falls dies der Fall ist, kann ein häufiger Kontextwechsel dafür sorgen, dass der Vorteil der LSTMs abgeschwächt wird, da immer dafür gesorgt wird, dass potentiell wenige Informationen im Kontext enthalten sind. In Tabelle 4.2 wird dargestellt wie oft so ein Kontextwechsel stattfindet. Dabei ist zu erkennen, dass in den Szenarien aus dem LID-DS bei einer Länge von $n = 6$ nur bis zu 13.8% der N-Gramme eine TCF $= 1$ haben. Die zuvor geschilderte Gefahr scheint also für die gegebenen Szenarien keine Rolle zu spielen, ist aber für andere

Einsatzbereiche mit nicht kontextfreien verarbeitenden Algorithmen zu beachten.

Thread Change Flag			
Szenario	#TCF= 1	#TCF= 0	Anteil TCF= 1 an allen N-Grammen in %
Bruteforce_CWE_307	2,534,165	20,383,314	11.1
$CVE-2012-2122$	1,206,151	10,365,309	10.4
$CVE-2014-0160$	1,120,786	7,026,864	13.8
$CVE-2017-7529$	1,130,717	10,721,010	9.5
$CVE-2018-3760$	1,453,876	38,255,576	3.7
$CVE-2019-5418$	3,131,792	74,159,462	4.1
$PHP\_CWE-434$	10,891,051	112,549,739	8.8
$EPS\_CEW-434$	12,657,844	374,439,042	3.3
ZipSlip	80,126,899	444,510,136	15.3

Tabelle 4.2: Vorkommen von eines Kontextwechsels angezeigt durch die TCF. Bei einer n-gram Länge von 6.

4.3 AUFBAU DES LSTMs

Die Architektur wird ebenfalls in Abbildung 4.5 dargestellt und besteht aus der Eingabe-Layer, der LSTM-Layer, einer *fully-connected* Linear-Layer und der Output-Layer. Wie dargestellt, wird für die Initialisierung der Implementierung des LSTMs die Anzahl an Input und Output Neuronen benötigt. Für die Eingangsgröße gilt:

$$input\_dim = (e + ret + time) \cdot n + \text{TCF} \tag{4.3}$$

Mit e für die gewählte Größe des W2V-Embeddings. Falls der zuvor beschriebene Zusatzparameter für die Rückgabewerte verwendet wird gilt $ret = 1$ ansonsten $ret = 0$. Gleiches gilt für den Extraparameter welcher die Zeitabstände kodiert und Kontextwechsel angebende TCF. Für die Ausgangsgröße gilt:

$$output\_dim = k + 1 \tag{4.4}$$

Dabei ist k die Anzahl an System Calls aus dem Trainingsdatensatz. Für unbekannte System Calls, welche in den Trainingsdaten nicht vorkamen, wird zusätzlich noch eins addiert. Nötig ist diese Variabilität, da für die Initialisierung des LSTMs die Anzahl der verschiedenen System Calls im Trainingsdatensatz für die Ausgangsgröße, sowie die Eingangsgröße des System Calls eine entscheidende Rolle spielt.

Festgelegt werden hingegen Größen für die *Hidden Layer* Dimension, *Batch Size*, den *Optimizer*, die *Loss*-Funktion und die Lernrate. Die

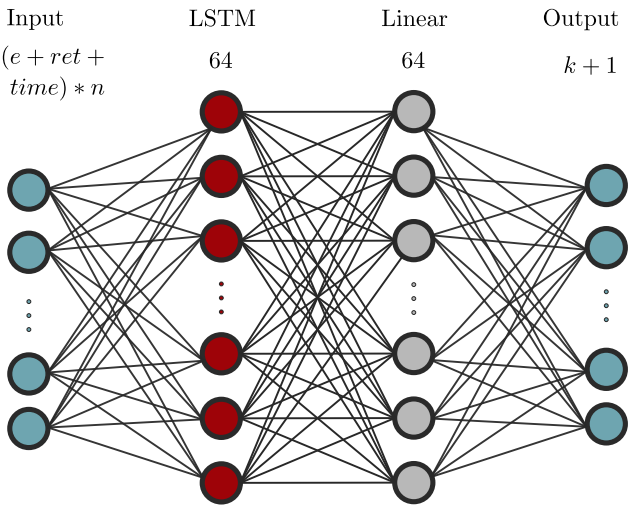

Abbildung 4.5: Aufbau des LSTM mit 64 Neuronen für die LSTM- und Linear-Layer. Die Input Layer besitzt $(e + ret + time) * n$ Neuronen, mit e für die Größe des W2V-Embeddings, $ret = 1$ falls Rückgabewerte verwendet werden und $time = 1$ falls die Zeitabstände verwendet werden. Ansonsten sind $ret = 0$ und $time = 0$. Und die Output Layer hat $k + 1$ Ausgangsneuronen, dabei ist k die Anzahl an System Calls aus dem Trainingsdatensatz. Eins wird addiert für unbekannte System Calls.

Dimension der LSTM-Layer wird auf 64 gesetzt, die *Batch Size* auf 1024. Es wird der für LSTMs gängige *Adam Optimizer* gewählt. Da es sich um eine Klassifikation mit $k + 1$ Klassen handelt, wird die *Cross-Entropy* Loss-Funktion des PyTorch Frameworks genutzt. Die Lernrate wird auf 0.001 festgelegt. Details zu der Parameterwahl wird in Unterabschnitt 4.4.4 erläutert.

Entgegen der typischen Implementierungen von LSTMs wird der *Hidden State* nach einem Batch gespeichert und dem nächsten Batch übergeben. Der Hauptgrund dafür ist die Länge der System Call Streams. Diese können länger als ein Batch sein. Standardmäßig werden die Hidden States nicht übergeben, weswegen die Kontextinformation zwischen den Batches verloren geht. Diese Idee ist allerdings ausgelegt für einen kontinuierlichen Stream an System Calls, der Datensatz besteht aber aus vielen Dateien. Deswegen wird der *Hidden State* zum Ende einer Aufnahme zurückgesetzt.

Ausgaben des LSTMs sind durch die Verwendung der PyTorch CrossEntropy Loss-Funktion sogenannte *Logits*. Auf diese kann mithilfe der *Softmax*-Funktion eine Liste an Wahrscheinlichkeiten für jeden System Call sowie den Platzhalter erstellt werden.

4.4 ALGORITHMUS

Im vorigen Kapitel wurden alle verwendeten Vorverarbeitungsschritte vorgestellt. In den kommenden Abschnitten werden diese Schritte zu einem Ablauf zusammengeführt. Zunächst soll in Unterabschnitt 4.4.1 ein allgemeiner Überblick gegeben werden. Anschließend wird in Unterabschnitt 4.4.2 behandelt was das LSTM Netzwerk lernt. Schließlich erklären Unterabschnitt 4.4.3 und Absatz 4.4.3, wie aus dem Gelernten die Klassifizierung in normales oder Angriffsverhalten erfolgt.

4.4.1 *Allgemein*

Wie bereits in Abschnitt 3.3 beschrieben, werden verschiedene Verfahren der NLP auch in der anomaliebasierten HIDS eingesetzt. Neben den bereits beschriebenen W2V Verfahren, kommen auch die in der NLP verbreiteten LSTM Netzwerke zum Einsatz. Mit dem LSTM soll in der Trainingsphase ein Sprachmodell der System Calls erstellt werden. Dieses Sprachmodell soll mit einer gegebenen Sequenz $s = (sc_{x_1}, \ldots, sc_{x_n})$ den folgenden System Call $sc_{x_{n+1}}$ vorherzusagen. Dabei bestehen die Trainingsdaten nur aus Normalverhalten, enthalten also keine Angriffe. Anhand dieses gelernten Modells soll dann in der Testphase eine Vorhersage über den nächsten System Call gemacht werden. Diese Vorhersage besteht aus den Wahrscheinlichkeiten für alle aus dem Training gesehenen System Calls, plus einem Platzhalter für noch unbekannte System Calls. Die Wahrscheinlichkeit des tatsächlich auftretenden System Calls von eins abgezogen, stellt dann den Anomaliescore dar. Ob dieser Anomaliescore zu einem Alarm führt oder nicht, hängt von einem zuvor bestimmten Schwellwert ab. Dieser wird bestimmt, indem nach dem Training die Anomaliescores für einen Ausschnitt des Datensatzes bestimmt werden. In diesem Ausschnitt, auch Validierungsdaten genannt, darf wie in den Trainingsdaten kein Angriffsverhalten enthalten sein. Eine Garantie die in der echten Welt nie komplett gegeben werden kann. In den Validierungsdaten soll folglich kein Alarm ausgelöst werden, weshalb der höchste Anomaliescore der Validierungsdaten als Schwellwert der Anomalieerkennung verwendet wird. Dies ist eine Annäherung an einen optimalen Schwellwert.

Die System Calls werden wie zuvor beschrieben durch das W2V Verfahren in einen Vektor umgewandelt und durch die zwei zuvor beschriebenen Verfahren, die Normalisierung der Zeitabstände und der Rückgabewerte bestimmter System Calls, angereichert. Der Stream der System Calls wird dann mit Betrachtung der Threads in N-Gramme zusammengetragen. Zusätzlich wird zu jedem n-gram die Information über einen Kontextwechsel durch die TCF angegeben.

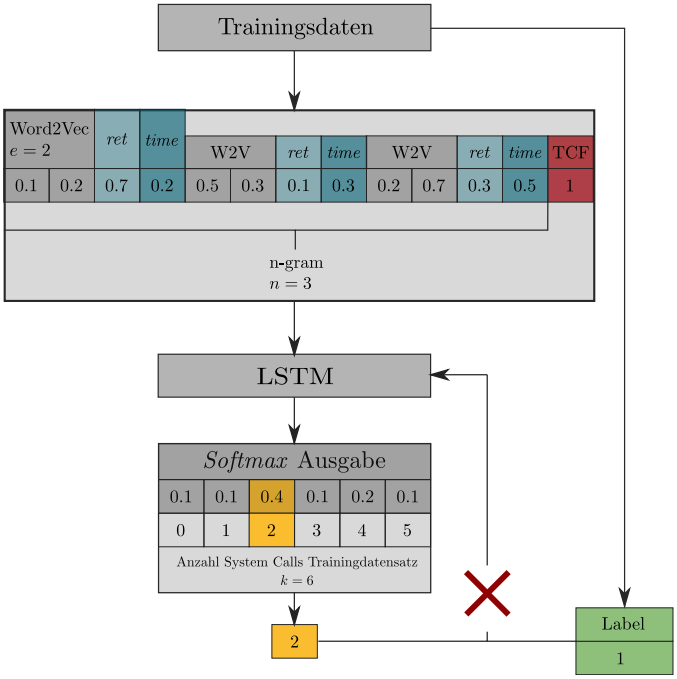

Abbildung 4.6: Ablauf der Trainingsphase mit Beispieldaten. Aus Trainingsdaten wird ein n-gram aus W2V, Rückgabewert (*ret*) und Zeitabstand (*time*) erstellt. Diese beiden Extraparameter sind bläulich hervorgehoben. Zusätzlich wird die TCF an das n-gram gefügt. Der Index des höchsten Wertes (gelb) aus der Ausgabe des LSTM wird mit dem eigentlichen Label (grün) verglichen. Aufgrund dieser Information werden die Gewichte im LSTM angepasst

4.4.2 *Training*

Pro Szenario werden 200 Aufnahmen ohne Angriffsverhalten, der insgesamt ca. 1100 Aufnahmen, für das Training ausgewählt. Die Grundidee besteht wie zuvor beschrieben darin, mit einer gegebenen Sequenz $s = (sc_{x_1}, \ldots, sc_{x_n})$ den folgenden System Call $sc_{x_{n+1}}$ vorherzusagen. Da der nächste System Call aus dem Datensatz stets bekannt ist, kann als Feedback für das Netzwerk die Information genommen werden, ob $sc_{x_{n+1}}$ korrekt vorhergesagt wurde, also dem Label entspricht. Das LSTM gibt wie in Abbildung 4.6 dargestellt diese Information an das LSTM zurück.

Seien die System Calls aus den Trainingsdaten aus der Menge $SC = \{sc_1, sc_2, \ldots, sc_k\}$. Womit k die Anzahl der in den Trainingsdaten vorkommenden System Call Namen angibt. Die Trainingsdaten werden durchlaufen und N-Gramme $ngram_i$ erstellt. Der auf das $ngram_i$ folgenden System Call dient als Label l_i. Nach dieser Idee ist das erste n-gram $ngram_0$ mit Label l_0. Dabei werden die System Call Namen für die Label durch Integerwerte ersetzt. Das LSTM Netzwerk bekommt $ngram_0$ und gibt einen Vektor der Länge $k + 1$ zurück. Dieser Vektor gibt für jeden der k möglichen System Calls aus den Trainingsdaten

eine Auftrittswahrscheinlichkeit an. Zusätzlich wird 1 addiert, da in den Testdaten unbekannte System Calls auftreten können. Diese bekommen den W2V-Vektor $(0)^e$, wobei e die Länge des W2V-Vektoren ist oder für das Label einen Integerwert von 0. Die Ausgabe des LSTMs sind *PyTorch Logits*, diese werden mithilfe der *Softmax*-Funktion in die gewünschten Wahrscheinlichkeiten umgewandelt. Der Index der Liste der so berechneten Wahrscheinlichkeiten wird mit dem aus den Trainingsdaten gezogenen Label verglichen. Dieser Vergleich dient dann als Feedback für das LSTM. Mit den gegebenen Trainingsdaten kann nun das LSTM mittels des *back-propagation through time* trainiert werden.

4.4.3 *Anomalieerkennung*

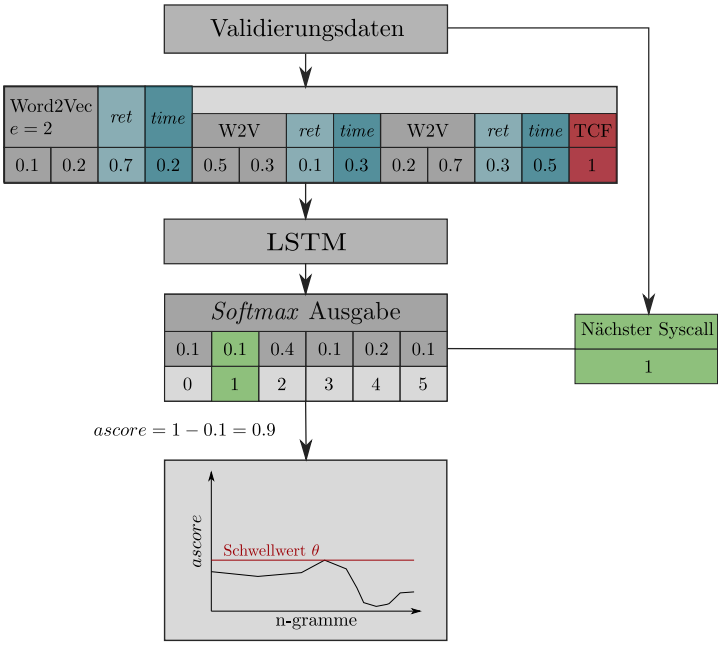

Abbildung 4.7: Ähnlich zu Abbildung 4.6 wird hier der Validierungsablauf beschrieben. Dabei wird wie zuvor ein Datenauszug dargestellt sowie die Auftrittswahrscheinlichkeiten für den folgenden System Call. In der Test-, wie in der Validierungsphase dient die Wahrscheinlichkeit des Integerwerts des tatsächlich als nächstes auftretende System Call als Berechnungsgrundlage für den Anomaliescore. Wie unten in der Abbildung dargestellt, wird das Maximum der *ascore*s der gesamten Validierungsdaten als Schwellwert festgelegt.

Es kann also bei Auftreten des System Calls sc_i überprüft werden mit welcher Wahrscheinlichkeit p_i dieser vorhergesagt wurde. Der eigentliche Anomalie-Score *ascore* für sc_i wird dann folgenderweise berechnet:

$$ascore = 1 - p_i \tag{4.5}$$

Übertrifft dieser Wert einen zuvor bestimmten Schwellwert θ, so wird sc_i als Anomalie und damit als Angriff gewertet.

SCHWELLWERTBESTIMMUNG Für die Berechnung des Schwellwertes θ wird für alle System Calls der Validierungsdaten *ascore* berechnet. Die Validierungsdaten bestehen aus 50 Aufnahmen und enthalten ebenfalls wie die Trainingsdaten nur das Normalverhalten. Da in den Validierungsdaten kein Angriffsverhalten integriert ist, darf keiner der *ascore*s einen Alarm auslösen. Weshalb der höchste Wert der *ascore*s den Validierungs als Schwellwert θ genutzt wird. Der Ablauf der Schwellwertfindung wird zusätzlich in Abbildung 4.7 dargestellt. Dabei ist ähnlich zu Abbildung 4.6 dargestellt wie ein System Call n-gram kodiert wird und als Eingabe in das LSTM dient. Zusätzlich wird die Berechnung des *ascores* dargestellt, indem das Label als Index für die Softmax Ausgabe des LSTMs dient. Das Maximum der *ascores* wird wie beschrieben dann als Schwellwert θ genutzt.

4.4.4 *Parameterwahl*

Das Aufsetzen des beschriebenen Algorithmus bedarf der Festlegung einiger Parameter. Dazu gehören allgemeine Parameter für neuronale Netze, aber auch spezifischere Parameter wie die Länge der N-Gramme. Da nicht alle Parameter experimentell untersucht werden können werden sie im Folgenden in festgelegte und zu ermittelnde Parameter unterteilt.

FESTGELEGTE PARAMETER Entscheidende Parameter der LSTMs sind die Anzahl der Neuronen pro Layer sowie die Anzahl der LSTM-Layer in dem Netzwerk. Da sich bei ersten Tests schnell die Problematik aufgetan hat, dass die Berechnungen sehr großer Szenarien zu nicht tragbaren Laufzeiten geführt hat, muss dies bei der Findung vieler Parameter bedacht werden. Deshalb wird in dieser Arbeit nur eine LSTM-Layer verwendet, auch wenn in manchen Anwendungen der NLP mehrere Layer zu einer verbesserten *Accuracy* führen.[2] Die Anzahl der Neuronen dieser Layer wird wie bereits in Abschnitt 4.3 beschrieben auf 64 festgelegt. Die Lernrate wurde anhand der Ergebnisse mehreren Tests auf wenigen Szenarien auf 0.001 festgelegt. Außerdem wird der für LSTMs häufig eingesetzte Adam-Optimizer verwendet. Diese Einstellungen wurden auch in der Arbeit von Dymshits et al. [16] eingesetzt. Als Loss Funktion wird für das $k + 1$ Klassen Klassifizierungsproblem die *Cross-Entropy* gewählt. Wie auch in Abbildung 4.8 erkennbar ist stagniert der Fehler bei verschiedenen Szenarien nach nur wenigen Epochen, weshalb für die Epochenanzahl ein Maximum von 20 festgelegt wird. Der Einfluss der Batch size auf die Ergebnisqualität wird unter anderem in [8] als gering eingestuft und wird auf 1024 festgelegt.

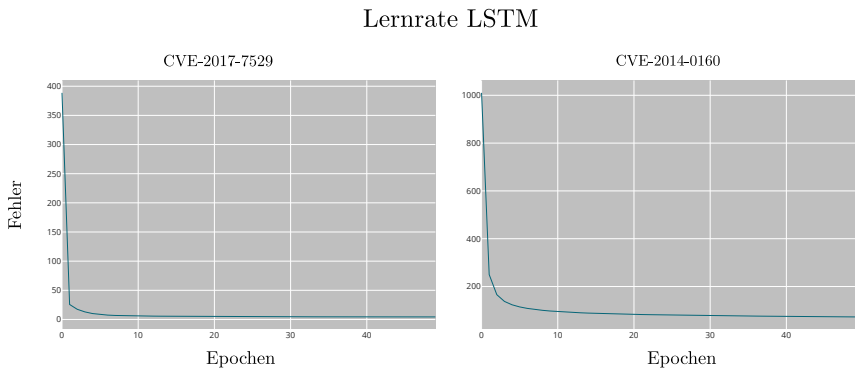

Abbildung 4.8: Darstellung des Fehlers während des Trainingsvorgangs für zwei Szenarien. Zum einen ist ein Lernfortschritt zu erkennen und zum anderen tritt dieser bereits nach wenigen Epochen ein.

Da das Berechnen des W2V im Vergleich zum Training und Testen des LSTMs nur einen geringen Zeitaufwand benötigt wird eine Epochenanzahl von 500 festgelegt. Es werden aufgrund der Ergebnissen von Grimmer et al. [25] nach verschiedenen Tests mit alten Konfigurationen nur N-Gramme mit Betrachtung der Thread Information aufgebaut.

ZU ERMITTELNDE PARAMETER Ziel ist es eine Konfiguration der Parameter zu finden, mit der alle Szenarien aus dem LID-DS mit hoher DR und niedriger FP-Rate ausgewertet werden können. Dafür gilt es speziell verschiedene **n-gram** Größen in Kombination mit verschiedenen Größen für das **W2V-Embedding** zu finden. Für die n-gram Größen wurden $[2, 6, 10]$ und für das W2V-Embedding $[4, 6, 8, 10]$ zur Auswahl gestellt. Zusätzlich soll untersucht werden ob sich durch die Hinzunahme weiterer System Call Parametern neue erfolgreiche Konfigurationen ergeben. Speziell wird getestet ob sich **TCF**, **Rückgabewerte**, **zeitliche Abstände** oder jegliche **Kombinationen** dieser positiv auf die Ergebnisse auswirken.

4.5 METRIKEN

Typische Metriken für die Auswertung von neuronalen Netzen und damit auch für LSTMs, welche auch in der Anomalieerkennung eingesetzt werden, umfassen die *Precision*, *Recall* und den *F1-Score*. Diese werden wie in Abbildung 4.9 dargestellt berechnet. Da der Datensatz aus System Calls besteht bedeutet Recall in diesem Fall, der Anteil an korrekt erkannten bösartigen System Calls, an allen bösartigen System Calls. Wie in Absatz 2.2.3 beschrieben liefert der Datensatz allerdings nur den Angriffszeitpunkt und keine Label für jeden einzelnen System Call. Also muss jeder System Call nach dem Angriffszeitpunkt als bösartig eingestuft werden, obwohl neben dem Angriffsverhalten

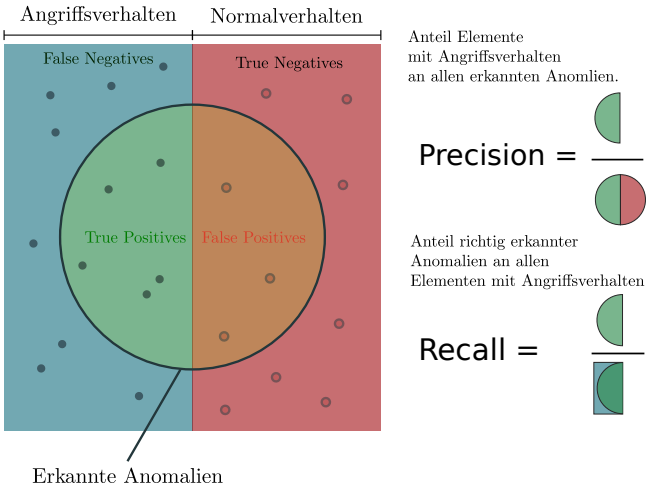

Abbildung 4.9: Visualisierung der Berechnung der häufig verwendeten Metriken der Precision und des Recalls, bezogen auf die Domäner der *Intrusion Detection*.

noch Normalverhalten stattfindet. Wird von einem optimalen Fall ausgegangen, dass alle dem Angriff zugehörigen System Calls einen Alarm auslösen, wird der Recall immer noch sehr niedrig sein und damit keinen realistischen Wert liefern. Somit ist auch der F1-Score wenig nützlich, da sich dieser aus Recall und Precision zusammensetzt. Es scheint deshalb sinnvoller andere Metriken zur Hand zu nehmen, um einheitliche und vom Datensatz unabhängige Messwerte bereitstellen zu können. Da der LID-DS aus vielen verschiedenen Aufnahmen besteht, kann die Berechnung der Precision und des Recalls auch auf Dateiebene stattfinden. Falls mehrere Angriffe in einer Datei vorkommen wird die Berechnung jedoch umständlicher, dies ist im LID-DS nicht der Fall. Grundgedanke der Findung einer neuen Metrik muss dabei auch sein, einen klaren Praxisbezug zu finden. Dort ist vor allem interessant wieviele Angriffe erkannt werden und nicht wieviele System Calls der Angriffe als bösartig eingestuft werden. Zum anderen entscheidet die Anzahl, beziehungsweise die Häufigkeit von Fehlalarmen über die Nutzbarkeit der IDSs. Die DR steht für den zuvor beschriebenen Recall auf Dateiebene, also der Anteil der erkannten Angriffe an allen vorkommenden Angriffen. Werden alle Aufnahmen mit Angriffsverhalten korrekt erkannt gilt: $DR = 1$. Dabei wird außer Acht gelassen, dass ein Alarm nach dem Angriffszeitpunkt wie in Absatz 2.2.3 beschrieben theoretisch auch ein Fehlalarm sein kann. Nur ein Alarm vor dem Angriffszeitpunkt oder in einer Aufnahme ohne Angriffsverhalten wird als Fehlalarm beziehungsweise als FP gewertet. Zusätzlich gelten alle System Calls solange zu einem Alarm, bis der *ascore* wieder unter dem Schwellwert θ liegt. Ziel ist es also eine hohe DR und eine möglichst geringe FP-Rate zu erreichen. Um einschätzen zu können welche Konfiguration am besten abgeschnitten hat, ergibt sich eine neue Schwierigkeit. Es ist schwierig zu beurteilen

welche Konfiguration am besten funktioniert, denn wenn alle System Calls aus dem Datensatz als Angriff erkannt werden, ist DR = 1. Eine hohe DR alleine ist also wenig aussagekräftig. In der Auswertung muss deswegen eine Abwägung zwischen FP-Rate und DR stattfinden.

5 ERGEBNISSE

In diesem Kapitel sollen die Ergebnisse der Experimente dargestellt und ausgewertet werden. Die Strukturierung der Experimente werden in Abschnitt 5.1 aufbereitet und in Abschnitt 5.2 wird auf die Problematik der Berechnungsdauer eingegangen. Abschnitt 5.3 untersucht Konfigurationen des LSTM ohne Verwendung von Extraparametern. Dabei werden diese speziell nur entweder nach DR oder Anzahl der FPs eingestuft. In Abschnitt 5.4 werden auch Ergebnisse mit Extraparametern hinzugezogen. Zusätzlich werden in Abschnitt 5.5 Ergebnisse weiterer Arbeiten mit dieser Arbeit verglichen. Dabei gibt es wie in Kapitel 3 beschrieben nur eine Arbeit, die ebenfalls ein LSTM auf demselben Datensatz verwenden.

5.1 STRUKTURIERUNG DER EXPERIMENTE

Ziel der Experimente soll es sein eine Konfiguration zu finden, die bei allen Szenarien des LID-DS gut abschneidet. Dafür werden wie zuvor beschrieben die n-gram und Embedding Größe sowie verschiedene Kombinationen der Extraparameter getestet. Wieso nicht zusätzlich verschiedene Architekturen, Batchgrößen oder andere Einstellungen in die Konfigurationen einfließen, liegt im Wesentlichen an den vorhandenen Ressourcen. Dies wird in Abschnitt 5.2 genauer betrachtet. Anhand der Experimente wird zunächst untersucht, ob es Konfigurationen des LSTMs ohne zusätzliche Parameter gibt, welche konkurrenzfähige Ergebnisse liefern können. Danach sollen auch die zusätzlichen Parameter eingebunden werden und es wird geprüft inwiefern sich die Extraparameter auf die Ergebnisse auswirken. Dies zu untersuchen, erweist sich jedoch als nicht so einfach. Eine Möglichkeit besteht darin dieselben Konfigurationen mit und ohne Extraparametern zu testen und die Ergebnisse zu vergleichen. Doch die Hinzunahme von Extraparametern wird die optimale Konfiguration der LSTMs voraussichtlich beeinflussen, was den Vergleich negativ beeinflusst. Eine weitere Möglichkeit könnte es sein, den Mittelwert aller Ergebnisse für jede Extraparameterkonfiguration zu vergleichen. Da aber eine beste Konfiguration gesucht wird, ist es möglich, dass viele Konfigurationen, außer der besten, schlecht abschneiden und damit den Mittelwert entscheidend verschlechtern. Mit dieser Herangehensweise kann es passieren, dass Konfigurationen fälschlicherweise nicht mehr berücksichtigt werden. Nichtsdestotrotz wird in Abschnitt 5.4 und Abschnitt 5.5 versucht durch verschiedene Darstellungen einen sinnvollen Vergleich zu ziehen, um so sinnvolle Konfigurationen darzulegen.

5.2 BERECHNUNGSDAUER

Wie bereits in mehreren Kapiteln erwähnt ist die Berechnungszeit eine der größten Schwierigkeiten bei der Verwendung dieses LSTM-basierten Algorithmus in der HIDS. Für die Berechnungen wurden zwei verschiedene Grafikkarten auf dem Galaxy Cluster der Universität Leipzig genutzt. Das EPS CWE-434 sowie das ZipSlip Szenario wurden auf einer NVIDIA Tesla V100 Grafikkarte berechnet und für die restlichen kleineren Szenarien wurde eine NVIDIA RTX 2080 Ti verwendet. Die Trainingszeit spielt für den Live-Betrieb eine untergeordnete Rolle, da das Training nur einmalig durchgeführt werden muss. Relevanter ist die Berechnungszeit die für die Auswertung der Testdaten benötigt wird. Diese durchschnittlichen Berechnungszeiten werden im Folgenden für die verschiedenen Szenarien gelistet. Wie zu

Berechnungszeiten der verschiedenen Szenarien	
Szenario	$\overline{DetectionTime}$ in min
CVE-2017-7529	16.22
CVE-2014-0160	34.48
Bruteforce CWE-307	53.69
CVE-2012-2122	55.87
CVE-2019-5418	155.69
CVE-2018-3760	171.56
PHP CWE-434	193.55
SQL Injection CWE-89	212.00
EPS CWE-434	1157.83
ZipSlip	2253.75

Tabelle 5.1: Durchschnittliche Berechnungszeiten zur Bestimmung der Anomaliescores aller Testdaten der einzelnen Szenarien.

erkennen ist, benötigen die beiden größten Szenarien trotz der besseren Grafikkarte wesentlich länger. Um die in Tabelle 5.1 beschriebenen Ergebnisse zu erzielen wurden bereits einige Abstriche gemacht. So wurde kein Vergleich des OHE mit dem W2V gezogen, aufgrund der Größe des OHE. Auch wurden die möglichen Parameter sowie Parametergrößen stark eingeschränkt, um die Anzahl der zu berechnenden Konfigurationen möglichst gering zu halten. Für die Auswahl wurden Tests auf kleineren Szenarien durchgeführt.

Bei größeren Eingaben, also N-Grammen größer 2 oder Embedding Größen größer als 8 bei einer n-gram Größe von größer 2 reicht auch der Speicher der NVIDIA Tesla V100 Grafikkarte für das ZipSlip Szenario nicht mehr aus. Das mindert leider die Vergleichbarkeit der Ergebnisse, was die Erkenntnisse dieser Arbeit aber nur geringfügig

beeinflusst. Konfigurationen, welche ein Ergebnis für das ZIPSlip Szenario beinhalten sind extra blau markiert.

5.3 LSTM ANSATZ

Im ersten Schritt der Auswertung wird die DR isoliert betrachtet. Dabei werden nur Konfigurationen einbezogen, die keine zusätzlichen Parameter der System Calls enthalten. Alle Konfigurationen verwenden dabei *Thread Aware* N-Gramme, also nur System Calls mit derselben ThreadID werden in ein n-gram aufgenommen. In Tabelle 5.2 werden die zehn besten Ergebnisse dargestellt. Die durchschnittliche DR von 0.71 bei 14.56 Fehlern pro Szenario zeigt, dass es generell möglich ist mit LSTMs eine Anomalieerkennung auf Basis von System Calls umzusetzen. Eine klarer Zusammenhang zwischen Embedding Größe (e) oder n-gram Größe (n) und der DR ist dabei nicht direkt zu erkennen. So hat die beste Konfiguration mit einer n-gram Größe von 10 und einer Embedding Größe von 4 $10 \cdot 4 = 40$ Eingangsneuronen. Hingegen hat die Konfiguration aus Zeile 4 nur $2 \cdot 4 = 8$ Eingangsneuronen. Beide Konfigurationen erzielen dabei sehr ähnliche Ergebnisse mit einer durchschnittlichen DR von 0.71 und 14.56 FPs beziehungsweise 0.68 und 12.89.

Ohne Extraparameter nach DR			
n	e	$\overline{\text{FP}}$	$\overline{\text{DR}}$
10	4	14.56	0.71
6	6	22.22	0.69
2	10	9.22	0.68
2	4	12.89	0.68
10	8	17.78	0.67
2	6	20.60	0.64
6	4	11.34	0.58
10	10	5.78	0.56
10	6	6.44	0.56
6	10	7.33	0.56

Tabelle 5.2: Konfigurationen mit den 10 höchsten DRs. Dabei wurden nur Konfigurationen betrachtet die keine Extraparameter nutzen. Alle N-Gramme sind *Thread Aware*. Nur blaue Zeilen enthalten Ergebnisse des ZipSlip Szenarios.

Im zweiten Schritt der Auswertung sollen die besten Ergebnisse nur im Bezug auf die Anzahl der Fehlalarme, also FPs untersucht werden. In Tabelle 5.3 werden wieder *Thread Aware* N-Gramme genutzt und nur Ergebnisse mit DR > 0.5 einbezogen.

Ohne Extraparameter, nach FP			
n	e	$\overline{\text{FP}}$	$\overline{\text{DR}}$
10	10	5.78	0.56
10	6	6.44	0.56
6	10	7.33	0.56
2	8	8.40	0.50
2	10	9.22	0.68
6	4	11.34	0.58
2	4	12.89	0.68
10	4	14.56	0.71
10	8	17.78	0.67
2	6	20.60	0.64

Tabelle 5.3: Konfigurationen mit den 10 niedrigsten FPs. Dabei wurden nur Konfigurationen betrachtet die keine Extraparameter nutzen. Alle N-Gramme sind *Thread Aware*. Es wurden nur Konfigurationen mit DR > 0.5 einbezogen. Nur blaue Zeilen enthalten Ergebnisse des ZipSlip Szenarios.

Hierbei überschneiden sich viele Konfigurationen aus Tabelle 5.2. Das liegt vor allem daran, dass es nur wenig mehr als zehn Konfigurationen ohne Extraparameter mit einer DR > 0.5 gibt. Wieder lässt sich kein klarer Trend in den Konfigurationen von n-gram und Embedding Größe erkennen. Doch die Tabelle kann genutzt werden um noch einmal auf die Schwierigkeit der Auswahl der besten Konfigurationen hinzuweisen. Ist eine Konfiguration mit hoher DR oder eine Konfiguration mit wenigen FPs zu bevorzugen? Oder konkret wie in diesem Fall, ist eine DR $= 0.71$ bei im Schnitt ca. 15 Fehlern in grob acht Stunden Testaufnahmen der Konfiguration mit einer DR $= 0.56$ bei ca. 6 Fehlern zu bevorzugen? Der Versuch eine beste Konfiguration auszuwählen soll nach Betrachtung der Konfigurationen mit Extraparametern unternommen werden.

In Tabelle 5.4 werden für die beiden besten Konfigurationen alle Szenarien aufgeschlüsselt. Zu erkennen ist dabei, dass in dem $CVE-2014-0160$ wie auch dem $CVE-2012-2122$ Szenario bei der Konfiguration mit der besten DR insgesamt keine Angriffe erkannt werden. Interessanter weise löst die Konfiguration mit der geringsten FP-Rate das $CVE-2014-0160$ Szenario fast perfekt mit nur einem Fehler. Ein Szenario in welchem das LSTM typischerweise schlecht abschneidet. Dies zeigt sich auch in Tabelle 5.5. Hier wird die durchschnittliche DR für alle Szenarien dargestellt. Dabei werden bei vier Szenarien im Schnitt nur sehr wenige Angriffe erkannt. Dazu gehören das *ZipSlip* Szenario sowie $CVE-2012-2122$, $CVE-2014-0160$ und das $Bruteforce\_CWE-307$. Problematisch sind dabei speziell das

Ohne Extraparameter - pro Szenario					
Szenario	$\overline{\text{FP}}$	$\overline{\text{DR}}$	vs	$\overline{\text{FP}}$	$\overline{\text{DR}}$
Bruteforce_CWE − 307	60	0.94	x	15	1.00
CVE − 2012 − 2122	8	0.01	x	4	0.03
CVE − 2014 − 0160	3	0.00	x	1	0.99
CVE − 2017 − 7529	1	0.99	x	0	0.05
CVE − 2018 − 3760	11	1.00	x	9	0.01
CVE − 2019 − 5418	32	1.00	x	12	1.00
EPS_CWE − 434	14	1.00	x	9	1.00
PHP_CWE − 434	4	0.96	x	1	1.00
SQL_Injection_CWE − 89	1	0.46	x	1	0.00
ZipSlip	x	x	x	x	x

Tabelle 5.4: Auflistung der einzelnen Szenarien für die Konfiguration mit der höchsten DR ($n = 10$, $e = 4$) links und der wenigsten FPs ($n = 10$, $e = 10$)

ZipSlip Szenario sowie das *Bruteforce_CWE* − 307 Szenario, da sie zusätzlich zu einer geringen DR auch zu vielen FPs führen.

Ohne Extraparameter - pro Szenario		
Szenario	$\overline{\text{FP}}$	$\overline{\text{DR}}$
Bruteforce_CWE − 307	22.09	0.29
CVE − 2012 − 2122	14.00	0.02
CVE − 2014 − 0160	3.25	0.08
CVE − 2017 − 7529	0.58	0.82
CVE − 2018 − 3760	17.00	1.00
CVE − 2019 − 5418	12.17	0.58
EPS_CWE − 434	14.80	1.00
PHP_CWE − 434	10.09	0.91
SQL_Injection_CWE − 89	9.81	0.95
ZipSlip	35.5	0.15

Tabelle 5.5: Darstellung der durchschnittlichen DR und FPs für alle Szenarien mit Konfigurationen ohne Extraparametern. Dabei muss beachtet werden, dass bei den ZipSlip Ergebnissen weitaus weniger Konfigurationen in den Durchschnitt eingeflossen sind.

5.4 EINSATZ VON EXTRAPARAMETER

In Tabelle 5.6 werden die Ergebnisse der Konfigurationen mit und ohne Extraparametern angezeigt. Wieder werden die Ergebnisse zunächst nach der besten DR eingestuft. Als erstes fällt dabei ins Auge, dass die besten sechs der zehn Konfigurationen Extraparameter verwenden. Im Vergleich der besten Konfigurationen werden ohne Extraparameter wie oben beschrieben eine DR von 0.71 bei durchschnittlich 14.56 Fehlern erreicht. Mit Extraparametern beträgt die höchste DR 0.88 bei durchschnittlich 22.89 Fehlern. Also auf die DR bezogen eine deutliche Verbesserung. Im Gegensatz zu Tabelle 5.2 ist diesmal ein Zusammenhang zwischen der Embedding sowie n-gram-Größe und der Ergebnisqualität zu erkennen. So liefert die Konfiguration mit $n = 10$ und $e = 4$ für viele Extraparameterkombinationen gute Ergebnisse. Sie kommt fünf mal unter den besten 7 Ergebnissen vor. Das beste Ergebnis liefert eine n-gram Größe von 6 bei einer Embedding Größe von 8. Zusätzlich ist sie auch die beste Konfiguration ohne Extraparameter. Es ist auch die einzige Konfiguration ohne Extraparametern, die es in unter die besten zehn geschafft hat.

Mit Extraparametern nach DR						
n	*e*	*rv*	TCF	*time*	$\overline{\text{FP}}$	$\overline{\text{DR}}$
6	8	1	1	1	22.89	0.88
10	4	1	1	0	14.00	0.76
10	8	0	1	1	18.56	0.76
10	4	0	1	0	5	0.74
10	4	1	0	0	13.78	0.74
10	4	1	1	1	22.33	0.74
10	4	0	0	0	14.56	0.71
6	8	1	1	0	9.00	0.70
2	8	1	1	1	14.00	0.70
6	4	1	0	1	18.22	0.70

Tabelle 5.6: Konfigurationen mit den 10 höchsten DRs. Es wurden Konfigurationen mit und ohne Extraparameter betrachtet. Alle N-Gramme sind *Thread Aware*. Nur blaue Zeilen enthalten Ergebnisse des ZipSlip Szenarios.

Auch bei den Ergebnissen nach den geringsten FPs in Tabelle 5.7 verwenden sieben der besten zehn Konfigurationen Extraparameter, darunter auch die besten drei. So kann mit nur durchschnittlich 4.66 Fehlalarmen bei einer DR von 0.67 eine deutliche Steigerung gegenüber durchschnittlich 5.78 Fehlern bei einer DR von 0.56 erreicht werden. Interessant ist, dass auch hier wieder die Konfiguration $n = 10$ und $e = 4$ unter den besten Ergebnissen auftaucht und das die beiden

besten Ergebnisse durch die Konfiguration $n = 6$ und $e = 8$ erreicht wurden. *nach DR und FP*

Mit Extraparametern nach FP						
n	e	rv	TCF	$time$	$\overline{\text{FP}}$	$\overline{\text{DR}}$
6	8	0	0	1	4.66	0.67
10	4	0	1	0	5	0.74
10	6	0	1	0	5.66	0.60
10	10	0	0	0	5.78	0.56
6	4	0	0	1	5.80	0.64
6	6	1	1	1	6.33	0.66
10	6	1	0	0	6.33	0.55
10	6	0	0	0	6.44	0.56
6	10	0	0	0	7.33	0.56
6	6	0	0	1	7.89	0.64

Tabelle 5.7: Konfigurationen mit den 10 niedrigsten FP-Raten. Es wurden Konfigurationen mit und ohne Extraparameter betrachtet. Alle N-Gramme sind *Thread Aware*. Auch hier kommen bei den meisten Konfigurationen die Extraparameter zum Einsatz.

Wie in Abschnitt 5.1 beschrieben kann die Betrachtung des Mittelwertes auch für nicht repräsentative Ergebnisse sorgen. Dadurch können gute Konfigurationen übersehen werden. Denn wie in Tabelle 5.8 dargestellt, werden die Ergebnisse mit Hinzunahme im Schnitt nicht deutlich besser. Dies kann zu einem frühzeitigen Ausschluss bestimmter Konfigurationen führen. Die Verbesserungen zu Tabelle 5.5 werden mit blau und die Verschlechterungen mit rot dargestellt. Eine deutliche Verbesserungen ist dabei nicht zu erkennen.

In Tabelle 5.9 und Tabelle 5.10 werden die zwei Gewinnerkonfigurationen aus Tabelle 5.6 und Tabelle 5.7 mit jeweils den Gewinnerkonfigurationen ohne Extraparametern gegenübergestellt. Dabei werden die Ergebnisse jedes Szenarios angegeben. Auffällig dabei ist die unterschiedliche Verteilung welche Szenarien vergleichsweise gut gelöst werden. In Tabelle 5.10 ist zu erkennen, dass ohne Extraparameter das $CVE-2014-0160$ Szenario gut gelöst wird. Diese ist wie in Tabelle 5.5 beschrieben im Schnitt eines der am schlechtesten gelösten Szenarien. Hingegen werden die vermeintlich einfachen Szenarien wie das $CVE-2018-5418$ nicht gut gelöst. Im Vergleich werden mit Extraparametern das Bruteforce, $CVE-2012-2122$ und das $CVE-2014-0160$ Szenario nicht gut gelöst und alle anderen haben eine sehr hohe DR bei einer geringen Anzahl an Fehlalarmen.

Mit Extraparameter - pro Szenario				
Szenario	$\overline{FP}$	Δ	$\overline{DR}$	Δ
Bruteforce_CWE – 307	25.82	3.73	0.30	0.01
CVE – 2012 – 2122	11.82	2.19	0.03	0.01
CVE – 2014 – 0160	5.95	2.70	0.01	0.07
CVE – 2017 – 7529	1.25	0.67	0.90	0.08
CVE – 2018 – 3760	8.82	8.18	1.00	0.00
CVE – 2019 – 5418	7.00	5.17	0.58	0.00
EPS_CWE – 434	18.56	3.76	0.97	0.03
PHP_CWE – 434	13.22	3.13	0.91	0.00
SQL_Injection_CWE – 89	25.51	15.70	0.99	0.04
ZipSlip	11.64	23.86	0.19	0.04

Tabelle 5.8: Darstellung der durchschnittlichen DR und FPs für alle Szenarien mit Konfigurationen ohne Extraparametern. Dabei muss bei ZipSlip beachtet werden, dass weitaus weniger Konfigurationen in den Durchschnitt eingeflossen sind.

Vergleich beste Konfiguration nach DR					
Szenario	FP	DR	vs.	FPs	DR
Bruteforce CWE-307	60	0.94	x	29	0.94
CVE-2012-2122	8	0.01	x	18	0.99
CVE-2014-0160	3	0.00	x	13	0.03
CVE-2017-7529	1	0.99	x	0	0.99
CVE-2018-3760	11	1.00	x	4	1.00
CVE-2019-5418	32	1.00	x	7	1.00
EPS CWE-434	14	1.00	x	29	1.00
PHP CWE-434	4	0.96	x	24	.1.00
SQL Injection CWE-89	1	0.46	x	82	1.00
ZIP Slip	xxx	xxx	x	xxx	xxx

Tabelle 5.9: Vergleich bester Ergebnisse. Höchste DR links mit folgenden Parametern: $n = 6, e = 8, rv = 0, \text{TCF} = 0, time = 0$. Höchste DR rechts mit folgenden Parametern: $n = 6, e = 8, rv = 1, \text{TCF} = 1, time = 1$. Eine Verschlechterung wird mit Rot hervorgehoben, ansonsten Blau.

Vergleich beste Konfiguration nach FP					
Szenario	FP	DR	vs	FPs	DR
Bruteforce CWE-307	15	1.00	x	10	0.07
CVE-2012-2122	4	0.03	x	10	0.02
CVE-2014-0160	1	0.99	x	5	0.00
CVE-2017-7529	0	0.05	x	3	0.99
CVE-2018-3760	9	0.01	x	0	1.00
CVE-2019-5418	12	1.00	x	3	0.99
EPS CWE-434	9	1.00	x	9	1.00
SQL Injection CWE-89	1	1.00	x	1	1.00
PHP CWE-434	1	0.00	x	1	1.00
ZIP Slip	xxx	xxx	x	xxx	xxx

Tabelle 5.10: Vergleich bester Ergebnisse. Niedrigste FPs links mit folgenden Parametern: $n = 10, e = 10, rv = 0, \text{TCF} = 0, time = 0$. Niedrigste FPs rechts mit folgenden Parametern: $n = 6, e = 8, rv = 0, \text{TCF} = 0, time = 1$. Eine Verschlechterung wird mit Rot hervorgehoben, ansonsten Blau.

5.5 VERGLEICH ANDERER ARBEITEN

Um die Ergebnisse einordnen zu können soll im Folgenden ein Vergleich zu anderen Arbeiten die den LID-DS verwenden gezogen werden. Dafür steht der STIDE von Grimmer et al. [25] zur Verfügung. Der Vergleich der beiden besten LSTM Konfigurationen und dem STIDE Algorithmus wird in Tabelle 5.11 dargestellt. Dabei wird im STIDE ein Sliding Window (SW) verwendet. Ein Anomaliescore mit SW beinhaltet in diesem speziellen Fall den Mittelwert von 1000 Anomaliewerten der N-Gramme. Die DR des STIDE von 0.986 wird nicht erreicht, allerdings werden die FPs um ca. 40% von durchschnittlich 61.5 auf 22.89 reduziert.

Ergebnisse für LSTM mit Extraparameter								
Algorithmus	n	e	SW	rv	TCF	$time$	$\overline{\text{FP}}$	$\overline{\text{DR}}$
STIDE	5	int	1000	0	0	0	61.50	0.986
best DR LSTM	6	8	1	1	1	1	22.89	0.88
best FP LSTM	6	8	1	0	0	1	4.66	0.67

Tabelle 5.11: Vergleich mit nach Grimmer et al. [25] besten STIDE Konfiguration mit den besten Ergebnissen dieser Arbeit.

Um eine weitere Einschätzung zu bekommen welche Konfigurationen am besten funktionieren werden in [25] fünf Anforderungslevel

für Fehlalarme erstellt. Diese Anforderungen lauten für: Level 1 FPs<40, Level 2 FPs<20, Level 3 FPs<10, Level 4 FPs<5, Level 5 FPs<2.5. In Tabelle 5.12 wird für jedes Level die beste Konfiguration dieser Arbeit, links, mit der Arbeit von [25], rechts, verglichen. Dabei werden auch die noch andere Algorithmen wie Bag of System Calls (BOSC), Auto Encoder (AE) und Multilayer Perceptron (MLP) untersucht. Insgesamt ist erkennbar, dass der LSTM-Ansatz auf allen Leveln bis auf Level 5 ähnlich gut abschneidet. Es gab keine Konfiguration mit welcher weniger als durchschnittlich 2.5 Fehler pro Szenario erreicht werden konnten. Was daraus zu schließen ist, soll im nächste Kapitel besprochen werden. Wieder erzielen die in den vorigen Kapiteln speziell erwähnten Konfigurationen $n = 10$ und $e = 4$ bzw. $n = 6$ und $e = 8$ die besten Ergebnisse.

Ergebnisse für LSTM mit Extraparameter									
lvl	Algorithmus	n	e	SW	rv	TCF	$time$	$\overline{\text{FP}}$	$\overline{\text{DR}}$
1	STIDE	5	int	100	0	0	0	23.7	0.983
2	BOSC	3	int	10	0	0	0	18.9	0.982
3	MLP	7	OHE	1	0	0	0	4.2	0.788
4	MLP	7	OHE	1	0	0	0	4.2	0.788
5	AE	7	W2V	1	0	0	0	2.2	0.622
1	LSTM	6	8	1	1	1	1	22.89	0.88
2	LSTM	10	4	1	1	1	0	14	0.76
3	LSTM	10	4	1	0	1	0	5	0.74
4	LSTM	6	8	1	0	0	1	4.66	0.67
5	x	x	x	x	x	x	x	x	x

Tabelle 5.12: Vergleich weiterer Algorithmen von Grimmer et al. [25]. Dabei werden wie in [25] die Ergebnisse nach verschiedenen Leveln von FP Grenzwerten eingeteilt. Level 1 FPs<40,Level 2 FPs<20, Level 3 FPs<10, Level 4 FPs<5, Level 5 FPs<1.5. int steht dabei für das Integerembedding bei dem die System Call Namen in Integerwerte übersetzt werden.

6 FOLGERUNGEN

Nachdem die Ergebnisse der Versuchsreihen vorgestellt wurden, folgt nun eine Bewertung der Ergebnisse. Dafür wird in Abschnitt 6.1 zunächst nur die Ergebnisse des LSTM Ansatz besprochen. Im Anschluß wird in Abschnitt 6.2 die Folgerungen aus den Experimenten mit Extraparametern erörtert. Schließlich werden in Abschnitt 6.3 die Ergebnisse des Vergleichs anderer Arbeiten mit demselben Datensatz besprochen

6.1 LSTM ANSATZ

Die Ergebnisse beim Einsatz von LSTMs in anomaliebasierten HIDSs zeigen zwei Dinge sehr deutlich. Das LSTM kann ein Sprachmodell der System Calls erstellen und damit erfolgreich System Calls vorhersagen. Dies haben auch andere Arbeiten versucht zu zeigen, doch wie zuvor in Abschnitt 3.3 beschrieben, erfolgten die Auswertungen bisher auf veralteten Datensätzen mit praxisfernen Metriken und oft ohne detaillierte Beschreibung des Aufbaus. Aufgrund dessen kann leider kein direkter Vergleich der Architekturen der LSTMs gezogen werden.

Die Ergebnisse der Experimente konnten für die N-Gram und Embidding Größe keinen klaren Trend abbilden. So können bei einer N-Gram Größe von $n = 10$ und einer Embedding Größe von $e = 4$ ähnlich gute Ergebnisse erzielt werden wie bei $n = 2$ und $e = 4$. Allerdings ist der Berechnungsaufwand gerade bei großem n sehr hoch. Speziell bei sehr großen Szenarien hat dies in dieser Arbeit zu Ressourcenengpässen geführt. So konnten, wie zuvor beschrieben einige Konfigurationen nicht auf dem größten Szenario des LID-DSs berechnet werden. Ein weiterer Nachteil der Anomalieerkennung durch ein LSTM besteht darin, dass ein GPU für eine effiziente Berechnung nötig ist. Hinzu kommt, dass mit einem LSTM eine Parallelisierung nur geringfügigen Einfluss auf die Berechnungszeit hat. Grundproblem dabei ist die Rekurrenz des Netzes. Die Ausgabe des letzten Zeitschrittes muss bekannt sein um die Ausgabe des aktuellen Zeitschrittes berechnen zu können. So kann daraus gefolgert werden, dass der reine Einsatz von LSTMs nur auf der Sequenz der System Call Namen keine Verbesserung im Vergleich mit zum Beispiel der STIDE Implementierung von Grimmer et al. [25] erreicht werden kann. Aus den vorliegenden Ergebnissen kann der Einsatz von LSTMs in der beschriebenen Architektur, ohne die Verwendung von zusätzlichen Parametern keine Verbesserung erzielen. Daraus lässt sich schließen, dass LSTMs ohne die Verwendung zusätzlicher Parameter aufgrund des mit der Berechnung

verbundenen Mehraufwands keine nennenswerten Vorteile bringen konnte.

Wie dies unter der Verwendung von Extraparametern aussieht wird im Folgenden betrachtet.

6.2 EINSATZ VON EXTRAPARAMETERN

Wie in Abschnitt 5.4 beschrieben, kann eine deutliche Verbesserung der DR sowie der FP-Rate mit dem Einsatz verschiedener Extraparameter erzielt werden. So sind 9 unter den besten 10 Ergebnissen nach DR und 7 nach FP-Rate, Konfigurationen mit mindestens einem Extraparameter. Im Vergleich zu den Ergebnissen ohne Extraparametern ist hier ein Trend zu größeren N-Grammen erkennbar. Es ist nur noch insgesamt eine Konfiguration unter den besten zehn Ergebnissen nach DR und FP mit einer N-Gram Größe von 2. Im Vergleich zu insgesamt sieben der besten zehn bei Konfigurationen ohne Verwendung von Extraparametern. Daraus lässt sich schließen, dass bei komplexeren Eingaben größere N-Gramme bessere Ergebnisse liefern. Auffällig sind auch die Verteilungen der Embedding Größen. Dabei ist keine Abweichung der erfolgreichen Größen zwischen den Konfigurationen mit und ohne zusätzlichen Parametern zu erwarten gewesen. Doch so ist in Tabelle 5.2 zu erkennen, dass drei der Konfigurationen eine Größe $e = 10$ und drei $e = 6$ verwenden, während diese Größen mit zusätzlichen Parametern nicht mehr unter den besten zehn Konfigurationen vorkommen. Dort sind es entweder $e = 4$ oder $e = 8$. Da nach FP in Tabelle 5.7 unter den besten zehn Konfigurationen wiederrum für fünf Konfigurationen $e = 6$ gilt, kann keine beste Größe angegeben werden. Es kann aber zumindest vermutet werden, dass eine größere Einbettung nicht unbedingt mehr Informationsgehalt bedeutet. Dies kann jedoch nur definitiv festgestellt werden, wenn auch größere LSTM-Architekturen in die Experimente einbezogen werden, ebenso wie unterschiedliche Kontextgrößen des W2V-Verfahrens.

drei nach DR, vier nach FP

Abschießend lässt sich sagen, dass sich unter Verwendung der in dieser Arbeit entwickelten zusätzlichen Parameter die Kombinationen von $n = 10$ und $e = 4$ sowie $n = 6$ und $e = 8$ für die N-Gramm Größe und Embedding Größe, nach DR sowie FP als sehr erfolgreich erwiesen haben. Zusätzlich konnte zwar keine eindeutig beste Konfiguration für die Kombination der zusätzlichen Parameter gefunden werden, aber fast alle Kombinationen konnten eine Verbesserung der Ergebnisse erzielen

6.3 VERGLEICH ANDERER ARBEITEN

Der Vergleich zu anderen Arbeiten, die ebenfalls LSTMs verwenden, ist wie bereits an verschiedenen Stellen erwähnt schwierig, da unter anderem andere Datensätze verwendet wurden. Dennoch kann

ein Vergleich zu der Arbeit von Grimmer et al. [25] gezogen werden. Speziell wurde hierfür die beste Konfiguration verglichen, sowie das von Grimmer et al. eingeführte 5 Level Modell. Im Direktvergleich der besten Konfiguration in Tabelle 5.11 ist signifikanter Rückgang der FPs zu erkennen, in Tabelle 5.12 ist dies nicht mehr erkennbar. Das LSTM kann mit einer Auswahl an Algorithmen auf allen Leveln bis auf Level 5 mithalten. Dabei verwenden diese Algorithmen keine Extraparameter. Der entworfene anomaliebasierte HIDS Ansatz unter Verwendung eines LSTMs kann somit mit den in der Forschung verwendeten Algorithmen mithalten. Wie die erlangten Erkenntnisse in Zukunft weiter ihren Einsatz finden können, soll im folgenden Kapitel untersucht werden.

7 ZUSAMMENFASSUNG UND AUSBLICK

IT-Sicherheit ist in der Praxis oft einen Schritt hinter den Angriffen zurück. Dies liegt daran, dass Verteidigungssysteme üblicherweise versuchen Angriffe anhand ihrer Signaturen zu erkennen. Sind die Angriffe noch nicht bekannt, so können auch noch keine Signaturen dafür existieren. Das sorgt dafür, dass sie auch nicht erkannt werden können. Gleiches gilt für die meisten Abwandlungen von bereits bekannten Angriffen. Um den Angriffen nicht immer hinterher zu sein, wird versucht mit anomaliebasierten Verteidigungssystemen auch unbekannte Angriffe zu erkennen. Statt eine Liste von Angriffen ständig up-to-date halten zu müssen, wird versucht Muster des Normalverhaltens zu erlernen. Eine Abweichung des Normalverhaltens wird dann als Angriff gewertet und ermöglicht es zeitnahe weitere Schritte einzuleiten.

7.1 ZUSAMMENFASSUNG

Um die Sicherheit von Computersystemen zu gewährleisten werden häufig IDSs in die IT-Infrastruktur eingepflegt. Häufig werden NIDS verwendet, welche Netzwerkdaten untersuchen und damit mehrere Hosts gleichzeitig überwachen. Um Angriffe auf einzelnen Hosts zu erkennen, reichen diese meist nicht aus. HIDSs können IT-Sicherheitssysteme um diese Fähigkeit erweitern. Seit mehr als 20 Jahren werden HIDSs in der Forschung behandelt, wobei sich das Untersuchen von System Call Daten als vielversprechend herausgestellt hat. Seit dem Beginn der Forschung wurden verschiedene Algorithmen sowie Datensätze auf welchen diese ausgewertet werden können präsentiert. Da aber viele Datensätze Unzulänglichkeiten aufweisen, leidet die Vergleichbarkeit der Ergebnisse stark. Der auf modernen Systemen aufgezeichnete LID-DS beseitigt einige dieser Unzulänglichkeiten. Diese Arbeit beschäftigt sich speziell mit HIDSs welche Anomalien in Sytem Call Streams des LID-DS erkennen. In der ersten Iteration wird die Sequenz der System Calls betrachtet, wobei dafür die Sequenz der System Call Namen verwendet wird. Dabei bietet es sich an von etablierten Verfahren der Mustererkennung zu profitieren. Ein Bereich der Mustererkennung welcher großen Fortschritt erfahren hat, ist die NLP. Es stellt sich also die Frage, ob bestimmte Verfahren, die in der NLP eingesetzt werden, auch im Bereich der HIDS erfolgreich sind. Speziell die LSTMs zeigten sich in der Sprachverarbeitung sowie in der Analyse von Zeitreihendaten als effektiv. LSTMs haben gegenüber vielen anderen Algorithmen die in der Forschung bisher eingesetzt wurden einen Vorteil. Sie be-

ziehen zuvor gesehene Eingaben in die Klassifizierung der aktuellen Eingabe mit ein, kennen also den Kontext der aktuellen Eingabe. Es wurde in dieser Arbeit gezeigt, dass sich die Fortschritte der NLP in Form von LSTMs auf die anomaliebasierte HIDS übertragen lassen. Um die System Call Namen sinnvoll für das LSTM darzustellen, wird auf ein ebenfalls aus der NLP stammendes Verfahren zurückgegriffen. Das W2V-Verfahren stellt nicht nur ein Lookup-Table für System Calls dar, sondern kodiert auch Kontextinformationen der System Calls. Um die Ergebnisse des entworfenen Algorithmus weiter zu verbessern, werden neue Verfahren zur Kodierung zusätzlicher Informationen der System Calls vorgestellt. Zum einen werden dafür die Rückgabewerte bestimmter lesender und schreibender System Calls verwendet. Dies wird erreicht in dem für jede Kategorie, zum Beispiel lesende System Calls, das Maximum der gelesenen Bytes aus den Trainingsdaten ermittelt wird. Mit dem ermittelten Maximum werden dann die Werte aus den Testdaten normalisiert. Und zum anderen werden auf gleiche Weise die zeitlichen Abstände zwischen zwei System Calls innerhalb eines Threads normalisiert und dienen so als Eingabe für das LSTM. In einem weiteren Feature werden die von Grimmer et al. [25] präsentierten *Thread aware* N-Gramme für Algorithmen mit Kontextwissen erweitert. Da es für RNNs von Bedeutung ist was für eine Eingabe im letzten Zeitschritt gesehen wurde, soll dem LSTM ein Wechsel des Threads und damit ein Wechsel des Kontextes mitgeteilt werden. Dieses Feature wird also nur Algorithmen einen Vorteil bringen, welche den Kontext der Eingaben miteinbeziehen. Um zu überprüfen ob und mit welcher Konfiguration sich die beschriebenen Verfahren als erfolgreich erweisen, werden ausführliche Experimente auf dem LID-DS durchgeführt. Die Auswertung der Ergebnisse bringt zwei Probleme mit sich. Erstens sind einige bekannte und verbreitete Metriken zur Auswertung von neuronalen Netzen mit dem vorhandenen Datensatz nicht umzusetzen. Deswegen werden in der Arbeit FP-Rate und DR verwendet. Diese werden auch in anderen Arbeiten auf diesem Datensatz angewandt und bieten zusätzlich einen größere Praxisnähe. Zweitens ist die Auswahl der besten Konfigurationen auf der Grundlage von zwei Kriterien ohne Gewichtung der Kriterien nicht möglich. Es werden verschiedene Eingabegrößen und Konfigurationen der Extraparameter untersucht. Eine klare Tendenz für optimale Eingabegrößen oder Konfiguration sind dabei nicht zu erkennen. Dennoch kann gezeigt werden, dass die LSTMs konkurrenzfähige, aber nicht signifikant bessere Ergebnisse liefern, insbesondere wenn die zusätzlichen Parameter hinzugefügt werden. Der größte Nachteil des präsentierten HIDS Verfahren besteht in der aufwendigen Berechnung der LSTMs. Es zeigt sich also, dass sich der Erfolg der LSTMs in der NLP auf die Erkennung von Anomalien in der Cyber-Sicherheit übertragen lässt, dabei aber auch neue Schwierigkeiten mit sich bringt. Auch die

Zunahme von Extraparametern der System Calls erzielt eine klare Verbesserung.

7.2 AUSBLICK

Wie beschrieben kann der LSTM Ansatz zwar keine offensichtliche Verbesserung der Ergebnisse erreichen, doch zeigt er, dass eventuell auch weitere Lösungsansätze aus der NLP auf die Anomalieerkennung übertragen werden können. Eine Möglichkeit diese Ergebnisse zu verbessern, könnte in Konfigurationen bestehen, die noch nicht getesteten wurden. So könnten die Auswirkungen von größeren LSTM-Architekturen auf die Ergebnisse untersucht werden. Dies wird aber die Berechnungszeiten noch weiter erhöhen. Ein Punkt gegen die weitere Untersuchung von LSTMs in dieser Domäne besteht darin, dass Transformer sich bereits bei den meisten Übersetzungs- und Sprachgenerierungsprogrammen gegenüber den LSTMs durchgesetzt haben. Denn sie ermöglichen wie die LSTMs eine Verarbeitung der System Calls mit Einbindung des Kontexts. Gleichzeitig kann die Berechnung der Eingaben in Transformern besser parallelisiert werden. Dieser Ansatz könnte die Vorteile der LSTMs ausspielen und einen erheblichen Geschwindigkeitsvorteil bringen.

Die Erkenntnisse der Arbeit bezüglich des Entwurfs neuer Kodierungen für Extraparameter der System Calls könnte sich hingegen direkt auf weitere Arbeiten auswirken. So kann vielleicht der Einsatz der Extraparameter in Zukunft eine Verbesserung der Ergebnisse des STIDE oder anderen Algorithmen erzielen. Insbesondere können weitere kontextsensitive Algorithmen durch die Erweiterung der *Thread Aware* N-Gramme profitieren. Aber die präsentierten zusätzlichen Parameter sind ebenfalls noch erweiterbar. So können noch weitere Normalisierungen von System Call Rückgabewerten erfolgen. Zum Beispiel der System Call *send* wurde bisher noch nicht mit einbezogen. Auch weitere Gruppierungen von System Calls oder eventuell das Auflösen von Gruppierungen sollte in weiteren Arbeiten untersucht werden. Es kann zwar davon ausgegangen werden, dass sich die Erfolge der Extraparameter im Zusammenhang mit den LSTM Ansatz nicht eins zu eins auf Ansätze wie den STIDE Algorithmus übertragen lassen, dennoch wurden in dieser Arbeit neue Kodierungsansätze präsentiert, die zur breiteren Nutzung von zusätzlichen Parametern beiträgt.

[1] Adamu I Abubakar, Haruna Chiroma, Sanah Abdullahi Muaz und Libabatu Baballe Ila. „A review of the advances in cyber security benchmark datasets for evaluating data-driven based intrusion detection systems“. In: *Procedia Computer Science* 62 (2015), S. 221–227.

[2] Bilal Zahran Aufa, Suyanto Suyanto und Anditya Arifianto. „Hyperparameter setting of LSTM-based language model using grey wolf optimizer“. In: *2020 International Conference on Data Science and Its Applications (ICoDSA)*. IEEE. 2020, S. 1–5.

[3] V Kishore Ayyadevara. „Word2vec“. In: *Pro Machine Learning Algorithms*. Springer, 2018, S. 167–178.

[4] M. H. Bhuyan, D. K. Bhattacharyya und J. K. Kalita. „Network Anomaly Detection: Methods, Systems and Tools“. In: *IEEE Communications Surveys Tutorials* 16.1 (2014), S. 303–336.

[5] C.M. Bishop, P.N.C.C.M. Bishop, G. Hinton und Oxford University Press. *Neural Networks for Pattern Recognition*. Advanced Texts in Econometrics. Clarendon Press, 1995. ISBN: 9780198538646.

[6] Lydia Bouzar-Benlabiod, Lila Méziani, Stuart H Rubin, Kahina Belaidi und Nour Elhouda Haddar. „Variational encoder-decoder recurrent neural network (VED-RNN) for anomaly prediction in a host environment“. In: *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*. IEEE. 2019, S. 75–82.

[7] Rory Bray, Daniel Cid und Andrew Hay. *OSSEC host-based intrusion detection guide*. Syngress, 2008.

[8] Thomas M Breuel. „Benchmarking of LSTM networks“. In: *arXiv preprint arXiv:1508.02774* (2015).

[9] Robert A Bridges, Tarrah R Glass-Vanderlan, Michael D Iannacone, Maria S Vincent und Qian Chen. „A survey of intrusion detection systems leveraging host data“. In: *ACM Computing Surveys (CSUR)* 52.6 (2019), S. 1–35.

[10] Varun Chandola, Arindam Banerjee und Vipin Kumar. „Anomaly Detection: A Survey“. In: *ACM Comput. Surv.* 41.3 (Juli 2009). ISSN: 0360-0300.

[11] Ashima Chawla, Paul Jacob, Brian Lee und Sheila Fallon. „Bidirectional LSTM autoencoder for sequence based anomaly detection in cyber security.“ In: *International Journal of Simulation–Systems, Science & Technology* (2019).

[12] *Cost of a Data Breach. A view from the cloud 2021*. IBM Security, 2021.

[13] Gideon Creech und Jiankun Hu. „Generation of a new IDS test dataset: Time to retire the KDD collection“. In: *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2013, S. 4487–4492.

[14] Gideon Creech und Jiankun Hu. „A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns“. In: *IEEE Transactions on Computers* 63.4 (2014), S. 807–819.

[15] Roberto Di Pietro und Luigi V Mancini. *Intrusion detection systems*. Bd. 38. Springer Science & Business Media, 2008.

[16] Michael Dymshits, Benjamin Myara und David Tolpin. „Process monitoring on sequences of system call count vectors“. In: *Proceedings - International Carnahan Conference on Security Technology* 2017-Octob (2017), S. 1–5.

[17] Francis Ysidro Edgeworth. „Xli. on discordant observations“. In: *The london, edinburgh, and dublin philosophical magazine and journal of science* 23.143 (1887), S. 364–375.

[18] Vegard Engen. „Machine learning for network based intrusion detection: an investigation into discrepancies in findings with the KDD cup'99 data set and multi-objective evolution of neural network classifier ensembles from imbalanced data.“ Diss. Bournemouth University, 2010.

[19] S. Forrest und University of New Mexico. *University of New Mexico (UNM) Intrusion Detection Dataset*. URL: `https://www.cs.unm.edu/~immsec/systemcalls.htm` (besucht am 11. 04. 2022).

[20] S. Forrest, S. A. Hofmeyr, A. Somayaji und T. A. Longstaff. „A sense of self for Unix processes“. In: *Proceedings 1996 IEEE Symposium on Security and Privacy*. 1996, S. 120–128.

[21] Peter B Galvin, Greg Gagne, Abraham Silberschatz u. a. *Operating system concepts*. Bd. 10. John Wiley & Sons, 2003.

[22] Felix A. Gers, Jürgen A. Schmidhuber und Fred A. Cummins. „Learning to Forget: Continual Prediction with LSTM“. In: *Neural Comput.* 12.10 (Okt. 2000), S. 2451–2471. ISSN: 0899-7667.

[23] Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean und Larry Heck. „Contextual lstm (clstm) models for large scale nlp tasks“. In: *arXiv preprint arXiv:1602.06291* (2016).

[24] Markus Goldstein und Seiichi Uchida. „A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data“. In: *PLoS ONE* 11.4 (2016), S. 1–31. ISSN: 19326203.

[25] Martin Grimmer, Tim Kaelble und Erhard Rahm. „Improving Host-Based Intrusion Detection Using Thread Information“. In: *International Symposium on Emerging Information Security and Applications*. Springer. 2021, S. 159–177.

[26] Martin Grimmer und Martin Max Röhling. *Leipzig Intrusion Detection Data Set*. URL: https://www.exploids.de/lid-ds/ (besucht am 04. 11. 2022).

[27] Martin Grimmer, Martin Max Röhling, Matthias Kricke, Bogdan Franczyk und Erhard Rahm. „Intrusion Detection on System Call Graphs“. In: *Sicherheit in vernetzten Systemen* (2018), S. 1–18.

[28] Martin Grimmer, Martin Max Röhling, Dennis Kreusel und Simon Ganz. „A modern and sophisticated host based intrusion detection data set“. In: *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung* (2019), S. 135–145.

[29] Frank E. Grubbs. „Procedures for Detecting Outlying Observations in Samples“. In: *Technometrics* 11.1 (Feb. 1969), S. 1–21.

[30] Waqas Haider, Jiankun Hu, Jill Slay, Benjamin P Turnbull und Yi Xie. „Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling“. In: *Journal of Network and Computer Applications* 87 (2017), S. 185–192.

[31] S. He, J. Zhu, P. He und M. R. Lyu. „Experience Report: System Log Analysis for Anomaly Detection“. In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, S. 207–218.

[32] Sepp Hochreiter. „The vanishing gradient problem during learning recurrent neural nets and problem solutions“. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), S. 107–116.

[33] Sepp Hochreiter und Jürgen Schmidhuber. „LSTM Can Solve Hard Long Time Lag Problems“. In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. NIPS'96. MIT Press, 1996.

[34] Steven A. Hofmeyr, Stephanie Forrest und Anil Somayaji. „Intrusion Detection Using Sequences of System Calls“. In: *J. Comput. Secur.* 6 (1998), S. 151–180.

[35] Kishan G. Mehrotra Chilukuri K. Mohan HuaMing Huang. *Anomaly Detection Algorithms and Principles*. 2017, S. 1–229. ISBN: 9783319675244.

[36] Bundesamt für Sicherheit in der Informationstechnik. *Die Lage der IT-Sicherheit in Deutschland 2021*. URL: https://www.bsi.bund.de/DE/Service-Navi/Publikationen/Lagebericht/lagebericht_node.html.

[37] Shijoe Jose, D Malathi, Bharath Reddy und Dorathi Jayaseeli. „A survey on anomaly based host intrusion detection system“. In: *Journal of Physics: Conference Series*. Bd. 1000. 1. IOP Publishing. 2018, S. 12049.

[38] Dae-Ki Kang, Doug Fuller und Vasant Honavar. „Learning classifiers for misuse and anomaly detection using a bag of system calls representation“. In: *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. IEEE. 2005, S. 118–125.

[39] Ansam Khraisat, Iqbal Gondal, Peter Vamplew und Joarder Kamruzzaman. „Survey of intrusion detection systems: techniques, datasets and challenges“. In: *Cybersecurity* 2.1 (2019), S. 1–22.

[40] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek und Sungroh Yoon. „LSTM-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems“. In: *arXiv preprint arXiv:1611.01726* (2016).

[41] Oualid Koucham, Tajjeeddine Rachidi und Nasser Assem. „Host intrusion detection using system call argument-based clustering combined with Bayesian classification“. In: *IntelliSys 2015 - Proceedings of 2015 SAI Intelligent Systems Conference* (2015), S. 1010–1016.

[42] Christopher Kruegel, Darren Mutz, Fredrik Valeur und Giovanni Vigna. „On the detection of anomalous system call arguments“. In: *European Symposium on Research in Computer Security*. Springer. 2003, S. 326–343.

[43] Vipin Kumar. „Parallel and distributed computing for cybersecurity“. In: *IEEE Distributed Systems Online* 6.10 (Nov. 2005).

[44] Wenke Lee, S.J. Stolfo und P.K. Chan. „Learning patterns from unix process execution traces for intrusion detection“. In: *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management* (1997), S. 50–56.

[45] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin und Kuang-Yuan Tung. „Intrusion detection system: A comprehensive review“. In: *Journal of Network and Computer Applications* 36.1 (2013), S. 16–24.

[46] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba und Kumar Das. „1999 DARPA off-line intrusion detection evaluation“. In: *Computer Networks* 34.4 (2000), S. 579–595. ISSN: 13891286.

[47] Patrick Luckett, J Todd McDonald und Joel Dawson. „Neural network analysis of system call timing for rootkit detection“. In: *2016 Cybersecurity Symposium (CYBERSEC)*. IEEE. 2016, S. 1–6.

[48] Lincoln Laboratory MIT. *DARPA Intrusion Detection Evaluation Data Set*. 1998-2000. URL: https://www.ll.mit.edu/r-d/datasets (besucht am 11. 04. 2022).

[49] F. Maggi, M. Matteucci und S. Zanero. „Detecting Intrusions through System Call Sequence and Argument Analysis". In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (2010), S. 381–395.

[50] John Mchugh. „Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory". In: *ACM Transactions on Information and System Security* 3.4 (2000), S. 262–294. ISSN: 15577406.

[51] Chad R Meiners, Jignesh Patel, Eric Norige, Alex X Liu und Eric Torng. „Fast regular expression matching using small TCAM". In: *IEEE/Acm Transactions On Networking* 22.1 (2013), S. 94–109.

[52] Patricia Melin, Julio Cesar Monica, Daniela Sanchez und Oscar Castillo. „Multiple Ensemble Neural Network Models with Fuzzy Response Aggregation for Predicting COVID-19 Time Series: The Case of Mexico". In: *Healthcare* 8.2 (2020). ISSN: 2227-9032.

[53] Daniel Merkle und Martin Middendorf. „Ant colony optimization with global pheromone evaluation for scheduling a single machine". In: *Applied Intelligence* 18.1 (2003), S. 105–111.

[54] Andries Brouwer Michael Kerrisk Stepan Kasal. *syscalls(2) — Linux manual page*. URL: https://man7.org/linux/man-pages/man2/syscalls.2.html.

[55] Tomas Mikolov, Kai Chen, Greg Corrado und Jeffrey Dean. „Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[56] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel und Muttukrishnan Rajarajan. „A survey of intrusion detection techniques in cloud". In: *Journal of network and computer applications* 36.1 (2013), S. 42–57.

[57] Rajat Moona. *Assembly Language Peogramming in GNU/LINUS for IA32 Architectures*. PHI Learning Pvt. Ltd., Jan. 2009. ISBN: 9788120331563.

[58] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj und Mario Couture. „A host-based anomaly detection approach by representing system calls as states of kernel modules". In: *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 2013, S. 431–440.

[59] Darren Mutz, Fredrik Valeur, Giovanni Vigna und Christopher Kruegel. „Anomalous system call detection". In: *ACM Transactions on Information and System Security (TISSEC)* 9.1 (2006), S. 61–93.

[60] Zijian Niu, Ke Yu und Xiaofei Wu. „LSTM-Based VAE-GAN for Time-Series Anomaly Detection“. In: *Sensors* 20.13 (Juli 2020), S. 3738. ISSN: 1424-8220.

[61] Christopher Olah. *Understanding LSTM Networks*. Aug. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 01. 06. 2021).

[62] *PYTORCH. An open source machine learning framework that accelerates the path from research prototyping to production deployment*. URL: https://pytorch.org/ (besucht am 17. 03. 2022).

[63] DaeKyeong Park, KyungJoon Ryu, DongIl Shin, DongKyoo Shin, JeongChan Park und JinGoog Kim. „A Comparative Study of Machine Learning Algorithms Using LID-DS DataSet“. In: *KIPS Transactions on Software and Data Engineering* 10.3 (2021), S. 91–98.

[64] Daekyeong Park, Sangsoo Kim, Hyukjin Kwon, Dongil Shin und Dongkyoo Shin. „Host-Based Intrusion Detection Model Using Siamese Network“. In: *IEEE Access* 9 (2021), S. 76614–76623.

[65] Animesh Patcha und Jung-Min Park. „An overview of anomaly detection techniques: Existing solutions and latest technological trends“. In: *Computer networks* 51.12 (2007), S. 3448–3470.

[66] Refat Khan Pathan, Munmun Biswas und Mayeen Uddin Khandaker. „Time series prediction of COVID-19 by mutation rate analysis using recurrent neural network-based LSTM model“. In: *Chaos, Solitons & Fractals* 138 (2020), S. 110018.

[67] Clifton Phua, Damminda Alahakoon und Vincent Lee. „Minority report in fraud detection: classification of skewed data“. In: *Acm sigkdd explorations newsletter* 6.1 (2004), S. 50–59.

[68] M.T. Pilehvar und J. Camacho-Collados. *Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2020. ISBN: 9781636390222.

[69] IBM X-Force Research. *X-Force Threat Intelligence Index 2022*. IBM Security, 2022.

[70] Anil Somayaji und Stephanie Forrest. „Automated Response Using {System-Call} Delay“. In: *9th USENIX Security Symposium (USENIX Security 00)*. 2000.

[71] Ching Y Suen. „N-gram statistics for natural language understanding and text processing“. In: *IEEE transactions on pattern analysis and machine intelligence* 2 (1979), S. 164–172.

[72] Shraddha Suratkar, Faruk Kazi, Rohan Gaikwad, Akshay Shete, Raj Kabra und Shantanu Khirsagar. „Multi Hidden Markov Models for Improved Anomaly Detection Using System Call Analysis“. In: *2019 IEEE Bombay Section Signature Conference (IBSSC)*. IEEE. 2019, S. 1–6.

[73] Zarrin Tasnim Sworna, Zahra Mousavi und Muhammad Ali Babar. „NLP Methods in Host-based Intrusion Detection Systems: A Systematic Review and Future Directions“. In: *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03.05, 2018, Woodstock, NY* 1.1 (2022), S. 1–35.

[74] Inc. Sysdig. *Seeing is Securing For containers, Kubernetes and cloud services*. URL: `https://sysdig.com/` (besucht am 11. 04. 2022).

[75] Kymie MC Tan, Kevin S Killourhy und Roy A Maxion. „Undermining an anomaly-based intrusion detection system using common exploits“. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2002, S. 54–73.

[76] Kymie MC Tan und Roy A Maxion. „"Why 6?"Defining the operational limits of stide, an anomaly-based intrusion detector“. In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE. 2002, S. 188–201.

[77] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu und Ali A. Ghorbani. „A detailed analysis of the KDD CUP 99 data set“. In: *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009* Cisda (2009), S. 1–6.

[78] Amjad M. Al Tobi und Ishbel Duncan. „KDD 1999 generation faults: a review and analysis“. In: *Journal of Cyber Security Technology* 2.3-4 (2018), S. 164–200.

[79] David Wagner und Paolo Soto. „Mimicry attacks on host-based intrusion detection systems“. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, S. 255–264.

[80] Peipei Wang, Xinqi Zheng, Gang Ai, Dongya Liu und Bangren Zhu. „Time series prediction for the epidemic trends of COVID-19 using the improved LSTM deep learning method: Case studies in Russia, Peru and Iran“. In: *Chaos, Solitons & Fractals* 140 (2020), S. 11021.

[81] Christina E. Warrender, Stephanie Forrest und Barak A. Pearlmutter. „Detecting intrusions using system calls: alternative data models“. In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)* (1999), S. 133–145.

[82] Gregory S Watson, David W Green, Lin Schwarzkopf, Xin Li, Bronwen W Cribb, Sverre Myhra und Jolanta A Watson. „A gecko skin micro/nano structure–A low adhesion, superhydrophobic, anti-wetting, self-cleaning, biocompatible, antibacterial surface“. In: *Acta biomaterialia* 21 (2015), S. 109–122.

[83] Sarah Wunderlich, Markus Ring, Dieter Landes und Andreas Hotho. „Comparison of system call representations for intrusion detection“. In: *International Joint Conference: 12th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th International Conference on EUropean Transnational Education (ICEUTE 2019)*. Springer. 2019, S. 14–24.

[84] Yong Yu, Xiaosheng Si, Changhua Hu und Jianxun Zhang. „A review of recurrent neural networks: LSTM cells and network architectures“. In: *Neural computation* 31.7 (2019), S. 1235–1270.

[85] Radim Řehůřek. *GENSIM, topic modelling for humans*. URL: `https://radimrehurek.com/gensim/` (besucht am 17. 03. 2022).