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 System

AIDS Anomaly-based Intrusion Detection System

LSTM Long-Short-Term-Memory

NLP Natural Language Procession

FPR False Positive Rate

INHALTSVERZEICHNIS

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.

Neue Methoden werden benötigt um komplexere und neuartige Angriffe zu erkennen.

Erkennung von Angriffen Intrusion Detection System (IDS).

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 hinzugefügt werden. Ein wesentlicher Vorteil liefert hier die Angriffserkennung über Anomalien. Im Gegensatz zu dem erwähnten signaturbasierten Ansatz, muss dabei 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. Eine der größten Herausforderungen die beim Einsatz von Anomalieerkennungen auftreten, besteht darin die Anzahl der Fehlalarme auf einem geringen Niveau zu halten. So kann bei großen Datenmengen schon eine sehr geringe Fehlalarmrate zu einer für einen Analysten nicht zu bewältigenden Aufgabe werden. [4] In dieser Arbeit ein Host-based Intrusion Detection System (HIDS) entwickelt werden, dabei werden

die Daten auf dem zu überwachenden *Host* abgegriffen werden. Im Gegensatz dazu stehen zum Beispiel die Network Intrusion Detection System (NIDS), hier werden die Daten auf Netzwerkebene und damit auch von mehreren Hosts überwacht. HIDS bieten Vorteile durch ihre feingranularität und bieten die Möglichkeit auch interne Attacken erkennen zu können. Im realen Einsatz von HIDS besteht eine Schwierigkeit darin, dass das IDS Zugriff auf den Kernel des zu überwachenden Systems benötigt. Diese und weitere Herausforderungen die sich mit der praktischen Umsetzung in einem potentiellen Betrieb entstehen können sollen allerdings in dieser Arbeit nicht behandelt werden, da lediglich die Algorithmen selbst untersucht werden sollen.

Doch neben der Frage wo die Daten abgegriffen werden, muss geklärt werden welche Daten genutzt werden sollen. Welche Daten ermöglichen es das dem System zugrunde liegene Verhalten zu beschreiben. Dabei sollten sie zusätzlich so abstrakt sein, dass sie in vielen Systemen ihren Einsatz finden. Eine häufig verwendete Information für die Charakterisierung von Systemen bieten zum Beispiel System-Logs [14].

In dieser Arbeit werden System-Calls verwendet. Sie bieten eine sehr abstrakte Betrachtung auf Linux Betriebssystemebene. Die Grundidee besteht darin, dass Programme auf einer Festplatte meist erst Schaden anrichten, sobald sie ausgeführt werden. Dabei führen sie betriebssystemspezifische System-Calls aus, welche über verschiedene Tools wie zum Beispiel Sysdig [30] 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. Generell konnten Probleme in der Verarbeitung von sehr großen Datenmengen unter anderem durch die Verwendung selbst lernender Algorithmen erfolgreich angegangen werden. Es gilt also in großen Datenmengen Zusammenhänge zwischen potentiell auseinanderliegenden Abfolgen von System-Calls zu erlernen. Erfolgreich haben sich dabei Long-Short-Term-Memory (LSTM) Netzwerke gezeigt. Sie haben den Vorteil auch Zusammenhänge mit größerer zeitlicher Verzögerung noch zu erkennen [16] und können in unterschiedlichsten Architekturen einen Nutzen bringen. Da die LSTM Netzwerke auch in der Spracherkennung erfolgreich sind [37], ensteht die Frage in wie weit auch andere Erkenntnisse aus der Welt der Natural Language Procession (NLP) sich in den Bereich der Analyse von System-Calls übertragen lassen.

In verschiedenen Arbeiten wurde bereits die Abfolge von System-Calls betrachtet, um darin Anomalien zu erkennen. Doch nur in wenigen Arbeiten werden auch die Parameter zur Anomalieerkennung verwendet. Eine der ersten Arbeiten von Forrest et al. [7] betrachtet lediglich die Sequenzen der System-Calls. Grimmer et al. versuchen die Bildung der Sequenzen durch Betrachtung der Thread-ID anzupassen [10], dies bindet Zusatzparameter zur Verbesserung der Sequenz-

bildung ein, aber dennoch werden „nur “ Sequenzen betrachtet. Maggi et al. verwenden zusätzlich auch Parameter und verweisen in ihrer Arbeit [23] auf diverse Ansätze. In dieser Arbeit soll ebenfalls versucht werden durch die Hinzunahme von Parametern, wie zum Beispiel den Rückgabewerten (sofern vorhanden), die Erkennungsquote bzw. die False Positive Rate (FPR) des IDS zu verbessern.

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 [20] 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 sollen nun Grundlagen betrachtet werden, welche bei der Umsetzung des IDS nötig sind. Essentiell dabei sind zum einen die Definition von Anomalien und der Anomaliedetektion (siehe Kapitel 4.3.1) und eine allgemeine Einführung in den Bereich der IDS (siehe Kapitel 2.1). Dabei wird auf den im praktischen Teil verwendeten Ansatz der *Host-Based Intrusion Detection Systems* (HIDS) genauer eingegangen. Nachdem so eine allgemeine Herangehensweise an die Erkennung von Angriffen dargelegt wurde, soll im Anschluss die Grundlagen des verwendeten Algorithmus untersucht werden. Dazu gehören rekurrente neuronale Netze (RNN), sowie die Erweiterung der RNNs die *Long Short-Term Memory* neuronalen Netzen (LSTM).

2.1 INTRUSION DETECTION SYSTEM

Eine *Intrusion* ist eine unauthorisierte Aktivität, welche einem System Schaden zufügen kann. Eine *Intrusion Detection* Software/Hardware versucht automatisch diese unauthorisierten Aktivitäten zu identifizieren, um dadurch die Sicherheit des Systems gewährleisten zu können. [21] Um solche unautorisierte Aktivitäten erkennen zu können müssen zunächst Daten erfasst und anschliessend analysiert werden. In den folgenden beiden Abschnitten werden diese Schritte genauer untersucht.

zu dt. Eindringung

2.1.1 *Datenanalyse*

Das Erkennen von Angriffen auf Computersystemen mittels IDS wird meist, wie auch in [21] und [17], in drei Kategorien eingeteilt. Dazu zählen die signaturbasierten und anomaliebasierten Verfahren auf die im Folgenden eingegangen wird, sowie die *Stateful Protocol Analysis* .

zu dt. Zustandsorientierte Protokoll Analyse

SIGNATURBASIERT Signature-based Intrusion Detection System (SIDS) versuchen Muster von bekannten 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 auch hier diverse Methoden wie diese Signaturen erstellt werden. Dazu zählen zum Beispiel das Betrachten von *Host-Logs*, oder ein wenig ausgefeilter das Erstellen von *State Machines* [24] Doch ein wesentlicher Nachteil der signaturba-

sierten IDS liegt in der Tatsache, dass keinerlei neuartigen Angriffe, sowie viele Abwandlungen von Angriffen nicht erkannt werden, da diese noch nicht in der Datenbank vorhanden sind. [17]

ANOMALIEBASIERT Ziel einer Anomaly-based Intrusion Detection System (AIDS) ist es Muster in Daten wiederzuerkennen welche von einem definierten Normalverhalten abweichen [3]. Die Bedeutung der Anomalieerkennung liegt in der Tatsache begründet, dass Anomalien in Daten zu signifikanten und oft kritischen Veränderungen eines System führen können. So kann eine Anomalie in Netzwerkdaten dafür stehen, dass ein gekaperter Computer sensitive Daten an ein unautorisiertes Ziel sendet [19]. 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 definierten Normalverhalten zeigen und damit interessant für Analysen sind. Genau dieser signifikante Unterschied des aktuellen Systemzustand zu dem entworfenen Modell (wohldefiniertes Normalverhalten) zu einem Zeitpunkt t wird dann als Anomalie eingestuft. Jede Anomalie gilt dann wiederrum als Intrusion. Grundannahme dieser Methode liegt darin, dass Intrusions von dem gelernten Normalverhalten des Systems unterschieden werden können.

Im ersten Schritt der AIDS wird ein Modell des Normalverhaltens erstellt. Dabei können ML-basierte, Statistik-basierte oder auch *Knowledge*-basierte Algorithmen verwendet werden. Weiter kann die anomaliebasierte *Intrusion Detection* in zwei Phasen aufgeteilt werden, die *Trainingsphase* sowie die *Testphase*. In der ersten Phase wird versucht ein Modell des Programmverhaltens zu ermitteln, beziehungsweise zu trainieren. In der zweiten soll 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 einen Anomaliescore. Liegt dieser über einem festgelegten Schwellwert so gilt die Eingabe als „anormal“ ansonsten als „normal“. Die Trainingsphase wird von Chandola et al. [3] weiter unterteilt in *Supervised*, *Semisupervised* sowie *Unsupervised Anomaly Detection*. Diese unterscheiden sich hauptsächlich in der Anforderung an den Datensatz. Bei einer *Supervised* Anomalieerkennung werden Trainingsdaten benötigt in welchen Anomalien sowie auch Normalverhalten gelabelt sind. Hingegen wird beim *Semisupervised* Verfahren lediglich ein Datensatz benötigt, welcher das Normalverhalten kennzeichnet. 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 [28]. Bei der Umsetzung der Anomaliedetektion ergeben sich allerdings einige Schwierigkeiten:

zu dt. Wissens-basierte

- *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 [3].
- *Datensätze*: Gelabelte Daten zur Erfassung des Normalverhaltens sind oft veraltet oder nicht sehr detailreich.
- *Schwellwert*: Festlegung eines Schwellwertes, welcher die eigentliche Unterscheidung zwischen Normalverhalten und Angriffsverhalten umsetzt.

Mehr dazu in Abschnitt 2.3

Auch wenn sich die verschiedenen Teilbereiche der **AIDS !** (**AIDS !**)stark unterscheiden, haben sie einen wesentlichen Vorteil gegenüber den SIDS. Denn ihnen ist es nicht generell verwehrt Zero-Day Angriffe zu erkennen, also Angriffe die in dieser Form noch nicht bekannt waren. Jedoch ergibt sich im Vergleich zu den SIDS eine andere Problematik. Denn die AIDS müssen einen Schwellwert festlegen ab welchem ein Vorkommnis als Anomalie erkannt wird. Dieser Wert entscheidet über die Ergebnisqualität der AIDS und sollte daher sorgfältig und am besten automatisch ermittelt werden, um anwendungsspezifische Justierungen zu vermeiden. Eine weitere Schwierigkeit bei AIDS liegt in der Ermittlung von Daten für das Normalverhalten und die damit verbundene Gefahr, dass bei nicht kompletten Erfassen des Normalverhalten viele Fehlalarme enstehen können. Daher ist eine entscheidende Frage bei Verwendung von AIDS: Woher kommen die für das Lernen des Normalverhalten benötigten Daten?

mehr dazu in Kapitel 4.5

2.1.2 *Datenerfassung*

Die Datenerhebung wird in verschiedenen Arbeiten in zwei Kategorien unterteilt [17], [21]. Zum einen die HIDS und zum anderen die NIDS In den folgenden Abschnitten werden diese beiden Ansätze genauer untersucht.

NETWORK BASED INTRUSION DETECTION Überwacht den Netzwerkverkehr über Pakete, NetFlow oder andere Netzwerkdaten. Ein großer Vorteil daran ist, dass viele Computer in einem Netzwerk überwacht werden. Ziel ist es dabei Angriffe möglichst früh zu erkennen und zu verhindern, dass sich die Gefahr weiter ausbreiten kann. Doch der erwähnte Vorteil kann schnell zu Schwierigkeiten führen, da bei besonders großen Netzen der hohe Datendurchsatz das Erkennen von Angriffen erschwert. [2] 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 [17], oder aber auch Daten aus dem Kernel wie z.B. System Calls [23]. Im Gegensatz zur NIDS kann hier auf den Inhalt von jeder Information eingegangen werden, da die interne Kommunikation unverschlüsselt stattfindet.

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 sowie die *Stateful Protocol Analysis* 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, ...
- Wie wird die eigentliche Anomalie in den Daten erkannt?

Die erste Frage soll in dem folgenden Abschnitt mit der Betrachtung von System Calls zur Beschreibung des Systemverhaltens angegangen werden. Die letztere soll dann speziell in Kapitel 4.5 beleuchtet 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. Zu ihnen gehören zum Beispiel die in Tabelle **??** beschriebenen System Calls.

zu dt. Systemaufrufe

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 von Programmen genutzt werden

zu dt. Betriebssystemkern

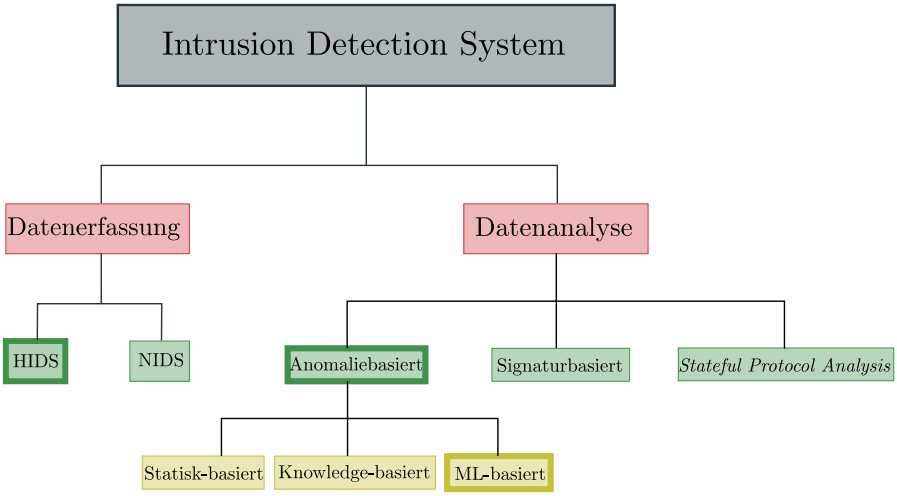

Abbildung 2.1: Einordnung des verwendeten IDS (breiter markiert).

System Calls			
Name	Beschreibung	Argumente	Rückgabewerte
open	Öffnet die in *path* spezifizierte File und gibt einen *file descriptor* zurück.	*path*, *flags*, *mode*	kp
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*	kp

Tabelle 2.1: Beschreibung ausgewählter System Calls

können. Die System Calls stellen dabei die Kommunikation zwischen Kernel und Programmen dar. Mögliche Services sind unter anderem Zugriffe auf Hardware aber auch das Erstellen und Ausführen von Prozessen. Üblicherweise können System Calls nur von Nutzerprozessen ausgelöst werden, welche eine eingeschränkte Berechtigung besitzen. Wie in Tabelle **??** ebenfalls beschrieben besitzen System Calls auch Argumente die beim Aufrufen mitgegeben werden können sowie einen Rückgabewert.

Auch ein möglicher Angreifer muss um Schaden anzurichten also an der Art der System Calls eine Veränderung vornehmen. Entweder kann die Abfolge, also die Sequenz von System Calls verändert werden, z.B. es werden neue Funktionen aufgerufen, welche wiederum andere System Calls aufrufen, 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.

2.2.1 *System Calls für IDS*

Viele verschiedene IDS Ansätze betrachten lediglich die Sequenz von System Calls und missachten die in den Argumenten enthaltene Informationen. Auch wenn diese Ansätze teilweise sehr erfolgreich sind,

lassen sie den Angreifenden einen Spielraum. Verschiedene Arbeiten [34], [31], [32] zeigen, wie dieser Spielraum ausgenutzt werden kann um unerkannt Angriffe durchzuführen. Tan et al. [32] erreichen dies durch die Veränderung eines zuvor von **der !** (**der !**)IDS erkannten Angriffes. Sie beschreiben, wie **dem !** (**dem !**)IDS fremde System Call Sequenzen derart Verändert werden können, dass sie als normal eingestuft werden. Dabei werden die 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 [34]. Doch diese Beispiele zeigen auch, dass wenigstens ein Faktor, also Sequenz oder Argumente, verändert werden muss, um das Systemverhalten abzuwandeln. Welche dieser Argumente und wie diese genutzt werden können um die Anomalieerkennung zu verbessern soll in Kapitel **??** untersucht werden. Im dem nachfolgenden Kapitel wird nun auf verschiedene Datensätze, welche System Call Sequenzen enthalten, eingegangen.

Verwendeter Algorithmus: STIDE [7]

2.3 DATENSÄTZE

Seit 1998 einer der ersten System Call Datensätze für HIDS veröffentlicht wurde [22], kamen verschiedene 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.

KDD Der unter anderen von der *Defence Advanced Research Project Agency*, kurz DARPA, erstellte Datensatz KDD-99 [22] simuliert ein militärisches Netzwerk bestehend aus drei Systemen mit unterschiedlichen Betriebssystemen und Services. Diese Systeme erzeugen mit wechselnden IP-Adressen Traffic, welcher sieben 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 [6] [33] [8]. Zum einen ist der Datensatz veraltet (1999) und zum anderen gibt es Diskrepanzen in den Daten, wie unter anderem von Vegard Engen beschrieben wird [6].

Auch als Priviledge Escalation bezeichnet

UNM Der *University of New Mexico* Datensatz 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 [8] ist veraltet (2004) und kommt für eine weitere Betrachtung nicht in Frage, da er zusätzlich auch keine weiteren Kontextinformationen wie Thread IDs enthält [5].

ADFA-LD Der *ADFA Linux Dataset* (ADFA-LD) wurde von Creech et al. [5] 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 [13] 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. Des Weiteren ergeben sich Schwierigkeiten in der Zuordnung von der beschriebenen Event ID und den Zeitstempeln [**Grimmer**].

LID-DS 2019 veröffentlichten Grimmer et al. [11] 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 Szenarien. Jedes Szenario repräsentiert dabei eine bekannte Schwachstelle eines Systems. In Tabelle **??** 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 System Calls Namen auch deren Metadaten. Dazu zählen unter anderem die Parameter, Rückgabewerte, Zeitstempel sowie User-, Prozess- und Thread-IDs. 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 werden.

zu dt. Allgemeine Schwachstellen und Gefährdungen

zu dt. Aufzählung gemeinsamer Schwachstellen

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

Szenarien	
Name	Beschreibung
Bruteforce-CWE-307	Ungeeignete Einschränkung von übermäßigen Authentifizierungsversuchen
CVE-2012-2122	Umgehung von MySQL Authentifizierung
CVE-2014-0160	Heartbleed: Schwachstelle in OpenSSL
CVE-2017-7529	Nginx Integer Overflow
CVE-2018-3760	Sprockets Datenleck Schwachstelle
CVE-2019-5418	Rails Fileinhalt Offenlegung
EPS_CWE-434	EPS file upload: uneingeschränkter Upload von gefährlichen Dateitypen
PHP_CWE-434	PHP file upload: uneingeschränkter Upload von gefährlichen Dateitypen
SQL Injection	SQL Injection mit sqlmap
ZipSlip	blablabla

Tabelle 2.2: Kurzbeschreibung der in dem Datensatz vorkommenden Sicherheitslücken [11]

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

Tabelle 2.3: Ausschnitt der runs.csv des LID-DS [11]

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.

2.4 KÜNSTLICHE NEURONALE NETZE

Das Nutzen von bestehenden und in der Natur vorkommenden Strukturen und Prozessen kommt in verschiedenen Forschungsbereichen zum Einsatz. In direkter Umsetzung von zum Beispiel der Struktur von Geckofüßen für Oberflächenstrukturen welche häufig in der Raumfahrt eingesetzt werden [36]. Auch Prozesse welche in der Natur beobachtbar sind werden versucht zu adaptieren, dazu zählen zum Beispiel Ameisenalgorithmen [26]. Dabei wird versucht das Verhalten von realen Ameisen zu modellieren um verschiedene Optimierungsprobleme zu lösen. 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

System Call								
event number	event time	cpu	user uid	process name	thread id	event direction	event type	event arguments
4012	10 : 18 : 20.1231231231	0	33	apache2	1425	>	writev	$fd = 12(<4t>$ 172.131.12.1 : 123 $\rightarrow$ 172.13.231.2 : 123)$size =$ 2392

Tabelle 2.4: Ausschnitt aus den eigentlichen Aufzeichnungen von System Calls aus dem LID-DS [11]

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 [29] [35] [25].

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 *Recurrent Neural Nets* (RNN). Da in dieser Arbeit sequentielle Daten betrachtet werden, die sequentielle Abfolge von System Calls, soll im Folgenden nur auf die RNNs und besondere Abwandlungen dieser eingegangen werden.

2.4.1 *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 kei-

zu dt. Ebene

nen 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 2.2 dargestellt, sowie in Abbildung 2.3 in einer alternativen Darstellung, wie sich dieser Knoten A über t Zeitpunkte verhält. Dabei werden für die Übersichtlichkeit Gewichte sowie der Bias nicht dargestellt. Doch auch die RNNs haben ein Problem bei

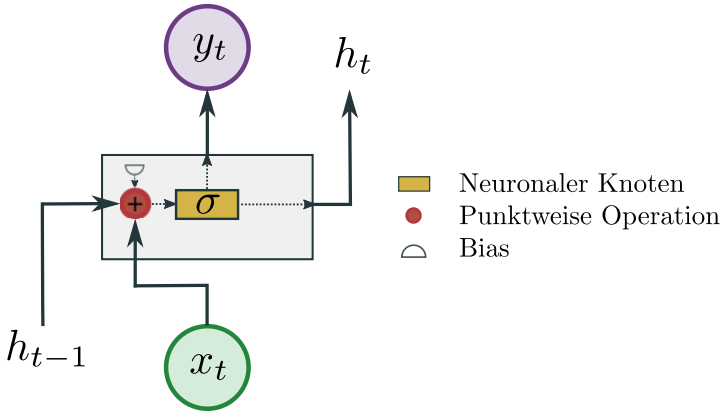

Abbildung 2.2: Darstellung einer RNN Zelle

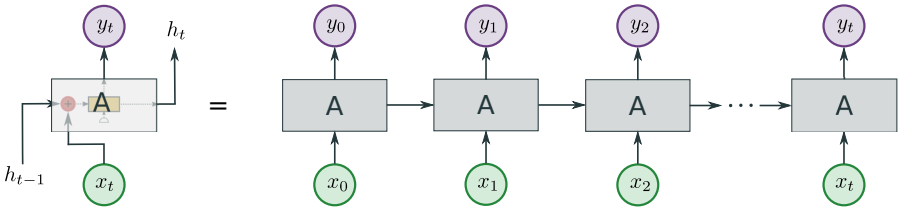

Abbildung 2.3: Darstellung einer „ausgerollten“ und vereinfachten RNN Zelle

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. [15]

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

2.4.2 *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.

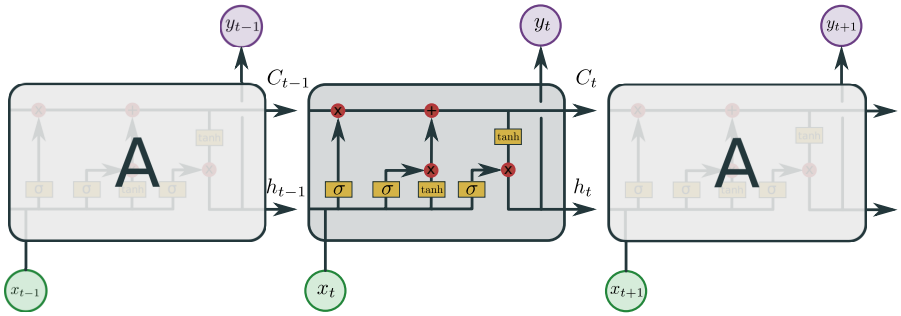

Abbildung 2.4: Schematische Darstellung eine Memory Cell A in einem LSTM NN, mit Input, Output und Forget Gate (inspiriert von [27]).

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 2.5 dargestellt.

zu dt. Gedächtniszelle

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 2.6 zu sehen und kann analog zur RNN Zelle folgendermaßen hergeleitet werden.

zu dt. Gatter/Tore

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

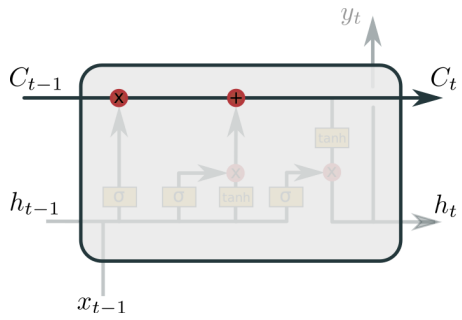

Abbildung 2.5: Weitergabe des Zellstatus innerhalb eines Knotens (inspiriert von [27]).

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.

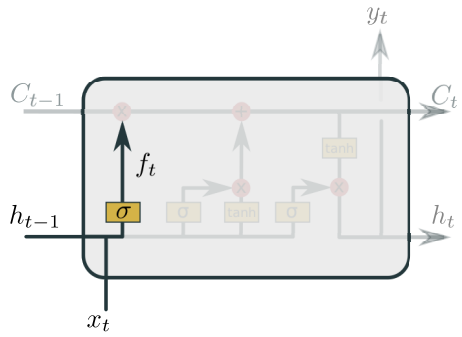

Abbildung 2.6: Einfluss des Forget Gates auf den Zellstatus (inspiriert von [27]).

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. Abbil-

dung 2.7). Der gesamte Zellstatus C_t wird dann zusammen mit Werten aus dem *Forget Get* verrechnet.

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

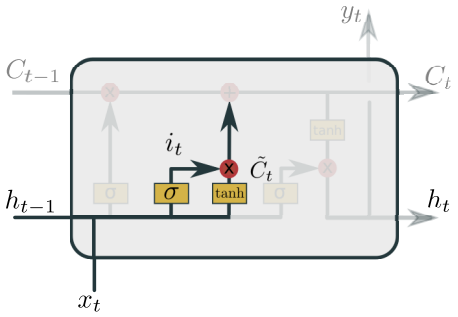

Abbildung 2.7: Einfluss des Input Gates auf den Zellstatus (inspiriert von [27]).

Wie der Zellstatus C_t nun die Ausgabe beeinflusst, wird über das *Output Gate* geregelt (siehe Abbildung 2.8).

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

Dies soll in unserem Sprachbeispiel entscheiden, ob die Information des Genus für die Vorhersage des nächsten Wortes eine Rolle spielt. [9] [27]

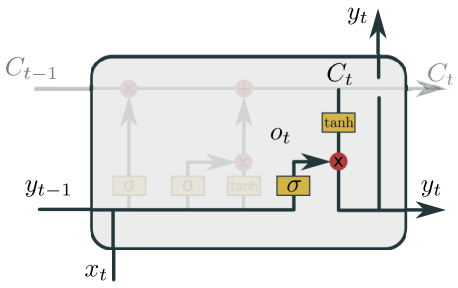

Abbildung 2.8: Das Output Gate regelt den Einfluss des Zellstatus auf die Ausgabe des Neurons (inspiriert von [27]).

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

3.2 UNTERSUCHEN VON ZEITREIHEN MIT LSTM

3.3 SYSTEM CALL MIT LSTM

Something [18]

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 2.3 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 4.5 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 4.2.4

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.

4.2.3 *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.

4.2.4 *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 [12]
- 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 ohe und w2v word embedding parameterwahl wichtig $sqrt(distinct)$ Threadid kodieren:
  - use entity embedding for ThreadID [12]
  - 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 ohe und w2v word embedding parameterwahl wichtig $sqrt(distinct)$ Threadid kodieren:
  - use entity embedding for ThreadID [12]
  - 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

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

4.3 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 $P(x_i | x_{1:i-1})$ für alle i.

4.3.1 *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.

4.3.2 *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.

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 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 4.7 beleuchtet.

4.3.3 *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

««««< HEAD 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

Nachdem nun eine Darstellung eines System Calls besprochen wurde muss auch noch auf die Streamverarbeitung eingegangen werden.

4.4.2 *Wie wird ein Datenstream dargestellt*

ngrams streamingwindow

4.4.3 *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:

4.4.4 *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

4.5 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.

4.5.1 *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:

$$ascore = 1 - p \tag{4.2}$$

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

4.5.2 *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.

4.6 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.

4.6.1 *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

4.6.2 *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

4.7 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 IMPLEMENTIERUNG

Verschiedene Komponenete um Modelle in Python umzusetzen zunächst soll auf vorverarbeitung eingegangen werden

5.1 ERSTELLEN DER TRAININGSDATEN

5.1.1 *w2v*

verschiedene varianten mit ignore thread info

5.1.2 *Parameterinfo*

6 ERGEBNISSE

6.0.1 *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

7

FOLGERUNGEN

LITERATUR

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

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

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

[4] Varun Chandola, Arindam Banerjee und Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), S. 1–58.

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

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

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

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

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

[10] Martin Grimmer, Tim Kaelble und Erhard Rahm. “Improving Host-based Intrusion Detection Using Thread Information”. In: ().

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

[12] Cheng Guo und Felix Berkhahn. “Entity embeddings of categorical variables”. In: *arXiv preprint arXiv:1604.06737* (2016).

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

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

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

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

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

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

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

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

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

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

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

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

[25] Patricia Melin, Julio Cesar Monica, Daniela Sanchez und Oscar Castillo. "Multiple Ensemble Neural Network Models with Fuzzy Response Aggregation for Predicting COVID-19 Time Series: The Case of Mexico". In: *Healthcare* 8.2 (2020). ISSN: 2227-9032. DOI: 10.3390/healthcare8020181. URL: https://www.mdpi.com/2227-9032/8/2/181.

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

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

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

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

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

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

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

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

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

[35] Peipei Wang, Xinqi Zheng, Gang Ai, Dongya Liu und Bangren Zhu. "Time series prediction for the epidemic trends of COVID-19 using the improved LSTM deep learning method: Case studies in Russia, Peru and Iran". In: *Chaos, Solitons & Fractals* 140 (2020), S. 110214.

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

[37] Lirong Yao und Yazhuo Guan. "An Improved LSTM Structure for Natural Language Processing". In: *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*. 2018, S. 565–569. DOI: 10.1109/IICSPI.2018.8690387.