VAE-MAD-GAN FOR HIDS

TIM KAELBLE

ScaDS-AI
November 2020 –

ABSTRACT

Short summary of the contents in English. . . a great guide by Kent Beck how to write good abstracts can be found here:

https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache. . .

ACRONYMS

IDS Intrusion Detection System

HIDS Host-based Intrusion Detection System

NIDS Network Intrusion Detection System

SIDS Signature-based Intrusion Detection Systems

AIDS Anomaly-based Intrusion Detection Systems

ADFA-LD ADFA Linux Dataset

LSTM Long-Short-Term-Memory

RNN Rekurrente neuronale Netze

BSI Bundesamt fuer Sicherheit in der Informationstechnik

API Application Programming Interface

CVE Common Vulnerabilities and Exposures

CWE Common Weakness Enumeration

CNN Convolutional Neural Nets

NLP Natural Language Processing

ML Maschinelles Lernen

FP False Positive

FN False Negative

TN True Negative

TP True Positive

TIDE Time-Delay Embedding

STIDE Sequence Time-Delay Embedding

RIPPER Repeated Incremental Pruning to Produce Error Reduction

HMM Hidden Markov Model

DR Detektionsrate

OHE One-Hot-Encoding

W2V Word2Vec

LID-DS Leipzig Intrusion Detection Dataset

TCF Thread Change Flag

CONTENTS

1 EINFÜHRUNG

1.1 EINLEITUNG

Notes:

- solarwinds as introduction
- use advances of sequence detection from NLP
- NIDS vs. HIDS
- signature vs. anomaly based
- Forrest et al 1996 erstmals syscall traces
- low level interactions between program and kernel
- syscall traces dont stop execution contrary to debuggers
- tracing virtually every linux without modifying source code
- whole system behaviour visible in kernel

Angriffe auf Computersysteme werden frequenter. Die häufig verwendeten auf Signaturen basierenden Abwehrmechanismen reichen nicht aus um viele drohende Gefahren abzuwenden. Dies liegt hauptsächlich daran, dass weder Abwandlungen von bekannten Angriffen, noch unbekannte Angriffe erkannt werden können. Zusätzlich müssen die Signaturen für jeden Angriffsvektor einzeln eingefügt werden. Ein wesentlicher Vorteil liefert hier die Angriffserkennung über Anomalien. Im Gegensatz zu dem erwähnten signaturbasierten Ansatz, muss nicht jeder Angriff der abgewehrt werden soll bekannt sein. Stattdessen wird versucht ein Modell des zu erwartenden Normalverhalten des Systems zu erstellen. Mit dem erstellten Modell sollen dann, möglichst in Echtzeit, Abweichungen bzw. Anomalien des erwarteten Verhalten signalisiert werden.

vgl. Abschnitt 2.1.1

Die Bedeutsamkeit der Erkennung von bisher unbekannten Angriffe wird ebenfalls durch das Bundesamt fuer 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. [32].

Juni 2020 bis Mai 2021

vgl.??

Speziell werden in dieser Arbeit Host-based Intrusion Detection System (HIDS) verwendet, das sie gegenueber den Network Intrusion Detection System (NIDS) feingranularer sind und auch interne Attacken

erkennen koennen. Nun bieten verschiedene Systeme unterschiedliche Möglichkeiten das ihnen zugrunde liegende Verhalten zu beschreiben. Eine häufig verwendete Information für die Charakterisierung bieten zum Beispiel System-Logs [27].

In dieser Arbeit werden System-Calls verwendet. Sie bieten eine sehr abstrakte Betrachtung auf Betriebssystemebene. Programme auf einer Festplatte können meist erst Schaden anrichten, sobald sie ausgeführt werden. Dabei führen sie betriebssystemspezifische System-Calls aus, welche über verschiedene Tools wie zum Beispiel Sysdig [64] ausgelesen werden können. Die Schwierigkeit im Vergleich zu dem Untersuchen der Logs besteht darin, die großen Datenmengen zu bewältigen, welche schon bei kleineren Anwendungen anfallen. Die Probleme in der Verarbeitung von sehr großen Datenmengen konnten unter anderem durch die Verwendung selbst lernender Algorithmen erfolgreich angegangen werden. Im realen Einsatz solcher Verteidigungsmechanismen besteht eine weitere Schwierigkeit darin, dass das Intrusion Detection System (IDS) Zugriff auf den Kernel des zu überwachenden Systems benötigt. Diese wird in dieser Arbeit allerdings nicht behandelt, da lediglich die Algorithmen selbst, jedoch nicht die praktische Umsetzung in einem potentiellen Betrieb betrachtet wird.

In verschiedenen Arbeiten wurden bereits die Abfolge von System-Calls betrachtet, doch nur in wenigen Arbeiten werden auch die Parameter zur Anomalieerkennung verwendet. Eine der ersten Arbeiten von Forrest et al. [16] betrachtet lediglich die Sequenzen der System-Calls. Maggi et al. verwenden zusätzlich auch Parameter und verweisen in ihrer Arbeit [47] auf diverse verschiedene Ansätze. In dieser Arbeit soll versucht werden die Hinzunahme eines Parameters, wie zum Beispiel den Dateipfad (sofern vorhanden) bei schreibenden und lesenden Befehlen, mit Hinblick auf die Erkennungsquote des IDS zu untersuchen.

Nachdem definiert wurde welche Information untersucht wird, stellt sich zu Beginn der Entwicklung einer Anomalieerkennung die Frage, wie das Normalverhalten der Systeme erfasst werden soll. Abstrakt betrachtet werden bei der Untersuchung von System-Calls zeitvariante und potentiell multivariate Datenstreams betrachtet, sofern neben der eigentlichen Sequenz noch weitere Parameter betrachtet werden. Besonders erfolgreich haben sich dabei Long-Short-Term-Memory (LSTM) Netzwerke gezeigt. Sie haben den Vorteil auch Zusammenhänge mit größerer zeitlicher Verzögerung noch zu erkennen [29] und können in unterschiedlichsten Architekturen einen Nutzen bringen.

1.2 ZIELSETZUNG/FORSCHUNGSFRAGE

In dieser Arbeit sollen zwei Forschungsfragen verfolgt werden.

- Kann der Erfolg von LSTM-Netzwerken in verschiedenen Bereichen 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 in Frage?

Um diese Forschungsfragen angemessen behandeln zu können müssen zunächst Grundlagen aus verschiedenen Bereichen gelegt werden. Zum einen werden unterschiedliche Herangehensweisen zur Überwachung von Systemen betrachtet und erläutert wieso es für diese Anwendung sinnvoll ist eine Host-Based Intrusion Detection zu wählen. Speziell soll auch beschrieben werden, warum sich System-Calls zur Überwachung von Computersystemen eignen. Des Weiteren müssen Grundlagen für die in dem verwendeten Algorithmus verwendeten Techniken gelegt werden. Dazu gehören Hauptsächlich Grundlagen zu rekurrenten neuronalen Netzen (RNN) sowie die Erweiterungen der LSTM Netzwerke.

Ein großer Teil der Implementierungsarbeit jedoch wird die Vorverarbeitung der Daten darstellen. Diese soll mit der genaueren Untersuchung der Zusammensetzung der Techniken für den Algorithmus in einem weiteren Kapitel dargestellt werden. Nachdem die verwendete Software analysiert wurde, wird eine Auswertung auf dem LID-DS [41] Datensatz durchgeführt. Dieser bietet den Vorteil, dass in einer reproduzierbaren Art System Calls aufgenommen wurden. Des Weiteren werden zusätzlich die System Call Parameter, wie zum Beispiel die *Thread ID* zur Verfügung gestellt.

Im letzten Teil 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?

1.3 BEITRAG

2 GRUNDLAGEN

In den folgenden Abschnitten werden die Grundlagen betrachtet, welche für die Umsetzung eines IDS nötig sind. Zu Beginn soll in Abschnitt 2.1.1 geklärt werden welche Analysetechniken für die Überwachung von Systemen zur Verfügung stehen. In Abschnitt 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 Rekurrente neuronale Netze (RNN), sowie die Erweiterung der RNNs die Long-Short-Term-Memory (LSTM) neuronalen Netzen beschrieben.

2.1 INTRUSION DETECTION SYSTEM

Eine *Intrusion* ist eine unautorisierte Aktivität welche zu Missbrauch eines Systems führen kann. Sie sorgt für ein zum Normalverhalten abweichendes, anormales Verhalten und kann durch diese Abweichung einem System großen Schaden zufügen. Ein IDS überwacht Ressourcen, wie zum Beispiel Netzwerkdaten und versucht automatisch Missbrauch und anormales Verhalten zu identifizieren, um dadurch die Sicherheit des Systems gewährleisten zu können. [12]

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. [35, 42, 54]. 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 Datenbank verglichen. Es gibt diverse Möglichkeiten wie diese Signaturen

erstellt werden. In mehreren Arbeiten werden dafür zum Beispiel Netzwerk Pakete betrachtet [35] oder auch ein wenig ausgefeilter *State Machines* erstellt [49].

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. [12] 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 Anomaly-based Intrusion Detection Systems (AIDS) 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 [31]. 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.” [25]

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 [39]. Festzuhalten ist also, dass sie das Anomalien eines Normalverhaltens darstellen und damit interessant für Analysen sind. Genau dieser signifikante Unterschied des aktuellen Systemzustand zu einem entworfenen Modell (wohldefiniertes Normalverhalten) zu einem Zeitpunkt t wird dann als Anomalie eingestuft. Jede Anomalie gilt dann wiederrum als *Intrusion*. Zusammengefasst besteht also die Grundannahme dieser Methode darin, dass Intrusions von dem gelernten Normalverhalten des Systems unterschieden werden können.

AIDS werden im Einsatz in zwei Phasen aufgeteilt, die *Trainingsphase* sowie die *Testphase*. Im ersten Schritt der AIDS wird ein Modell des Normalverhaltens erstellt. Dieser Schritt wird in Maschinelles Lernen (ML)-basierte, Statistik-basierte [12] oder auch wie in manchen Arbeiten [35] zusätzlich in *Knowledge*-basierten Algorithmen unter-

zu dt. Wissens-basierte

gliedert. In der zweiten Phase muss dieses Modell dann überprüft werden. Speziell soll dabei mit noch nicht betrachteten Daten getestet werden ob Normalverhalten und Anomalien auch als solche eingestuft werden. Dabei liefert der Algorithmus für jede Eingabe entweder einen Anomaliescore oder direkt ein Label. Liegt der Score über einem festgelegten Schwellwert so gilt die Eingabe als anormal ansonsten als normal. [12]

Intrusion oder keine Intrusion

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

mehr dazu in Abschnitt 4.3

Die Trainingsphase wird von Chandola et al. [8] 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 eine Ungleichgewicht zwischen der Datenmenge an Angriffs- und Normalverhalten besteht. [62] Zusätzlich sind Anomalien meist unbekannt womit das Erstellen eines sinnvollen Datensatzes erschwertwird. Auch deswegen ist diese Herangehensweise weniger verbreitet. [21]

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. [60]. Jedoch sind auch *unsupervised* Trainingsverfahren umsetzbar welche keine Anomalien in den Daten enthalten.[21]

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 Abschnitt 4.3.1 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 anormales 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 [8].
- *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 Abschnitt 2.2.3

Die Funktionalität eines AIDS steht und fällt also mit den für das Training bereitgestellten Daten. Daher ist eine entscheidende Frage bei Verwendung von AIDS: 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 [35, 42]. 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. [3]

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 konzentriert sich HIDS 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 [35], oder aber auch Daten aus dem Kernel wie z.B. System Calls [47]. Im Gegensatz zur NIDS kann hier auf den Inhalt von jeder Information eingegangen werden, da die interne Kommunikation unverschlüsselt stattfindet. [6]

HOST-DATEN ZUR BESCHREIBUNG DES SYSTEMVERHALTENS Nach Bridges et al. [7] 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 [11]. System Calls hingegen sind klar definiert und beschreiben das Systemverhalten zunächst ungefiltert, unterliegen dafür allerdings einer weitaus größeren Varianz. [7] 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 [64], 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 ... [7]
- Wie wird die eigentliche Anomalie in den Daten erkannt?

Die letztere Frage soll speziell in Abschnitt 4.3 beleuchtet werden, doch wie das Systemverhalten präzise dargestellt werden kann soll nun behandelt werden. In dieser Arbeit werden System Calls als

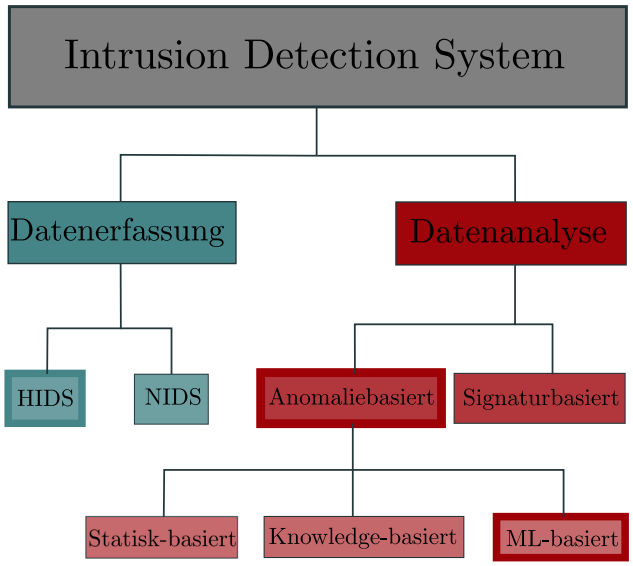

Figure 2.1: Einordnung des verwendeten IDS (Relevant für diese Arbeit dicker markiert).

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

zu dt. Systemaufrufe

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 einem gewöhnlichen Application Programming Interface (API) verglichen werden. Beispielhaft werden in Tabelle 2.1 zwei System Calls von Linux Betriebssystemen, ihre Argumente und die Art ihrer Rückgabewerte beschrieben.

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

Table 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 in welchem ein Programm aus dem User space über einen System Call eine privilegierte Aktion ausführen lassen kann dargestellt. Dies ermöglicht es Nutzerprozessen auf eine limitierte Auswahl an privilegierten Funktionen aus dem Kernel zugreifen zu können. [18]

zu dt. Betriebssystemkern

zu dt. Benutzer-Modus/Benutzerprozesse

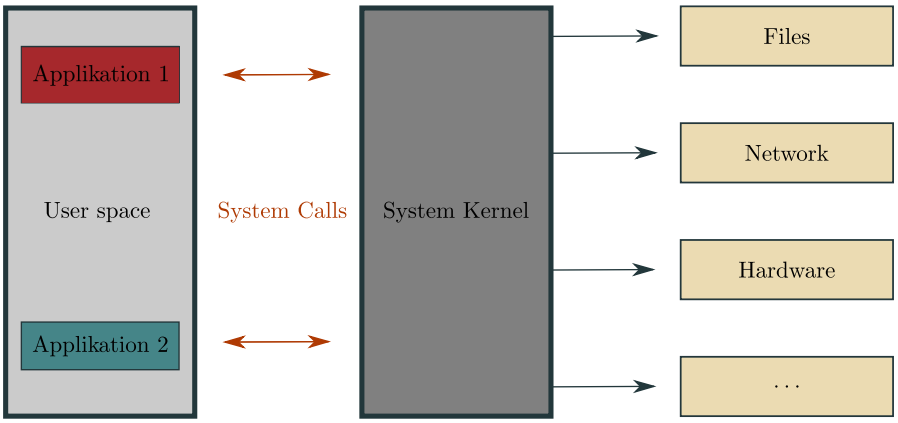

Figure 2.2: Darstellung der Kommunikation zwischen Applikationen aus dem User space mit dem System Kernel und dem Kernel mit Dateien/Netzwerk/Hardware/...

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 beschrieben besitzen System Calls Argumente die 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. [52] 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

Gekennzeichnet mit „res“

bei einem Fehler ein negativer Wert zurückgegeben [46]. 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 IDS*

Viele verschiedene IDS Ansätze betrachten die Sequenz von System Calls und erzielen vielversprechende Ergebnisse. [47] 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 [69, 70, 73] zeigen, wie dieser Spielraum ausgenutzt werden kann um unerkannt Angriffe durchzuführen. Tan et al. [70] 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. Ein weiterer Ansatz versucht lediglich die System Call Argumente zu verändern, ohne dabei die Sequenz zu beeinflussen [73]. 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 Abschnitt 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.

Verwendeter Algorithmus: STIDE [16]

2.2.3 *Datensätze*

Seit 1998 einer der ersten System Call Datensätze für HIDS veröffentlicht wurde [43, 45], 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 [45] 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 wechselnden 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 [17, 71, 72]. Unter anderem beschreiben McHugh et al. [48], 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. [71] und Engen [15] 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 [17] ist mittlerweile veraltet und kommt für eine weitere Betrachtung nicht in Frage, da er zusätzlich auch keine weiteren Kontextinformationen wie Thread IDs enthält [10].

ADFA-LD Der ADFA Linux Dataset (ADFA-LD) wurde von Creech et al. [10] 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 [26] 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 [23].

lid-ds! (lid-ds!) 2019 veröffentlichten Grimmer et al. [23] das *Leipzig Intrusion Detection-Data Set* (Leipzig Intrusion Detection Dataset (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 verwenden. Er wurde auf einem modernen Betriebssystem Ubuntu 18.04 aufgenommen und besteht aus

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

Table 2.2: Kurzbeschreibung der in dem Datensatz vorkommenden Szenarien und den dazugehörigen Sicherheitslücken [23]

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

Table 2.3: Ausschnitt aus der runs.csv des LID-DS, welche unter anderem die Labels für Angriffsfiles und Normalfiles enthalten. [23]

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	$res = 612, offset = 1225$		$fd = 12(< 4t > 172.131.12.1 : 123 \rightarrow 172.13.231.2 : 123), size = 2392$

Table 2.4: Ausschnitt aus den eigentlichen Aufzeichnungen von System Calls aus dem LID-DS [23]

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 Angriffszeitpunkt $t_{Angriff}$ wird es allerdings schwieriger. Denn liegt der Anomaliescore

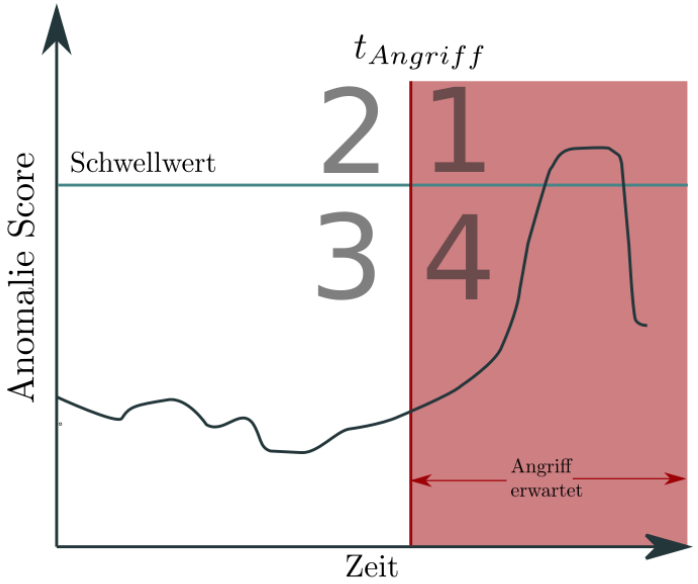

Figure 2.3: Illustration des Quadrantenproblems (Abwandlung von [22]). Es beschreibt die Problematik, dass Alle System Calls nach $t_{Angriff}$ als bösartig eingestuft werden müssen, obwohl auch Normalverhalten stattfindet.

nach dem Angriffszeitpunkt über dem Schwellwert, also im Quadranten 1, wird von einem True Positive (TP), also 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. [22] 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 [76], oder auch Abstrakter in der Umsetzung von Verhaltensweisen von Ameisen [51]. 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 [50, 61, 74].

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 Nets (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 Netze*

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

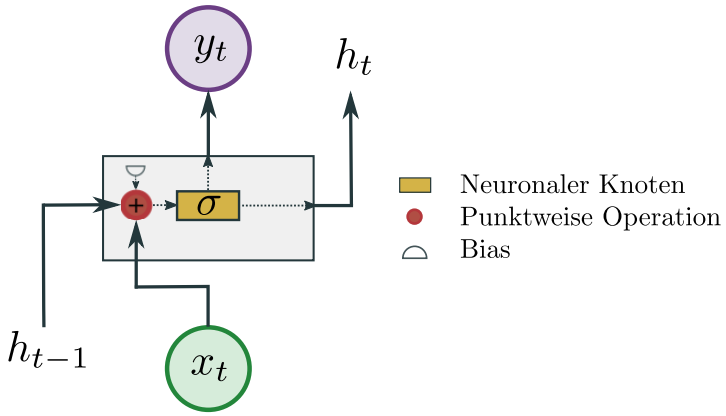

Figure 2.4: Darstellung einer RNN Zelle

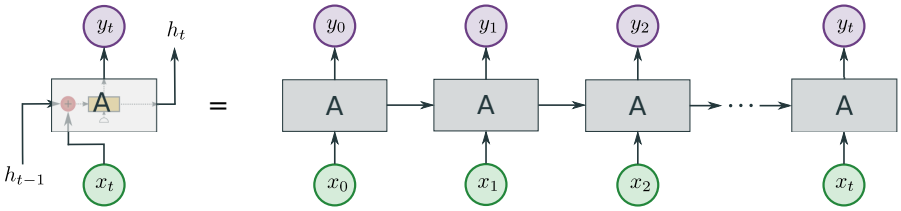

Figure 2.5: Darstellung einer „ausgerollten“ und vereinfachten RNN Zelle

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 *Long Short-Term Memory* (LSTM) Zellen ermöglichen durch verbesserte Fehlerkorrektur stabilere Lernergebnisse sowie auch das Lernen von Mustern mit noch größeren zeitlichen Abständen. [28]

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

2.3.2 *Long Short-Term Memory*

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.

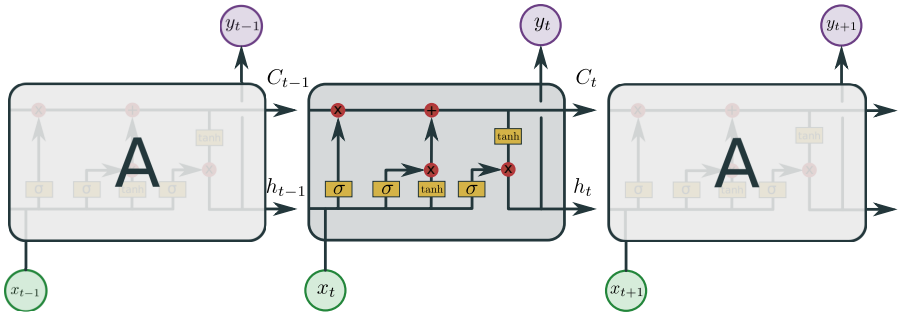

Figure 2.6: Schematische Darstellung eine Memory Cell A in einem LSTM NN, mit Input, Output und Forget Gate (Abwandlung von [58]).

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

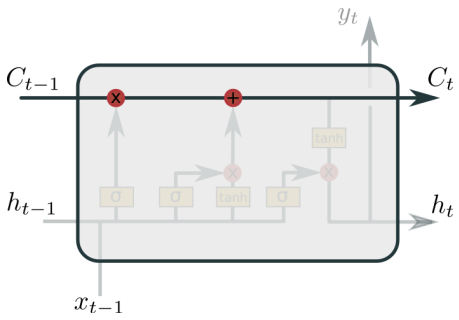

Figure 2.7: Weitergabe des Zellstatus innerhalb eines Knotens (Abwandlung von [58]).

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.

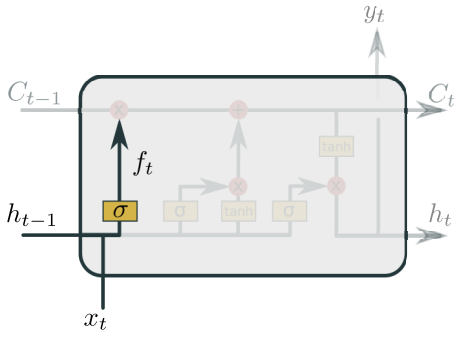

Figure 2.8: Einfluss des Forget Gates auf den Zellstatus (Abwandlung von [58]).

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. [19, 58]

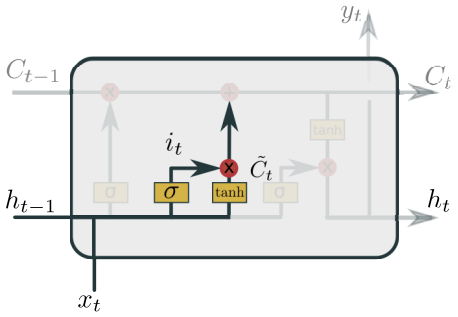

Figure 2.9: Einfluss des Input Gates auf den Zellstatus (Abwandlung von [58]).

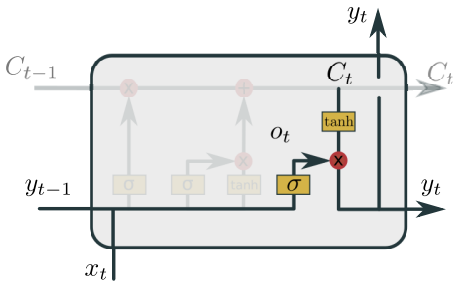

Figure 2.10: Das Output Gate regelt den Einfluss des Zellstatus auf die Ausgabe des Neurons (Abwandlung von [58]).

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 LSTM neuronale Netze in anomaliebasierten HIDS Vorteile bringen können. Und zum anderen wie durch die Anreicherung der Sequenzen von System Calls, durch weitere Informationen neben dem Namen des System Calls, die FP-Rate sowie die Detektionsrate (DR) verbessert werden kann. 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 dem Transfer von Natural Language Processing (NLP) Verfahren in die anomaliebasierte HIDS auseinandersetzen. Dazu zählen auch Ansätze der mit LSTM Netzwerken.

3.1 GRUNDLAGEN ANOMALIEDETEKTION

Die Anfänge der Anomalieerkennug werden von [31] auf die Arbeit von Grubbs [25] zurückgeführt, in welcher sogenannte Ausreißer in Sample-Daten gefunden und entfernt werden sollen. Hingegen berufen sich [8] bei den Anfängen der Anomalieerkennung auf eine Arbeit aus dem *Dublin Philosophical Magazine of Journal and Science* von 1887 [14]. Dort werden *discordant observations* anhand einer abweichenden gesetzmäßigen Frequenz isoliert. Speziell im Kontext der Angriffserkennung wird wie in Abschnitt 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 [8, 33, 60], 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 HIDS 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. [16] 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 zusammenhängende Sequenzen von System Calls befüllen. Kommt eine Sequenz nicht in der Datenbank vor wird diese als Anomalie eingestuft. So wird TIDE zu Sequence Time-Delay Embedding (STIDE). [30]

Ein anderen Ansatz wählten 1997 Lee et al. [40], wobei sie sich auch auf die Pionierarbeit von [16] berufen. Sie versuchen mit dem ML-Programm Repeated Incremental Pruning to Produce Error Reduction (RIPPER) Regeln aus den System Call Daten abzuleiten. Im Gegensatz zu [16] befinden sich bei diesem Ansatz normale sowie anormale Sequenzen in den Trainingsdaten.

1999 untersuchten Warrender et al. [75] wie verschiedene Algorithmen auf System Call Daten abschneiden. Dazu gehören die bereits erwähnten Verfahren TIDE, STIDE, RIPPER, sowie ein Hidden Markov Model (HMM). Dabei schienen alle Verfahren erfolgreich wobei das HMM als sehr rechenintensiv hervorgehoben wird. Speziell interessant an dieser Arbeit ist auch die Verwendung von n-grammen aus der Textverarbeitung. Ein n-gramm ist eine zusammenhängende Folge von n Elementen aus einer gegebenen Eingabe. Oft wird diese Art der Vorverarbeitung bei Datenstreams eingesetzt.

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. [34] 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 [16] und [30] bauen sie mit diesen Bündelungen eine Datenbank für das Normalverhalten auf.

zu dt. Bündel von Systemaufrufen

2013 stellen Murtaza et al. [55] 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*. Der Wahrscheinlichkeiten für das Auftreten von Zustandssequenzen wird dann zur Identifizierung von Anomalien genutzt.

2018 interpretieren Grimmer et al. [24] System Call Sequenzen als einen sogenannten *System Call Graph*. Dabei werden wie in [75] 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 dann die Übergangswahrscheinlichkeiten für alle Knoten. Der Anomaliescore eines Fensters von n-grammen aus den Testdaten wird dann 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 [73] werden verschieden Methoden untersucht mit welchen Angriffe nicht durch das IDS von [65] 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 [70] untersuchen speziell die von [16] 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.

beruht auf STIDE [30]

MIT BETRACHTUNG DER SYSTEM CALL ARGUMENTE Dies zeigt, dass es sinnvoll sein kann auch noch weitere Informationen welche in den System Calls enthalten sind 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. [38] 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 anormalen 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.S In der Testphase wird dann die Wahrscheinlichkeit dafür, dass die aktuelle String-Länge aus der Verteilung stammt mit der *Tschebyschaffschen 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. [38] aber auch besonders lange oder auffällige Verteilungen von Argumenten umge-

hen 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. [56] beschrieben. Beispielhaft wird aus dem Pfad `/usr/lib/libc.so` ⟶ `/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. [38] 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. Wurde ermittelt, dass ein Argument nicht einer Zufallsverteilung folgt, wird in der Testphase überprüft ob der aktuelle Wert in der Normaldatenbank vorkommt und dann eine 1 zurückgegeben und falls nicht eine 0. Folgt das Argument einer Zufallsverteilung wird immer 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. [38] Leider scheint das Framework aktuell nicht mehr nutzbar zu sein, da es online nicht mehr abrufbar ist.

Auch in [56] kommen diese Modelle der Argumente zum Einsatz.

Ebenso werden sie auch von Maggi et al. [47] 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. [37]

Luckett et al. [44] 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 [6]. Sie verwenden dafür nur das Timing von System Calls um mit neuronalen Netzen die Daten in normal oder

anormal zu klassifizieren. Dabei beschreiben sie aber nicht näher wie genau das Timing der System Calls verwendet wird.

Einen anderen Ansatz wählen Grimmer et al. indem sie die vorhandenen Informationen nicht direkt einbinden, sondern die Thread ID verwenden um die Streamverarbeitung anzupassen. Der Ausgangspunkt 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 zu verhindern bilden sie n-gramme, welche nur aus System Calls desselben Threads stammen. So konnten sie in den meisten Fällen eine Verringerung der Fehlalarme und eine Erhöhung der Erkennungsrate erreichen.[22]

3.3 NLP IN DER ANOMALIEDETEKTION UND HIDS

Speziell in den letzten Jahren wurden diverse weitere 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. [68] bereitgestellt. Besonders hervorzuheben ist dabei unter anderem die Studie von Wunderlich et al. [77] 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, Word2Vec (W2V) sowie GloVe und fastText. Zusätzlich untersuchen sie wie sich die Darstellung der System Calls als Kernelmodule, ähnlich zu [55], als Sequenz der Namen oder die Kombination beider auf die Ergebnisqualität auswirkt. In der Auswertung bevorzugen sie das One-Hot-Encoding, 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 Netzwerke [20, 78]. Diese Netzwerke finden ihre Anwendung auch in der HIDS, dazu gehören zum Beispiel [5, 9, 36, 57, 59, 67]. Doch die meisten Arbeiten verwenden entweder sehr alte oder anderweitig kritisch zu betrachtende Datensätze, wie in Abschnitt 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. [36], sie beschreiben wie sie Sprachmodelle der System Calls mithilfe von LSTM Netzwerken erstellen. Mit diesen Modellen wollen sie dann für eine Sequenz von System Calls eine Vorhersage treffen und die Ergebnisse der verschiedenen Modelle kombiniert ergeben dann einen Anomaliescore. Neben dem verwendeten Datensatz unterscheidet sich die Arbeit von Kim et al. von dieser Arbeit in dem Ansatz mehrere Sprachmodelle gleichzeitig aufzubauen und dass keine zusätzlichen Informationen der System Calls verwendet werden.

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. Des Weiteren werden hier ebenso keine zusätzlichen Argumente wie zum Beispiel den Rückgabewert betrachtet. [59]

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. [13]

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 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 einer AIDS müssen wie in Kapitel 2 beschrieben zwei Phasen durchlaufen werden. Die Trainingsphase und die Testphase. Diesen beiden Phasen liegt allerdings schon ein präparierter Datensatz und die Definition der exakten Eingaben und Ausgaben des LSTM zu Grunde. Welche Datensätze überhaupt in Frage kommen wurde bereits in Abschnitt 2.2.3 untersucht. 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. Zuvor in Abschnitt 4.1 werden verschiedene Tools betrachtet, die für die Vorverarbeitung und weitere Implementierungen nötig sind. Der eigentliche Algorithmus, also das Finden von Anomalien in den Daten wird dann im Abschnitt 4.3 beschrieben. Nachdem so also der verarbeitende Teil betrachtet wurde, soll in Abschnitt 4.4 die Strukturierung der Experimente präsentiert werden. Diese Strukturierung hat zum Ziel früh wenig vielversprechende Konfigurationen auszuschließen um ressourcenschonend Auswertungen durchzuführen. Speziell die Ressource Zeit, wie sich später zeigen wird, ist dabei im Rahmen dieser Arbeit eine Entscheidende. Abschließen soll dieses Kapitel dann in Abschnitt 4.5 die Untersuchung von Metriken, welche dann für die Auswertung der Experimente benötigt wird.

4.1 VERWENDETE TOOLS

Tensorflow Keras Rechencluster clara sysdig

4.2 VORVERARBEITUNG

Im kommenden Abschnitt 4.2.1 soll untersucht werden, welche Darstellungsformen für die eigentlichen System Calls interessant und sinnvoll sind. In dieser Arbeit werden die System Call Daten als Datenstream betrachtet. Weshalb in Abschnitt 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 zu erhöhen, bzw. die Fehlerrate zu verringern. Die Frage welche Informationen dafür verwendet werden können und wie diese dargestellt werden, soll in Abschnitt 4.2.2

4.2.1 *Wie wird ein System Call dargestellt?*

Neuronale Netze benötigen numerische werte deswegen Umwandlung von Namen der System Calls. 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 [4]. Werden die System Calls als kategorische Daten betrachtet bietet sich das für neuronale Netze typische One-Hot-Encoding (OHE) an.

ONE-HOT-ENCODING Für die Darstellung eines System Calls mit dem OHE, muss zunächst die Anzahl n der unterschiedlichen System Calls ermittelt werden. Der System Call sc_i aus der Menge der möglichen System Calls $SC = \{sc_1, sc_2, \ldots, sc_n\}$ wird dann als Bit-Vektor v_i der Länge n kodiert. Dabei nehmen alle Stellen bis auf i den Wert 0 an und dort den Wert 1. So wird aus dem System Call *open* aus der Menge $SC = \{open, close, read\}$ der Vektor $v_1 = (1, 0, 0)$ Da die Anzahl der möglichen System Calls und damit auch n 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 n 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 tatsächlich ebenso vorhandene Zusammenhänge verloren gehen. So besteht mindestens ein semantischer Zusammenhang zwischen *open* und *close* in der englischen Sprache, dieser ist in dem OHE verloren gegangen. Optimaler Weise gilt es also eine Darstellung zu finden die kurz ist und in der nur gewollte Ordnungen vorhanden sind. In der NLP gilt es ähnliche Probleme durch die Darstellungen der Wörter zu lösen, zum Beispiel mit dem W2V Verfahren.

WORD2VEC Das W2V Verfahren ist ein auf feedforward neuronalen Netzen basierender Ansatz, welcher häufig in der NLP eingesetzt wird [2]. Dabei werden Wörter aus einem gegebenen Satz an möglichen Wörtern anhand eines Trainingsdatensatzes in dichte Vektoren v fester Länge *embed_size* kodiert beziehungsweise *embedded*. Ziel des von [53] eingeführten W2V Verfahrens ist es eine Dimensionsreduktion durchzuführen, bei welcher möglichst viel Kontextinformationen erhalten bleiben. So sollen also die Vektordarstellungen von ähnlichen Wörtern ebenso ähnlich sein. 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. Wie umfangreich dieser Kontext für jedes Wort sein soll, wird mit der Fenstergröße *window_length* festgelegt.

zu dt. eingebettet

In der Trainingsphase gibt es zwei verschiedene Ansätze, zum einen das *Continuous Bag-Of-Words*, dabei wird versucht eine Vorhersage über ein Wort anhand des Kontextes zu machen. Hingegen wird unter Verwendung von *Skip-gram*, eine Vorhersage des Kontextes aufgrund des Wortes vorgenommen. [63]

Schwierigkeiten wie das häufige auftreten von Wörtern in der englischen Sprache wie *the*, treten in der System Call Domäne nicht auf. Die Vorteile der dichtbesetzten und dimensionsreduzierten Vektoren machen sich auch bereits [22] in der anomaliebasierten HIDS mit System Calls zu nutzen. Dieses Embedding wird auch im Folgenden verwendet.

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.

4.2.2 *Wie können weitere System Call Informationen dargestellt werden?*

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 einzelner Angriffe abbilden zu wollen, was wie in Abschnitt 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. [44] 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 Abschnitt 4.2.3. Um den Unterschied in den Normaldaten und Angriffsdaten zu untersuchen wurden in Abbildung 4.1 die

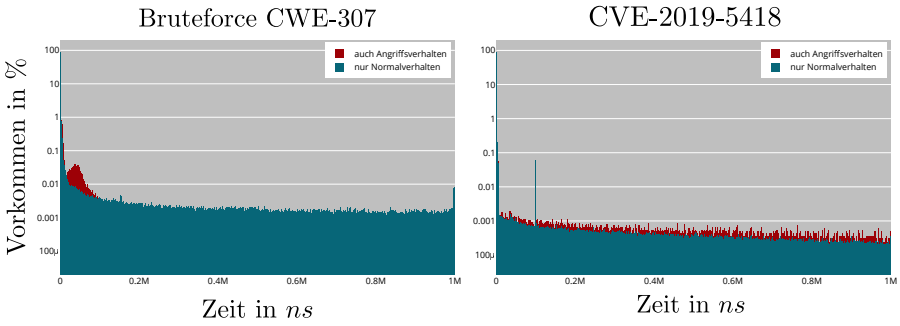

Figure 4.1: Dargestellt ist der zeitliche Abstand zwischen zwei System Calls aus dem selben Thread. Diese werden in ihrer Häufigkeit in Prozent an allen auftretenden Abstände in dem Plot eingetragen. Verwendet wurden dafür nur die Testdaten. Links für das Bruteforce Szenario und rechts für das CVE-2019-5418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

zeitlichen Abstände zwischen aufeinanderfolgenden System Calls aus dem Bruteforce und dem CVE-2019-5418 Szenario des **LIDDS!** (LIDDS!) 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 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 und diesen als weiteren Eingabeparameter für den verarbeitenden Algorithmus verwendet wird. 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.

zw. Normal- und Angriffsverhalten

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 könnten Informationen eingebettet werden

welche wenig Aussagekraft über das Normalverhalten eines Prozesses haben. Ob dies das Lernen des Normalverhaltens verbessert oder verschlechtert wird in Abschnitt 5.3.1 untersucht.

RETURN WERTE Wie in Abschnitt 2.2.1 angemerkt, besteht ein System Call aus zwei „Phasen“. Wobei die zweite Phase den Rückgabewert des Betriebssystemskernel beinhaltet. Bei erfolgreicher Durchführung ist dieser Rückgabewert positiv, ansonsten negativ. So gibt der Rückgabewert eines *write* System Calls an wieviele Bytes gelesen wurden. Der Rückgabewert enthält in diesem Beispiel also wertvolle Informationen über den tatsächlichen Ablauf des spezifischen System Calls. 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. Deswegen soll in diesem Abschnitt die Kodierung von System Call Rückgabewerten betrachtet werden welche Daten schreiben oder lesen und Daten über Sockets empfangen oder senden. Dabei wurden nur System Calls betrachtet, die auch im Datensatz vorkommen und einen Byte Rückgabewert haben. Leider fallen dabei System Calls wie *send* heraus, da der Rückgabewert die Anzahl der gesendeten Charaktere angibt und nicht die Anzahl an geschriebenen Bytes. In Tabelle 4.1 werden die ausgewählten 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 werden in Abbildung 4.2 die normalisierten Byte Rückgabewerte. 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 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. Weshalb $\rho_{norm} = -1$, falls ein System Call aus Tabelle 4.1 einen Fehlerwert liefert und damit $\rho = -1$ gilt. Wie sich die Verwendung der normalisierten Rückgabewerte spezieller System Calls auf die Ergebnisse auswirkt wird in Abschnitt 5.3.2 behandelt.

System Calls		
Name	Beschreibung	Rückgabewerte
write	Schreibt bis zu *count* Bytes aus dem Buffer (ab Stelle *buf*) in die Datei, welche über den Filedeskriptor *fd* definiert wird.	Geschriebene Bytes oder -1 bei Fehler
pwrite	Schreibt bis zu *count* Bytes aus dem Buffer (ab Stelle *buf*) 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 bis zu *count* Bytes von Filedeskriptor *fd* zu lesen und passt den Datei Offset an.	Filedeskriptor oder -1 bei Fehler
pread	Versucht bis zu *count* Bytes von Filedeskriptor *fd* zu lesen und passt den Datei Offset nicht an.	Filedeskriptor oder -1 bei Fehler
readv	Liest spezifizierten *iovcnt* Vektor aus Filedeskriptor *fd*	Gelesene Bytes oder -1 Fehler
sendfile, sendmsg	Sende Nachricht über Socket	Gesendete Bytes oder -1 bei Fehler
recvfrom, recv, recvmsg	Erhalte Nachricht über Socket	Empfangene Bytes oder -1 bei Fehler

Table 4.1: Kurzbeschreibung ausgewählter System Calls. [52]

4.2.3 *Wie wird ein Datenstream dargestellt?*

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 [66]. Auch in der auf System Call basierten Anomalieerkennung kommen n-gramme zum Einsatz [22, 24, 75]. Ein n-gramm ist eine zusammenhängende Folge von n Elementen aus einer gegebenen Eingabe. Diese werden wie in der linken Graphik in Abbildung 4.4 Beispielhaft dargestellt aufgebaut.

Figure 4.2: Histogramm der gelesenen/erhaltenen Bytes für die Testdaten des LID-DS [41] $CVE-2017-7529$ 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

In der Umsetzung wird ein Buffer der Länge n erzeugt, welcher das erste n-gram liefert sofern sich n Elemente in dem Buffer befinden. Zu beachten ist dabei, dass die Abfolge der System Calls mehrere logische Abfolgen zusammenführen. Denn moderne Computer Systeme verarbeiten mehrere Threads parallel. [18] 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. [22] 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 Graphik in Abbildung 4.4 veranschaulicht. Also nur System Calls aus demselben Thread bilden ein n-gram, weshalb sie von *Thread aware n-grams* sprechen. Grimmer et al. konnten zeigen, dass dies speziell auch für diesen Datensatz eine Verbesserung der Ergebnisse erreicht werden konnte. Zusätzlich zeigen sie, dass bei jedem einzelnen Scenario mindestens so gute Ergebnisse erzielt wurden.

zu dt. Thread bewusst

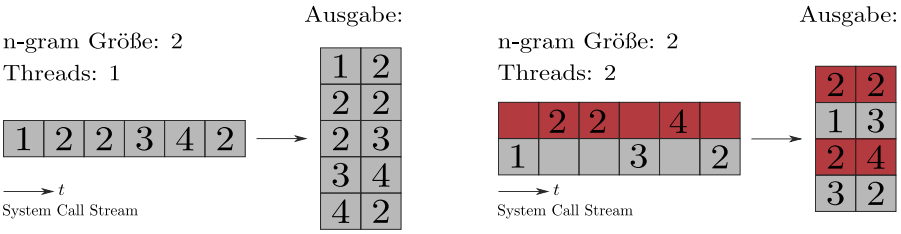

Figure 4.3: Erstellung der n-gramme mit $n = 2$ aus einem Datenstream von System Calls. Links ohne und rechts mit der Beachtung der Threadinformation. Die dabei entstehenden n-gramme dienen als Eingabe für den bewertenden Algorithmus. Die Reihenfolge der entstehenden n-gramme als Eingaben ist dabei von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

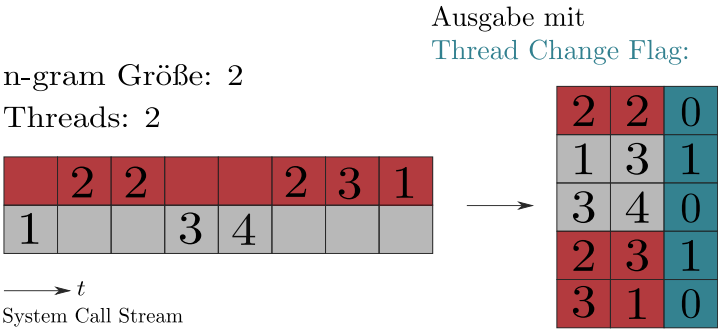

Figure 4.4

THREAD CHANGE FLAG Sie verwenden dafür aber ausschließlich Algorithmen welche kontextfrei arbeiten. Dies bedeutet, dass das zuvor gesehene n-gram keinen Einfluss auf das aktuelle hat. Wie in Abschnitt 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. Kodieren der tatsächlichen Thread ID unpraktisch, da Thread IDs wiederverwendet werden können. Dies wird erreicht indem ein n-gram mit der Information über einen Kontextwechsel angereichert wird. Initial wird, weil noch kein Kontextwechsel stattgefunden hat, für die Thread Change Flag (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.

Um zu untersuchen inwieweit die TCF Einfluss auf die Auswertung haben könnte wird in Tabelle 4.2 die der Anteil an n-grammen mit Kontextwechsel dargestellt.

4.3 ALGORITHMUS

LSTM Sprachmodell soll Wahrscheinlichkeit des nächsten System Calls vorhersagen, gegeben eines System Calls oder einer Sequenz von

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
$CVE-2018-3706$	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	10,891,051	112549739	8.8

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

System Calls. Gab es in den Trainingsdaten die feste Menge $S = 1, \ldots, N$ an System Calls, so gibt $x = x_1 \ldots x_l$ $(x_i \in S)$ die Sequenz an l System Calls an. Jeder dieser System Calls bekommt im ersten Schritt einen Integerwert zwischen 1 und N. Taucht in den Testdaten nun ein noch nicht bekannter System Call x_i auf, also $x_i \notin S$, so erhält dieser den vorläufigen Wert 0. Zu jedem Zeitpunkt wird x_i der Input Layer übergeben. Dabei wird ein Embedding aus Abschnitt **??** verwendet. Mit den gegebenen Trainingsdaten kann nun das LSTM mittels des *back-propagation through time* (BPTT) trainiert werden. An der Ausgangs Layer befindet sich eine Softmax Aktivierungsfunktion. Diese wird verwendet um die Ausgabe zu normalisieren und damit die Wahrscheinlichkeitsverteilung des nächsten System Calls zu erhalten. Also $P(x_i | x_{1:i-1})$ für alle i.

4.3.1 *Training*

Nächsten Syscall vorhersagen und überprüfen ob richtig vorhergesagt

4.3.2 *Anomalieerkennung*

Es kann also bei Auftreten des System Calls x_i überprüft werden mit welcher Wahrscheinlichkeit p dieser vorhergesagt wurde. Der eigentliche Anomalie-Score wird dann folgenderweise berechnet:

$$ascore = 1 - p \tag{4.3}$$

Unterschreitet dieser einen Schwellwert so wird dies als eine Anomalie gewertet und ein Alarm angezeigt.

4.3.3 *Schwellwertbestimmung*

Um den zuvor erwähnten Schwellwert automatisch zu bestimmen, wird der Algorithmus auf die Validierungsdaten angewendet. Dabei dient der höchste Wert dieser Daten dann als Schwellwert, da angenommen wird, dass mindestens alle Daten aus den Validierungsdaten harmlos sind und damit unter dem Schwellwert liegen sollten. Wichtig ist dabei dafür nicht die Trainingsdaten zu wählen, da eine starke Verzerrung der Schwellwertes durch Overfitting der Daten entstehen kann. Das würde bedeuten, dass nur sehr geringe Anomaliewerte auftreten und der Schwellwert sehr gering ist und damit die Gefahr für viele Fehlalarme besteht.

Alternativ betrachte die x wahrscheinlichsten vorhergesagten system calls, falls tatsächlicher system call nicht dabei –> alarm x ermitteln, betrachte validierungsdaten und schaue ob schlechtestes x aussehen würde tatsächlich oft einmal schlechteste platzierung und automatische erkennung von x schwer.

In Datensatzkapitel!

zu dt. schädlich

PROBLEM DES DATENSATZ In den Testdaten sind *malicious* Files beinhaltet, welche eine Information über den Angriffszeitpunkt liefert. Jedoch gibt es bei einer malicious File im Gegensatz zu den normalen Files vier mögliche Zuordnungen. Befindet sich der Anomaliescore unter dem Schwellwert kann von einem *True Negative* eingestuft werden. Also es wurde korrekter weise kein Alarm vorhergesagt. Befindet sich der Anomaliescore vor dem Angriffszeitpunkt über dem Schwellwert liegt ein *False Positive* vor. Es wurde ein Alarm gemeldet an einer Stelle an dem kein Angriff stattfand. Nach dem angegebenen Angriffszeitpunkt wird es allerdings schwieriger. Denn liegt der Anomaliescore nach dem Angriffszeitpunkt über dem Schwellwert, wird von einem *True Positive*, also einem korrektem Alarm ausgegangen. Jedoch kann da der Angriff schon vorbei sein, oder gar noch nicht gestartet sein. Es können nach dem Angriffszeitpunkt auch *False Positive* oder *False Negative* geben, welche allerdings nicht als solche erkannt werden können. Wie sich das auf die Auswertung der Ergebnisse auswirkt wird in Kapitel 4.5 beleuchtet.

4.3.4 *Parameterwahl*

ngram länge lstm merkt sich vorherige syscalls aber hinzunahme von syscalls weitere info -> finden von sweet spot generell großes n viele alarme kleines n weniger alarme vorteil LSTM? wichtiger Parameter den es zu ermitteln gilt

4.4 STRUKTURIERUNG DER EXPERIMENTE

Um aussagekräftige Experimente zu entwickeln müssen zuerst überlegungen zur praktischen umsetzung gemacht werden dabei wird in ersten Tests klar, dass zeit hierbei eine große rolle spielen wird

erste Tests also ausgelegt um Faktoren zu ermitteln, welche die auswertungen stark verlangsamen und diese ausschließen.

4.4.1 *Faktor Zeit*

zeit/dr als groesse und farbe von scatter plot batch size test und train x/y achse

eingrenzen von moeglichen konfigurationen

Berechnungszeiten aus verschiedenen Perspektiven relevant: soll live system werden begrenzte rechenleistung und viele Tests zur auswertung von parametern architektur etc erster test zur abschätzung diverser zeitl. faktoren:

Faktoren:

- Architektur
- Verarbeitung Stream

  ngram größe
- embedding

ngram größe, architektur und verwendung w2v statt ohe Grobe Abschätzung der Zeit, da Berechnungen auf Clustern ausgeführt werden von Auslastung beeinflusst werden. Klare Erkenntnisse:

Single Small 50 neuronen eine schicht: Single Big 250 neuronen eine schicht multi 50 neuronrn 3 schichten

erste Abschätzung von Nutzen von Thread einführen von stateful sowie Batch Normalization

4.4.2 *Optimale Parameter*

ARCHITEKTUR versch architekturen: Single Small 50 neuronen eine schicht Single Big 250 neuronen eine schicht multi small 20 neuronen 3 schichten multi big 50 neuronrn 3 schichten deep erste 50 sonst 20 6 schichten

singlesmall 43% von Deep insgesamt am schnellsten single small wie zu erwarten, deep am langsamsten

teste eine schicht viele neuronen eine schicht wenige neuronen mehrere schichten mehrere neuronen / mit dropout dazwischen viele schichten wenige neuronen /mit dropout dazwischen

auf Grund des zeitlichen Faktors fallen Deep und multibig weg Also zu testen: Single Small Single Multi Small Multi

HYPERPARAMETER aktivierungs funktion -> dense layer with softmax or tanh batch size learning rate optimizer

NGRAM GRÖSSE ngram größer -> langsamer

THREADINFO Hypothese: Threadinfos bringen was

Einbinden von thread information auf verschiedenen wegen: Thread aware ngrams (tan) Thread aware ngrams for w2v (tanw2v) Thread change flag (tcf)

varianten: tan tanw2v tcf tan tcf tan tanw2v tcf tanw2v tan tanw2v tcf

—> welcher dieser varianten am besten?

PARAMETER args time

LSTM ohne Threadinfos mit OHE LSTM mit W2V ohne Threadinfos (ngram) LSTM mit W2V mit Threadinfos (ngram) LSTM mit W2V threadaware mit Threadinfos (ngram) LSTM mit W2V threadaware mit Threadinfos (ngram) und threadchangeflag LSTM mit W2Vthreadaware mit Threadinfos (ngram) und threadchangeflag, spezialtraining –> LSTM final

Manche angriffe verändern Sequenz von syscalls nicht Hypothese: verwende Parameter um erg zu verb

LSTM final + strlen LSTM final + time delta LSTM final + strlen + time delta

4.5 METRIKEN

Auf Grund dessen Metrik False Alarm/ consecutive false alarm und Detection rate falls einmal pro malicious file in quadrant 4 -> HIT Wahl von Metriken in NN Precision, Recall, f-score, TNR, FNR, FPR

problematisch: nicht auf systemcall genau gelabelt recall precision usw nur auf file ebene: alarm nach exploitstarttime wird immer als hit gewertet -> aber evtl angriff noch nicht begonnen oder angriff bereits vorbei ebenso umgekehrt, eig muss jeder nicht alarm nach exploitstart als FN gewertet werden weswegen filegenau geschaut wird vorteil des Datensatzes gegenüber anderen, immerhin exploitstart time

ilarm in quadrant —> image

5 ERGEBNISSE

5.1 OPTIMALE PARAMETER

ARCHITEKTUR versch architekturen: Single Small 50 neuronen eine schicht Single Big 250 neuronen eine schicht multi small 20 neuronen 3 schichten multi big 50 neuronrn 3 schichten deep erste 50 sonst 20 6 schichten

singlesmall 43% von Deep insgesamt am schnellsten single small wie zu erwarten, deep am langsamsten

HYPERPARAMETER <++> aktivierungs funktion -> dense layer with softmax or tanh batch size learning rate optimizer

NGRAM GRÖSSE ngram größer -> langsamer

EMBEDDING overhead berechnung embedding, muss allerdings nur einmal berechnet werden zu erkennen w2v mit embedding size = 2 und window = 4 wesentlich schneller embedding größer -> langsamer

vergleich ngram im schnitt mit ngram gr 2 84% von ngr 3 und

w2v bringt entscheidenden Vorteil gegenüber ohe: Jeweils vergleich der selben parameter außer w2v vs ohe: Single small w2v nur 30% der zeit gegenüber single small ohe bei mulit w2v sogar nur 13% im mittel über alle architekturen 21.5% der Zeit von ohe bei verwendung w2v

ARCHITEKTUR teste eine schicht viele neuronen eine schicht wenige neuronen mehrere schichten mehrere neuronen / mit dropout dazwischen viele schichten wenige neuronen /mit dropout dazwischen

auf Grund des zeitlichen Faktors fallen Deep und multibig weg Also zu testen: Single Small Single Multi Small Multi

THREADINFO Hypothese: Threadinfos bringen was

Einbinden von thread information auf verschiedenen wegen: Thread aware ngrams (tan) Thread aware ngrams for w2v (tanw2v) Thread change flag (tcf)

varianten: tan tanw2v tcf tan tcf tan tanw2v tcf tanw2v tan tanw2v tcf

—> welcher dieser varianten am besten?

PARAMETER args time

LSTM ohne Threadinfos mit OHE LSTM mit W2V ohne Threadinfos (ngram) LSTM mit W2V mit Threadinfos (ngram) LSTM mit W2V threadaware mit Threadinfos (ngram) LSTM mit W2V threadaware mit Threadinfos (ngram) und threadchangeflag LSTM mit W2Vthreadaware mit Threadinfos (ngram) und threadchangeflag, spezialtraining –> LSTM final

Manche angriffe verändern Sequenz von syscalls nicht Hypothese: verwende Parameter um erg zu verb

LSTM final + strlen LSTM final + time delta LSTM final + strlen + time delta

5.2 LSTM ANSATZ

5.3 EXTRA PARAMETER

5.3.1 *Timing*

5.3.2 *Return Value*

6 FOLGERUNGEN

6.1 SCHLÜSSE

6.2 AUSBLICK

BIBLIOGRAPHY

[1] Adamu I Abubakar, Haruna Chiroma, Sanah Abdullahi Muaz, and 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), pp. 221–227.

[2] V Kishore Ayyadevara. "Word2vec." In: *Pro Machine Learning Algorithms*. Springer, 2018, pp. 167–178.

[3] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Network Anomaly Detection: Methods, Systems and Tools." In: *IEEE Communications Surveys Tutorials* 16.1 (2014), pp. 303–336. DOI: 10.1109/SURV.2013.052213.00046.

[4] C.M. Bishop, P.N.C.C.M. Bishop, G. Hinton, and Oxford University Press. *Neural Networks for Pattern Recognition*. Advanced Texts in Econometrics. Clarendon Press, 1995. ISBN: 9780198538646. URL: https://books.google.de/books?id=T0S0BgAAQBAJ.

[5] Lydia Bouzar-Benlabiod, Lila Méziani, Stuart H Rubin, Kahina Belaidi, and 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, pp. 75–82.

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

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

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey." In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

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

[10] Gideon Creech and 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, pp. 4487–4492.

[11] Gideon Creech and 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), pp. 807–819. DOI: 10.1109/TC.2013.13.

[12] Roberto Di Pietro and Luigi V Mancini. *Intrusion detection systems*. Vol. 38. Springer Science & Business Media, 2008.

[13] Michael Dymshits, Benjamin Myara, and David Tolpin. "Process monitoring on sequences of system call count vectors." In: *Proceedings - International Carnahan Conference on Security Technology* 2017-Octob (2017), pp. 1–5. ISSN: 10716572. DOI: 10.1109/CCST.2017.8167792. arXiv: 1707.03821.

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

[15] 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." PhD thesis. Bournemouth University, 2010.

[16] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. "A sense of self for Unix processes." In: *Proceedings 1996 IEEE Symposium on Security and Privacy*. 1996, pp. 120–128. DOI: 10.1109/SECPRI.1996.502675.

[17] S. Forrest and University of New Mexico. *University of New Mexico (UNM) Intrusion Detection Dataset*. URL: https://www.cs.unm.edu/~immsec/systemcalls.htm.

[18] Peter B Galvin, Greg Gagne, Abraham Silberschatz, et al. *Operating system concepts*. Vol. 10. John Wiley & Sons, 2003.

[19] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. "Learning to Forget: Continual Prediction with LSTM." In: *Neural Comput.* 12.10 (Oct. 2000), pp. 2451–2471. ISSN: 0899-7667. DOI: 10.1162/089976600300015015. URL: http://dx.doi.org/10.1162/089976600-300015015.

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

[21] Markus Goldstein and Seiichi Uchida. "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data." In: *PLoS ONE* 11.4 (2016), pp. 1–31. ISSN: 19326203. DOI: 10.1371/journal.pone.0152173.

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

[23] Martin Grimmer, Martin Max Röhling, D Kreusel, and Simon Ganz. "A modern and sophisticated host based intrusion detection data set." In: *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung* (2019), pp. 135–145.

[24] Martin Grimmer, Martin Max Röhling, Matthias Kricke, Bogdan Franczyk, and Erhard Rahm. "Intrusion Detection on System Call Graphs." In: *Sicherheit in vernetzten Systemen, pages G1–G18* (2018).

[25] Frank E. Grubbs. "Procedures for Detecting Outlying Observations in Samples." In: *Technometrics* 11.1 (1969), pp. 1–21. DOI: 10.1080/00401706.1969.10490657. URL: https://doi.org/10.1080%2F00401706.1969.10490657.

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

[27] S. He, J. Zhu, P. He, and M. R. Lyu. "Experience Report: System Log Analysis for Anomaly Detection." In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, pp. 207–218. DOI: 10.1109/ISSRE.2016.21.

[28] 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), pp. 107–116.

[29] Sepp Hochreiter and 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.

[30] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. "Intrusion Detection Using Sequences of System Calls." In: *J. Comput. Secur.* 6 (1998), pp. 151–180.

[31] Kishan G. Mehrotra Chilukuri K. Mohan HuaMing Huang. *Anomaly Detection Algorithms and Principles*. 2017, pp. 1–229. ISBN: 9783319675244. URL: http://www.springer.com/series/11955%0Ahttp://link.springer.com/10.1007/978-3-319-67526-8.

[32] Bundesamt für Sicherheit in der Informationstechnik. *Die Lage der IT-Sicherheit in Deutschland 2021*. 2021. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2021.pdf?__blob=publicationFile&v=3#%5B%7B%22num%22%3A44%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22FitR%22%7D%2C-396%2C-2%2C991%2C844%5D.

[33] Shijoe Jose, D Malathi, Bharath Reddy, and Dorathi Jayaseeli. "A survey on anomaly based host intrusion detection system." In: *Journal of Physics: Conference Series*. Vol. 1000. 1. IOP Publishing. 2018, p. 012049.

[34] Dae-Ki Kang, Doug Fuller, and 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, pp. 118–125.

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

[36] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, and 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).

[37] Oualid Koucham, Tajjeeddine Rachidi, and 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), pp. 1010–1016. DOI: 10.1109/IntelliSys.2015.7361267.

[38] Christopher Kruegel, Darren Mutz, Fredrik Valeur, and Giovanni Vigna. "On the detection of anomalous system call arguments." In: *European Symposium on Research in Computer Security*. Springer. 2003, pp. 326–343.

[39] Vipin Kumar. "Parallel and distributed computing for cybersecurity." In: *IEEE Distributed Systems Online* 6.10 (2005).

[40] Wenke Lee, S.J. Stolfo, and 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), pp. 50–56. URL: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Learning+Patterns+from+Unix+Process+Execution+Traces+for+Intrusion+Detection+1#0.

[41] *Leipzig Intrusion Detection Data Set*. URL: https://www.exploids.de/lid-ds/.

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

[43] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. "1999 DARPA off-line intrusion detection evaluation." In: *Computer Networks* 34.4 (2000), pp. 579–595. ISSN: 13891286. DOI: 10.1016/S1389-1286(00)00139-0.

[44] Patrick Luckett, J Todd McDonald, and Joel Dawson. "Neural network analysis of system call timing for rootkit detection." In: *2016 Cybersecurity Symposium (CYBERSEC)*. IEEE. 2016, pp. 1–6.

[45] Lincoln Laboratory MIT. *DARPA Intrusion Detection Evaluation Data Set*. 1998-2000. URL: https://www.ll.mit.edu/r-d/datasets.

[46] R. MOONA. *ASSEMBLY LANGUAGE PROGRAMMING IN GNU/LINUS FOR IA32 ARCHITECTURES*. PHI Learning, 2009. ISBN: 9788120331563. URL: https://books.google.de/books?id=C3_WIQOYE2EC.

[47] F. Maggi, M. Matteucci, and S. Zanero. "Detecting Intrusions through System Call Sequence and Argument Analysis." In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (2010), pp. 381–395. DOI: 10.1109/TDSC.2008.69.

[48] 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), pp. 262–294. ISSN: 15577406. DOI: 10.1145/382912.382923.

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

[50] Patricia Melin, Julio Cesar Monica, Daniela Sanchez, and 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. DOI: 10.3390/healthcare8020181. URL: https://www.mdpi.com/2227-9032/8/2/181.

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

[52] Andries Brouwer Michael Kerrisk Stepan Kasal. *syscalls(2) — Linux manual page*. 2021. URL: https://man7.org/linux/man-pages/man2/syscalls.2.html.

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

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

[55] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj, and 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, pp. 431–440. DOI: 10.1109/ISSRE.2013.6698896.

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

[57] Zijian Niu, Ke Yu, and Xiaofei Wu. "LSTM-Based VAE-GAN for Time-Series Anomaly Detection." In: *Sensors* 20.13 (2020), p. 3738. ISSN: 1424-8220. DOI: 10.3390/s20133738. URL: http://dx.doi.org/10.3390/s20133738.

[58] Christopher Olah. *Understanding LSTM Networks*. Aug. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[59] Daekyeong Park, Sangsoo Kim, Hyukjin Kwon, Dongil Shin, and Dongkyoo Shin. "Host-Based Intrusion Detection Model Using Siamese Network." In: *IEEE Access* 9 (2021), pp. 76614–76623.

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

[61] Refat Khan Pathan, Munmun Biswas, and 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), p. 110018.

[62] Clifton Phua, Damminda Alahakoon, and Vincent Lee. "Minority report in fraud detection: classification of skewed data." In: *Acm sigkdd explorations newsletter* 6.1 (2004), pp. 50–59.

[63] M.T. Pilehvar and 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. URL: https://books.google.de/books?id=U90MEAAAQBAJ.

[64] *Seeing is Securing For containers, Kubernetes and cloud services*. URL: https://sysdig.com/.

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

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

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

[68] Zarrin Tasnim Sworna, Zahra Mousavi, and 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), pp. 1–35. arXiv: 2201.08066. URL: http://arxiv.org/abs/2201.08066.

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

[70] Kymie MC Tan and 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, pp. 188–201.

[71] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and 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), pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[72] Amjad M. Al Tobi and Ishbel Duncan. "KDD 1999 generation faults: a review and analysis." In: *Journal of Cyber Security Technology* 2.3-4 (2018), pp. 164–200. DOI: 10.1080/23742917.2018.1518061. eprint: https://doi.org/10.1080/23742917.2018.1518061. URL: https://doi.org/10.1080/23742917.2018.1518061.

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

[74] Peipei Wang, Xinqi Zheng, Gang Ai, Dongya Liu, and 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), p. 110214.

[75] Christina E. Warrender, Stephanie Forrest, and 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), pp. 133–145.

[76] Gregory S Watson, David W Green, Lin Schwarzkopf, Xin Li, Bronwen W Cribb, Sverre Myhra, and 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), pp. 109–122.

[77] Sarah Wunderlich, Markus Ring, Dieter Landes, and 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, pp. 14–24.

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