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

CONTENTS

1 EINFÜHRUNG 3
1.1 Einleitung . . . 3
1.2 Zielsetzung . . . 4
2 GRUNDLAGEN 7
2.1 Intrusion Detection System . . . 7
2.1.1 Datenanalyse . . . 7
2.1.2 Datenerfassung . . . 10
2.2 System Calls . . . 12
2.2.1 Allgemeines . . . 12
2.2.2 System Calls für IDS . . . 13
2.2.3 Datensätze . . . 14
2.3 Künstliche neuronale Netze . . . 17
2.3.1 Rekurrente neuronale Netze . . . 18
2.3.2 Long Short-Term Memory . . . 20
3 VERWANDTE ARBEITEN 25
3.1 Anomaliedetektion mit System Calls . . . 25
3.1.1 Erweiterung der System Call Sequenzen . . . 25
3.2 Untersuchen von Zeitreihen mit LSTM . . . 25
3.2.1 System Call mit LSTM . . . 25
4 REALISIERUNG 27
4.1 Verwendete Tools . . . 27
4.2 Vorverarbeitung . . . 27
4.2.1 Vorverarbeitung des Datensatzes . . . 28
4.2.2 Wie wird ein System Call dargestellt? . . . 28
4.2.3 Wie wird ein Datenstream dargestellt? . . . 30
4.3 Algorithmus . . . 30
4.3.1 Training . . . 30
4.3.2 Anomalieerkennung . . . 30
4.3.3 Schwellwertbestimmung . . . 31
4.3.4 Parameterwahl . . . 31
4.4 Strukturierung der Experimente . . . 32
4.4.1 Faktor Zeit . . . 32
4.4.2 Optimale Parameter . . . 32
4.5 Metriken . . . 33
5 ERGEBNISSE 35
5.1 LSTM Ansatz . . . 35
5.1.1 Optimale Parameter . . . 35
5.2 Extra Parameter . . . 36
6 FOLGERUNGEN 37
6.1 Schlüsse . . . 37
6.2 Ausblick . . . 37

BIBLIOGRAPHY 38

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. [22].

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 [18].

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 [45] 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. [10] betrachtet lediglich die Sequenzen der System-Calls. Maggi et al. verwenden zusätzlich auch Parameter und verweisen in ihrer Arbeit [32] 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 [20] und können in unterschiedlichsten Architekturen einen Nutzen bringen.

1.2 ZIELSETZUNG

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 [28] 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?

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 den Ort 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 einem System Schaden zufügen kann. Ein IDS versucht automatisch diese unautorisierten Aktivitäten zu identifizieren, um dadurch die Sicherheit des Systems gewährleisten zu können. [29] alternativquelle (Buch?)

zu dt. Eindringen, Einbruch

Um solche unautorisierte 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. [24, 29, 38]. 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 darauf, eine ausgeprägte Datenbank an Signaturen von bekannten Angriffen zu besitzen. Verschiedene Methoden vergleichen dann aktuelle Signaturen des Systems mit der Datenbank. Es gibt diverse Methoden wie diese Signaturen erstellt werden. Dafür werden zum Beispiel in

mehreren Arbeiten Netzwerk Pakete betrachtet [24] oder auch ein wenig ausgefeilter *State Machines* erstellt [34].

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, da diese noch nicht in der Datenbank vorhanden sind. [24] 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.

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 Systemverhalten, ohne unerwünschtes Verhalten also zum Beispiel Angriffe, beschreiben. Anomalien des Verhaltens können 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 [27]. Anomalien in den Daten können aus verschiedenen Gründen entstehen, zum Beispiel durch böswillige Aktivitäten oder aber auch durch Programmfehler. Doch alle Anomalien haben gemein, dass sie ein Abweichen eines „Normalverhalten“ 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. Dabei können ML-basierte, Statistik-basierte oder auch *Knowledge*-basierte Algorithmen verwendet werden. In der zweiten muss dieses Modell dann überprüft werden. Speziell soll dabei mit noch nicht betrachteten Daten die Generalisierung des Modells getestet 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.

zu dt. Wissens-basierte

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. [5] 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. [44]

Hingegen wird beim *Semi-supervised* Verfahren ein Trainingsdatensatz benötigt welcher lediglich das Normalverhalten kennzeichnet. Somit werden auch keine Angriffe zum Trainingszeitraum gesehen.

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, da es sonst häufig zu Fehlalarmen kommen kann, falls Anomalien fälschlicherweise als Normalverhalten eingestuft wird. [42]. Jedoch sind auch *unsupervised* Trainingsverfahren möglich welche keine Anomalien in den Daten enthalten.*QUELLE*

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 [5].
- *Unzulängliche Datensätze*: Daten zur Erfassung des Normalverhaltens sind oft veraltet oder nicht sehr detailreich.

Mehr dazu in Abschnitt 2.2.3

- *Findung eines sinnvollen Schwellwerts*: Festlegung eines Schwellwertes, welcher die eigentliche Unterscheidung zwischen Normalverhalten und Angriffsverhalten umsetzt.

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 [24, 29]. 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 [24], oder aber auch Daten aus dem Kernel wie z.B. System Calls [32]. Im Gegensatz zur NIDS kann hier auf den Inhalt von jeder Information eingegangen werden, da die interne Kommunikation unverschlüsselt stattfindet.

HOST-DATEN ZUR BESCHREIBUNG DES SYSTEMVERHALTENS Nach Bridges et al. besteht im wesentlichen die Auswahl zwischen Log-Files und *System Calls*. 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 es findet aber bereits eine starke Filterung statt. Außerdem müssen die angesprochenen Filter manuell eingestellt wer-

zu dt. Systemaufrufe

den. System Calls hingegen sind klar definiert und beschreiben das Systemverhalten zunächst ungefiltert, unterliegen dafür allerdings einer weitaus größeren Varianz. [4] 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 [45], 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

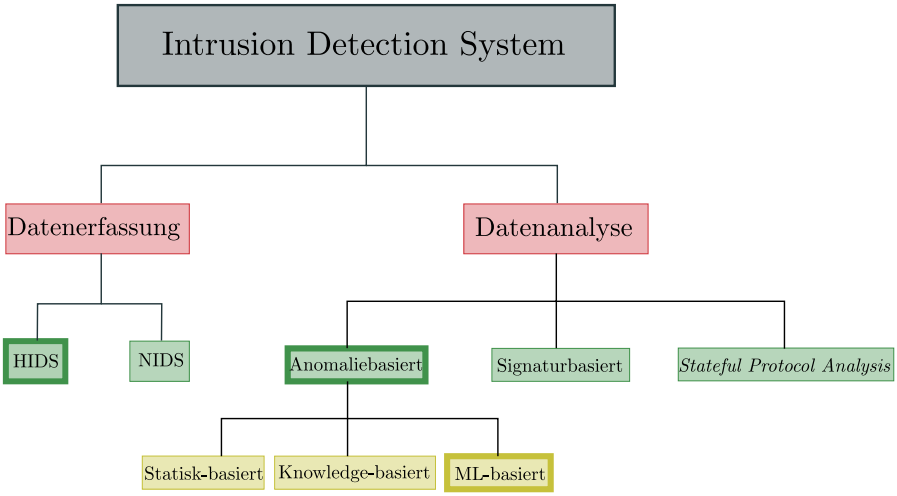

Figure 2.1: Einordnung des verwendeten IDS (Relevant für diese Arbeit dicker markiert).

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.Grafik anpassen 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 ... [4]
- 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.

2.2 SYSTEM CALLS

Jegliche Programme die auf einem Rechner mit einem Betriebssystem laufen, müssen mit diesem interagieren um Veränderungen am System vornehmen zu können. Diese Interaktion findet in Form von *System Calls* statt und kann mit 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.

zu dt. Systemaufrufe

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

Table 2.1: Beschreibung 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. [12]

zu dt. Betriebssystemkern

Figure 2.2: System Call Ablauf

Wie in Tabelle 2.1 ebenfalls beschrieben besitzen System Calls auch 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. [37] Zu jedem öffnenden System Call gibt es einen schließenden System Call, der Quasi die Antwort des Kernels darstellt. Schließende System Calls 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. Zu einem System Call gehört also mehr als nur der Funktionsname.

Gekennzeichnet mit „res“

Auch ein möglicher Angreifer muss um Schaden anzurichten also an den System Calls eine Veränderung 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. [32] 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 [46, 47, 50] zeigen, wie dieser Spielraum ausgenutzt werden kann um unerkannt Angriffe durchzuführen. Tan et al. [47] erreichen dies durch die Veränderung eines zuvor von der IDS erkannten Angriffes. Sie beschreiben, wie dem IDS fremde System Call Sequenzen derart Verändert werden können, dass sie als normal eingestuft werden. 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 [50]. Was diese Beispiele jedoch auch zeigen, ist dass wenigstens ein Faktor, also entweder die Abfolge, also die Sequenz von System Calls verändert werden, z.B. neue Funktionen aufrufen welche wiederum andere System Calls auslösen, oder es werden die Argumente der System Calls verändert. So könnte 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 4.2.2.1 untersucht werden. Im dem nachfolgenden Kapitel wird nun auf verschiedene Datensätze, welche System Call Sequenzen und teilweise Argumente und weiter Metadaten enthalten, eingegangen.

Verwendeter Algorithmus: STIDE [10]

2.2.3 *Datensätze*

Seit 1998 einer der ersten System Call Datensätze für HIDS veröffentlicht wurde [30, 31], 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.

2.2.3.1 *DARPA*

Der unter anderen von der *Defence Advanced Research Project Agency*, kurz DARPA, erstellte Datensatz KDD-99 [31] lieferte einen den 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 [11, 48, 49]. Unter anderem beschreiben McHugh et al. [33], 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. [48] und Engen [9] 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 selbstlernende Systeme kann dadurch ein ungewollter Bias entstehen. Des Weiteren ist der Datensatz mittlerweile stark veraltet (1999/1998).

Auch als Priviledge Escalation bezeichnet

2.2.3.2 *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 [11] ist mittlerweile veraltet und kommt für eine weitere Betrachtung nicht in Frage, da er zusätzlich auch keine weiteren Kontextinformationen wie Thread IDs enthält [7].

2.2.3.3 *ADFA-LD*

Der ADFA Linux Dataset (ADFA-LD) wurde von Creech et al. [7] 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 [2]. Hinzu kommt, dass lediglich Sequenzen von System Call IDs aufgezeichnet wurden und damit keine Metadaten im Datensatz enthalten sind.

2.2.3.4 *NGIDS-DS*

Der 2017 erstellte Datensatz NGIDS [17] 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 [14].

2.2.3.5 *LID-DS*

2019 veröffentlichten Grimmer et al. [14] das *Leipzig Intrusion Detection-Data Set* (LID-DS). Sie erkannten, dass die bisherigen Datensätze entweder veraltet oder nicht ausreichend waren um zum Beispiel Thread IDs oder System Call Argumente für ein IDS zu verwenden. Er wurde auf einem modernen Betriebssystem Ubuntu 18.04 aufgenommen und besteht aus 10 verschiedenen Systemen auf welchen jeweils ein Angriffszenario 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. Für die spätere Suche nach sinnvollen zusätzlichen Informationen eines System Calls lohnt es sich hier ein wenig detaillierter auf die in Tabelle 2.2 beschriebenen Szenarien zu schauen. Auffällig ist zunächst das Bruteforce-CWE-307 Szenario. Dabei wird ein Bruteforce Angriff durchgeführt in welchem in kurzer Zeit diverse Anmeldeveruche stattfinden. Hier unterscheiden sich die Abläufe der System Calls zwischen Normal- und Angriffsverhalten nur durch die auftrentende Frequenz. Irgendwas über Return Values?

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

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 [14]

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 soll zunächst die beschreibende *runs.csv* betrachtet werden und anschließend in Tabelle 2.4 eine Beispieldatei einer Aufzeichnung.

runs.csv					
image_name	scenario_name	is_executing_exploit	warmup_time	recording_time	exploit_start_time
actual_name	file_name_0001	False	10	35	-1
actual_name	file_name_0002	True	10	40	15

Table 2.3: Ausschnitt der runs.csv des LID-DS [14]

PROBLEM DES DATENSATZES 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 Negatives*, falls kein Alarm vorliegt, da in den normalen Dateien keine Angriffe stattfinden. Oder falls fälschlicherweise ein Angriff erkannt wird, die Zuordnung als *False Positive*. Die Zuordnungen der Dateien mit Angriff werden in Abbildung 2.3 dargestellt.

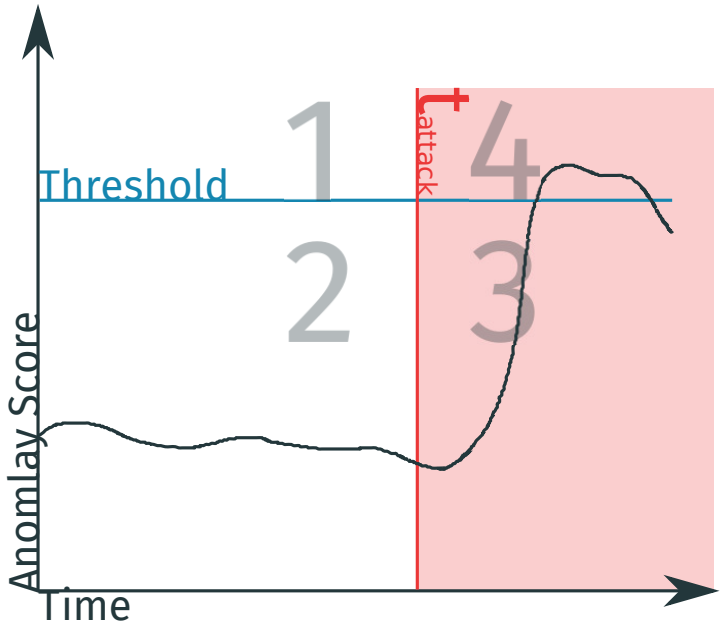

Figure 2.3: Illustration des Quadrantenproblems

Befindet sich der Anomaliescore vor dem Angriffszeitpunkt $t_{angriff}$ unter dem Schwellwert, also im Quadrant 2 der Abbildung, kann von einem *True Negative* ausgegangen werden. Also es wurde korrekterweise kein Alarm vorhergesagt. Befindet sich der Anomaliescore vor dem Angriffszeitpunkt $t_{angriff}$ über dem Schwellwert, also im Quadrant 1, liegt ein *False Positive* 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 nach dem Angriffszeitpunkt über dem Schwellwert, also im Quadranten 4, wird von einem *True Positive*, also einem korrektem Alarm ausgegangen. Jedoch könnte der Angriff zu diesem Zeitpunkt schon vorbei sein. Also könnte ein Alarm nach $t_{angriff}$ theoretisch auch ein *False Positive* sein. Genauso könnte umgekehrt ein Anomaliescore unter dem Schwellwert korrekt sein obwohl er als *False Negative* gewertet werden müsste. Diese Problematik muss in der Auswertung beachtet werden und wird in Abschnitt 4.5 beleuchtet.

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 Forschungsbere-

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

Table 2.4: Ausschnitt aus den eigentlichen Aufzeichnungen von System Calls aus dem LID-DS [14]

ichen zum Einsatz. Auch zum Beispiel in direkter Umsetzung für Oberflächenstrukturen welche häufig in der Raumfahrt eingesetzt werden [53], oder auch Abstrakter in der Umsetzung von Verhaltensweisen von Ameisen [36]. 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 [35, 43, 51].

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*,

zu dt. Ebene

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.

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

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

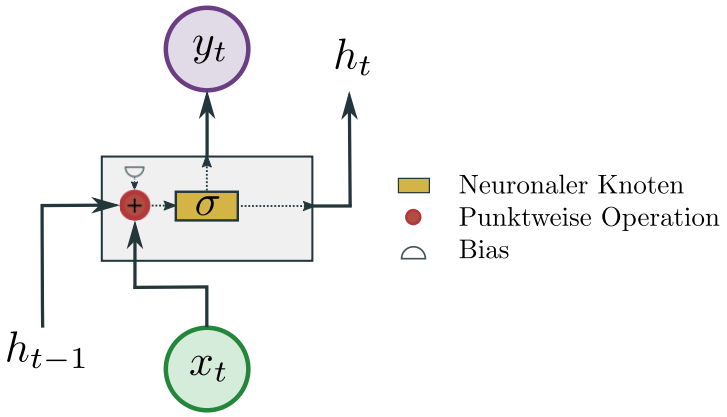

Figure 2.4: Darstellung einer RNN Zelle

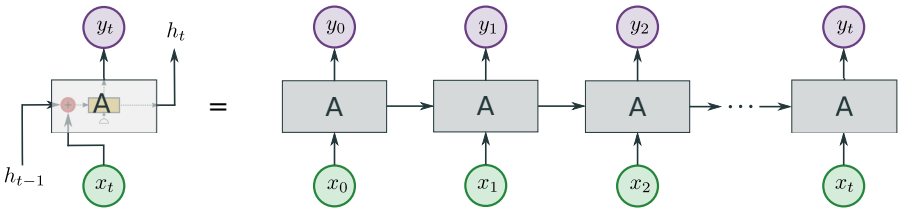

Figure 2.5: Darstellung einer „ausgerollten“ und vereinfachten RNN Zelle

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 Ab-

stand 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. [19]

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 sein könnten.

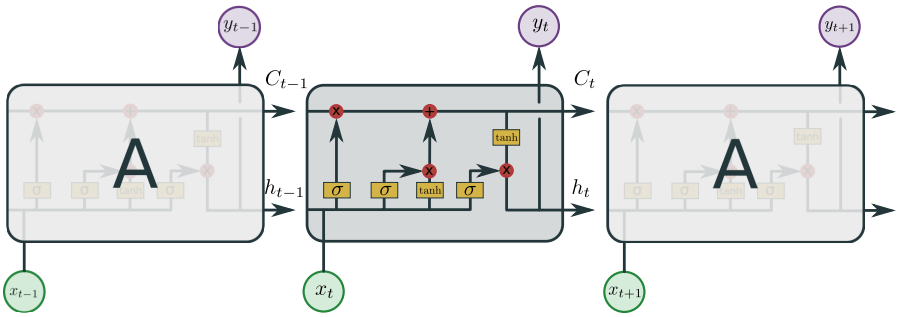

Figure 2.6: Schematische Darstellung eine Memory Cell A in einem LSTM NN, mit Input, Output und Forget Gate (inspiriert von [41]).

Und um zu erlernen welche früheren Ausgaben für die Ermittlung nächster Datenpunkte entscheidend sind, wird an jedem Knoten eine mit A in der Abbildung **??** 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

zu dt. Gatter/Tore

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

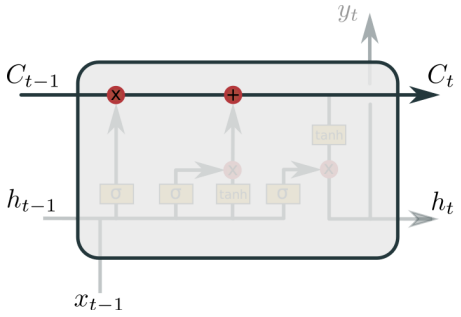

Figure 2.7: Weitergabe des Zellstatus innerhalb eines Knotens (inspiriert von [41]).

wird mit dem *Forget Gate* umgesetzt und ist in Abbildung 2.8 zu sehen und kann analog zur RNN Zelle folgendermaßen hergeleitet werden.

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

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 könnte das Genus (*grammatikalisches Geschlecht*) gespeichert werden, 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.

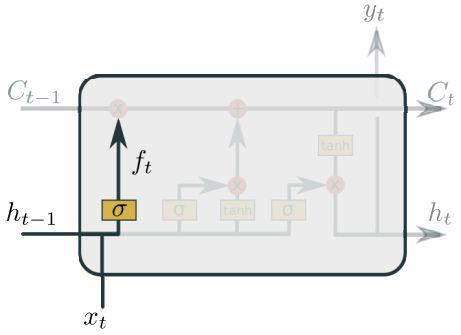

Figure 2.8: Einfluss des Forget Gates auf den Zellstatus (inspiriert von [41]).

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.

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

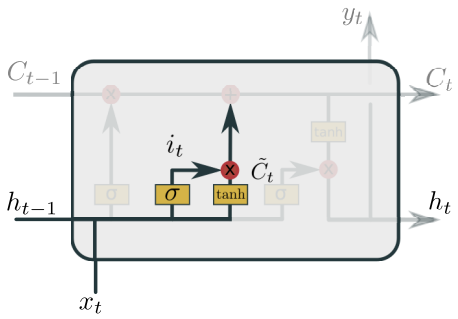

Figure 2.9: Einfluss des Input Gates auf den Zellstatus (inspiriert von [41]).

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. [13, 41]

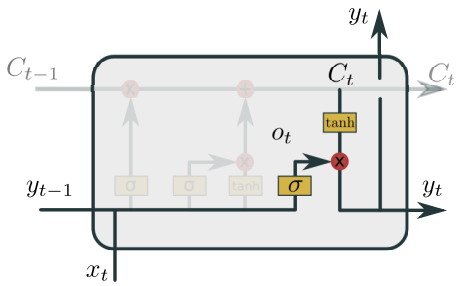

Figure 2.10: Das Output Gate regelt den Einfluss des Zellstatus auf die Ausgabe des Neurons (inspiriert von [41]).

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

3.1 ANOMALIEDETEKTION MIT SYSTEM CALLS

- Anfänge:
  - TIDE [10]
  - STIDE [21]
  - STIDE Alternatives [52]
- Fortschritte
  - System Call Graphs [15]
  - Frequency [1], [23]
  - Im Cloud/vm Einsatz [8]
  - System States [39]

3.1.1 *Erweiterung der System Call Sequenzen*

- extra models for syscalls [26] extended [32]
- further [40]

3.2 UNTERSUCHEN VON ZEITREIHEN MIT LSTM

3.2.1 *System Call mit LSTM*

- RNN [6]
- weiteres

Something [25]

4 REALISIERUNG

Bei der Realisierung einer AIDS müssen wie im Grundlagenkapitel 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 LSTMs 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 könnte, 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 mit entscheidende. Abschließen soll dieses Kapitel dann in Abschnitt **??** 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

In dem kommenden Abschnitt 4.2.1 soll zunächst auf die Vorverarbeitung des Datensatzes eingegangen werden. Anschließend soll in Abschnitt 4.2.2 untersucht werden, welche Darstellungsformen für die eigentlichen System Calls interessant und sinnvoll sind. Da, wie zuvor beschrieben, in dieser Arbeit ein Datenstream betrachtet werden soll, wird in Abschnitt 4.2.3 die Frage wie dieser für das LSTM dargestellt geprüft. Ein weiterer wichtiger Teil der Arbeit, neben dem Untersuchen ob sich LSTMs für die Anomalieerkennung bei System Calls eignen, besteht darin, 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 Infor-

mationen dafür verwendet werden können und wie diese dargestellt werden, soll in Abschnitt 4.2.2.1

4.2.1 *Vorverarbeitung des Datensatzes*

Gegeben 10 szenarien die aus bekannten CVEs bestehen. mit ca. 1000 files durschnittlich 45sec in runs.csv genauere beschreibung files mit label und zeitangabe falls exploit falls kein exploit dann exploit start time -1 keine dauer des exploits also ende nicht bekannt nicht systemcall genau start des angriffs angegeben führe puffer ein, da angegebener Zeitpunkt ungenau, sodass auch wirklich jeder angriff nach exploit start time alles nach dem angrffszeitpunkt muss als anomalie gewertet werden, auch wenn angriff evtl noch nicht gestartet hat oder schon vorbei. filtern von switch statements in Datensatz weil keine system calls nur schließende syscalls keine öffnenden. Muss wie in 2.2 beschrieben immer beide geben. Schließende interessant, da auch Rückgabewerte betrachtet werden können

4.2.2 *Wie wird ein System Call dargestellt?*

Neuronale netze benötigen numerische werte deswegen umwandlung von stringnamen sys to int nur bedingt brauchbar für netz auf Grund aktivierungsfunktion –> 2>1 3>1 mittelwert 2 Darstellung von kategorischen Daten für neuronale Netze bekanntes Problem, Auch in der Spracherkennung, hier ist allerdings auch ein ohe vorstellbar, da begrenzte Anzahl an versch System calls.

ONE-HOT-ENCODING Beschreibung OHE Eine weitere Möglichkeit System Calls darzustellen wird ebenfalls in der Spracherkennung verwendet. Durch *Word embeddings* kann zusätzlich zu der Kategorie auch noch der Kontext des Wortes in der Repräsentation eingebunden werden.

WORD-TO-VECTOR Beschreibung W2v
Nachdem nun eine Darstellung eines System Calls besprochen wurde muss auch noch auf die Streamverarbeitung eingegangen werden.

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

SYSTEM CALL ARGUMENTE most frequent calls Sql: vfrom, fcntl, lstat, nmap, poll php: vfrom, fcntl, lstat, nmap, poll bru: writev,read, close, nmap, poll eps: open, read, fstat, mmap, brk zip: futex, write, getpid,protect,open 2019: statat,write, futex, stat, getpid 2014: writev,read, nmap, close,poll 2017: lwait,close,ollctl,write,writev 2018: statat,write, futex, getpid, stat

unique: vfrom, fcntl, lstat, nmap, poll, writev, read, close, open, fstat, mmap, brk futex, write, getpid, protect, statat, stat, lwait, ollctl

interesting arguments:

THREAD AWARENESS *Thread aware ngrams* syscalls welche nicht in trainingsdaten bekommen eine 0 ngramme thread aware bilden unbrauchbar bei ngram länge von 1 *Thread change flag* oder thread change flag

ZEITLICHE ABSTÄNDE VON SYSTEM CALLS Berechne größten Abstand zwischen zwei aufeinanderfolgende System Calls in den Trainingsdaten. Normiere folgende System Call Abstände mit diesem Maximum.

RETURN WERTE Anzahl unique return Werte, nur numerische Werte betrachtet, res=-11(<f>/path/to/some/file) wird als -11 interpretiert Bruteforce: 17192 2851 int, 14333 hex, 8 else 2012: 1897 193 int, 1698 hex, 6 2014: 16736 2706, 14022, 8 2017: 381 26, 354, 1 2018: 110 106, 0, 4 2019: 125 119 int werte und 6 else SQL EPS PHP Zipslip

Hinweis 3 Kategorien evtl kategorische Einteilung bei geringer Anzahl Mögliche Bedeutungen bei else: Meist Fehlermeldungen!!! stat: file status, sollte 0 bei success und -1 bei error, gefunden auch -2 ioctl: manipulate underlying device, normalerweise 0 bei success manchmal negative werte -1 bei error manchmal return als ausgabe parameter Kategorische Betrachtung sinnvoll HEX: mmap: map or unmap files into memory return pointer to mapped area brk: change data segment size return 0 on success -1 on error got hex value

Systemcalls welche Rückgabewerte haben:

ohe und w2v word embedding parameterwahl wichtig $sqrt(distinct)$ Threadid kodieren:

- use entity embedding for ThreadID [16]
- relationship between threads and reduce size (possible 1000 different threads)
- choose size of embedding -thumbrule $sqrt(uniquevalue)$

zeit kodieren

- use time delta of two different syscalls as new input

parameterlänge kodieren

- syscall to int: Wandle Syscall name in Integer um

  ohe of sysint: use ohe for every syscall n * (distinct calls + 1) eingabeneuronen

  w2v von syscall weniger neuronen und nähe von syscall!!!

- ngram bilden: Bilde entsprechend angegebenes n ngramm

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

4.2.3 *Wie wird ein Datenstream dargestellt?*

Erstellung von ngrammen bekannte Vorgehensweise. Aufteilen von Datenstream unbekannter Länge in feste Größe. Wird oft als *Streaming Window* oder *Ngram* bezeichnet. Nötig da LSTM Eingaben fester Größe benötigen.

4.3 ALGORITHMUS

LSTM Sprachmodell soll Wahrscheinlichkeit des nächsten System Calls vorhersagen, gegeben eines System Calls oder einer Sequenz von 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 $P\left(x_i | x_{1:i-1}\right)$ 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:

$$ascore = 1 - p \tag{4.1}$$

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 könnte. 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!

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 könnte 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.

zu dt. schädlich

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

alarm in quadrant —> image

5 ERGEBNISSE

5.1 LSTM ANSATZ

5.1.1 *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

EMBEDDING w2v bringt verbesserung: alles ohne thread nur w2v!!

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 EXTRA PARAMETER

6 FOLGERUNGEN

6.1 SCHLÜSSE

6.2 AUSBLICK

BIBLIOGRAPHY

[1] Amr S Abed, Charles Clancy, and David S Levy. „Intrusion detection system for applications using linux containers." In: *International Workshop on Security and Trust Management*. Springer. 2015, pp. 123–135.

[2] 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.

[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] 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.

[5] 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.

[6] Ashima Chawla, Brian Lee, Sheila Fallon, and Paul Jacob. „Host based intrusion detection system with combined CNN/RNN model." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2018, pp. 149–158.

[7] 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.

[8] Joel A Dawson, Jeffrey T. McDonald, Lee Hively, Todd R. Andel, Mark Yampolskiy, and Charles Hubbard. „Phase Space Detection of Virtual Machine Cyber Events Through Hypervisor-Level System Call Analysis." In: *2018 1st International Conference on Data Intelligence and Security (ICDIS)*. 2018, pp. 159–167. DOI: 10.1109/ICDIS.2018.00034.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] 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).

[16] Cheng Guo and Felix Berkhahn. „Entity embeddings of categorical variables." In: *arXiv preprint arXiv:1604.06737* (2016).

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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).

[26] 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.

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

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

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

[43] 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.

[44] 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.

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

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.