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. . .

CONTENTS

1 EXPOSEE

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

Angriffe auf Computersysteme bestehen schon seit diversen Jahren. Die häufig verwendeten auf Blacklists basierenden Abwehrmechanismen reichen nicht aus um viele drohende Gefahren abzuwenden. Das liegt hauptsächlich daran, dass weder Abwandlungen von Angriffen, noch unbekannte Angriffe erkannt werden können. Ein wesentlicher Vorteil liefert hier die Angriffserkennung über Anomalien. Im Gegensatz zu dem erwähnten Blacklist Ansatz, muss nicht jeder Angriff der abgewehrt werden soll bekannt sein. Stattdessen wird versucht das Normalverhalten eines Systems zu ermitteln und jegliche Abweichung als Anomalie einzustufen. Nun bieten verschiedene Systeme verschiedene charakteristische Merkmale um das Verhalten zu beschreiben. Eine häufig verwendete Information für die Charakterisierung bieten zum Beispiel System-Logs [5].

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 [16] 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 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 [2] betrachtet lediglich die Sequenzen der System-Calls. Maggi et al. verwenden zusätzlich auch Parameter und verweisen in ihrer Arbeit [11] 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 [8] 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 kann. Zum anderen müssen die Grundlagen für den verwendeten Algorithmus gelegt werden. Dazu gehören Grundlagen zu rekurrenten neuronalen Netzen (RNN) sowie die Erweiterungen der LSTM Netzwerke.

Ein großer Teil der Implementierungsarbeit wird die Vorverarbeitung der Daten darstellen. Diese soll mit der genaueren Untersuchung des gesamten Algorithmus in einem weiteren Kapitel dargestellt

werden. Nachdem die verwendete Software analysiert wurde, wird eine Auswertung auf dem LID-DS [9] 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 VERWANDTE ARBEITEN

2.1 ANOMALIEDETEKTION MIT SYSTEM CALLS

2.2 UNTERSUCHEN VON ZEITREIHEN MIT LSTM

3 GRUNDLAGEN

Grundlagen die untersucht werden Intrusion Detection System (IDS) sowie die spezielle Form der RNNs die LSTM-NN Erster Abschnitt beschreibt verschiedene IDS Ansätze Hierbei auch auf die verwendete Anomalieerkennung Dann soll der verwendete Datensatz beschrieben werden Abschließend LSTM

3.1 INTRUSION DETECTION SYSTEM

Erkennen von Angriffen auf Computersysteme

3.1.1 *Ansätze*

- Signaturbasiert
  - Whitelist
  - Blacklist

    aufzählen aller Angriffe
- Anomaliebasiert

  Definiere was normal ist

  Analogie menschliches Immunsystem → Auch NN Analogie zur Natur <++> Host Based, Network traffic, logging

3.1.2 *Host Based Intrusion Detection*

Überwacht system intern, zum beispiel Kernel –> System calls verbinung Kernel Programme

3.2 ANOMALIEERKENNUNG

Anomalien sind Muster die nicht einem wohldefinierten Normalverhalten entsprechen Ziel der Anomalieerkennung ist es Muster in Daten wieder zu erkennen Normalverhalten muss in Daten wiedergespiegelt werden überwacht –> gelabelt normalverhalten und anormales -> problem overfitting von anormalen verhalten unüberwacht semi überwacht –> nur normalverhalten, liegt hier vor aber gelabelt testdaten benötigt Schwellwert —> siehe Kapitel algorithm wird dieser überschritten Es gibt verschiedene Arten von anomalien

3.2.1 *System Calls*

System Call Arguments

STRING LENGTH

STRING CHARACTER DISTRIBUTION

STRUCTURAL INFERENCE

TOKEN FINDER

3.3 KÜNSTLICHE NEURONALE NETZE

Das Nutzen von bestehenden und in der Natur vorkommenden Strukturen und Abfolgen kommt in verschiedenen Bereichen zum Einsatz. Sehr bekannt dabei sind zum Beispiel — Ein in den letzten Jahren immer weiter verbreiteter Ansatz ist in der Informatik die Verwendung von künstlichen neuronalen Netzen, welcher sich die Struktur von Gehirnen zu eigen macht. Man verspricht sich mit dem Einsatz von künstlichen neuronale Netze, welche eine varietät von verschiedenen Architekturen beinhalten, diverse Optimierungsprobleme zu lösen. In dieser Arbeit wird der Einsatz von neuronalen Netzen zur *Time Series Prediction* (zu dt. Vorhersagen von Zeitreihen) untersucht. Für die *Time Series Prediction* sind verschieden Architekturen von neuronalen Netzen weit verbreitet [1]. Zu aktuellen Beispielen zählen dabei auch Vorhersagen die im Zusammenhang mit der Verbreitung des COVID-19 Virus stehen [15] [17] [12].

unterteilung verschiedener Netze und einordnung von rnns und lstmnn

3.3.1 *Long Short-Term Memory Recurrent Neural Network*

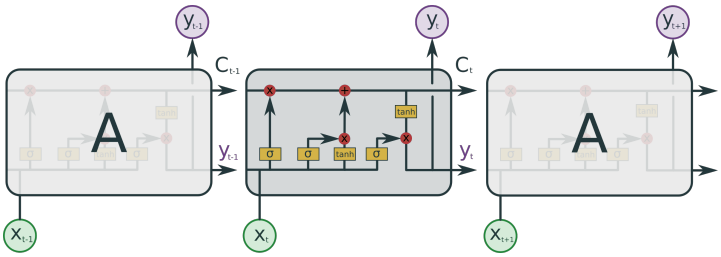

Figure 3.1: Schematische Darstellung eines Knotens in einem LSTM NN, mit Input, Output und Forget Gate (inspiriert von [14]).

Long Short-Term Memory (LSTM) ist eine Erweiterung der RNNs. Hauptziel der LSTMs ist es, das Lernen der zeitlich abhängigen Muster zu verbessern. Auch RNNs haben dieses Ziel, doch mit diesen Netzw-

erken kann schon ein Abstand von 10 diskreten Zeitschritten zwischen den abhängigen Ereignissen nicht überbrückt werden [6]. So kann ein RNN sofern es korrekt trainiert wurde mit hoher Wahrscheinlichkeit im Satz „Die Wolken am *Himmel*“ das Wort *Himmel* vorhersagen, doch bei der Satzfolge „Die Person kommt aus Frankreich. ... . Die Person spricht *französisch*.“ wird es Schwierigkeiten haben. Ein weitere Problemstellung mit RNNs ist das stabile trainieren. Dabei kommt es häufig vor, dass durch die Back Propagation die berechneten Gradienten entweder verschwindend klein, oder sehr groß werden. Gerade bei Abhängigkeiten über einen größeren zeitlichen Abstand tendieren die Fehlersignale, die durch die Back Propagation durch das Netz gegeben werden, zu geringe Gewichtsänderungen auszulösen [7]. Die LSTM Zelle ermöglichen durch verbesserte Fehlerkorrektur stabilere Lernergebnisse sowie auch das Lernen von Mustern mit noch größeren zeitlichen Abstände. Umgesetzt wird diese Anforderung, indem an jedem Knoten eine *Memory Cell* (Gedächtniszelle) angebracht wird 3.1. Sie ist mit sich selbst verbunden 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 dann 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 3.2 dargestellt.

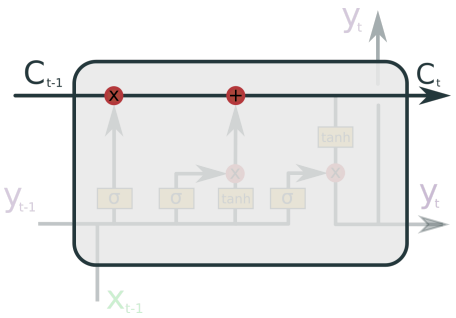

Figure 3.2: Weitergabe des Zellstatus innerhalb eines Knotens (inspiriert von [14]).

Einfluss auf den Zellstatus haben zwei verschiedene *Gates* (zu dt. Gatter/Tore). 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 3.3 zu sehen. Informationen aus dem Speicher, die keinen Einfluss mehr haben sollen, können so 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, sollte das bisher gespeicherte Genus keinen Einfluss mehr haben.

Das *Input Gate* soll im nächsten Schritt angeben, welche neuen Informationen in den Zellstatus C_t aufgenommen werden. Dies erfolgt

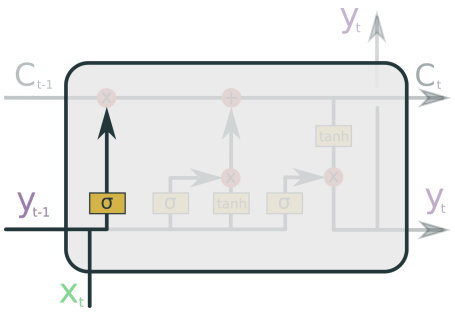

Figure 3.3: Einfluss des Forget Gates auf den Zellstatus (inspiriert von [14]).

in zwei Schritten, zunächst wird mit i_t ermittelt, welche Information geupdated werden soll. Im Vektor $\tilde{C}$ ist der eigentliche Werte (wie z.B. das Genus) enthalten, welcher den zuvor vergessenen Wert ersetzen soll (vgl. Abbildung 3.4) .

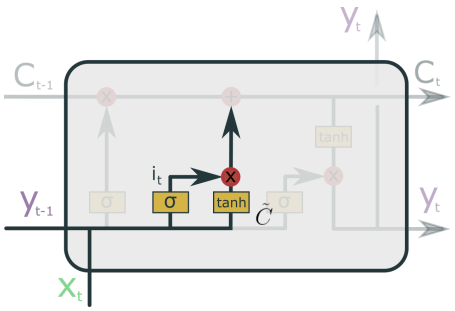

Figure 3.4: Einfluss des Input Gates auf den Zellstatus (inspiriert von [14]).

Wie der Zellstatus C_t nun die Ausgabe beeinflusst, wird über das *Output Gate* geregelt (siehe Abbildung 3.5). Dies soll in unserem Sprachbeispiel entscheiden, ob die Information des Genus für die Vorhersage des nächsten Wortes eine Rolle spielt. [3] [14]

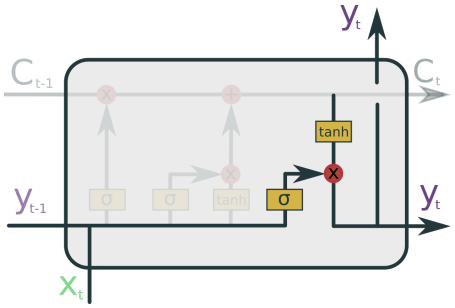

Figure 3.5: Das Output Gate regelt den Einfluss des Zellstatus auf die Ausgabe des Neurons (inspiriert von [14]).

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.

3.3.2 *Word2Vec*

4 REALISIERUNG

4.1 VERWENDETE TOOLS

Tensorflow Keras Rechencluster clara sysdig

4.2 VORVERARBEITUNG

Gegeben 10 szenarios 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. Ungenauigkeiten auf die in der Auswertung der Daten noch einmal genauer eingegangen wird filtern von switch statements weil keine system calls nur öffnende syscalls keine schließenden

Neuronale netze benötigen numerische werte deswegen umwandlung sys to int unbrauchbar für netz auf Grund aktivierungsfunktion –> 2>1 3>1 mittelwert 2

- 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 angegebnes n ngramme

syscalls welche nicht in trainingsdaten bekommen eine 0 ngramme thread aware bilden

Threadid kodieren:

- use entity embedding for ThreadID [4]
- relationship between threads and reduce size (possible 1000 different threads)
- choose size of embedding -thumbrule sqrt(unique value)

zeit kodieren

- use time delta of two different syscalls as new input

parameterlänge kodieren

4.2.1 *Parameterwahl*

N gram 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

W ord embedding parameterwahl wichtig sqrt(distinct)

4.3 ALGORITHMUS

aktivierungsfunktion üblich für Zeitreihendaten bei lstm cateforical cross entropy? ausgabe ebene wahrscheinlichkeit für jeden möglichen system call

4.4 ANOMALIEERKENNUNG

vorhersage des nächsten system calls nach sehen von ngram ausgabe wahrscheinlichkeit für jeden syscall, vgl nächsten tatsächlich gesehenen syscall mit vorhersage liefert prediction wahrscheinlichkeit anomalie score = 1-prediction wahrscheinlichkeit überschreitet dieser wert threshold –> anomalie

4.4.1 *Threshold*

betrachte validation daten und mache predicion höchster wert in validation set ist threshold falls darunter haben wir in val bereits mind ein fehlalarm, die es zu verhindern gilt trainingsdaten für berechnung von threshold kann bei overfitting zu unpassenden werten führen

4.5 STRUKTURIERUNG DER EXPERIMENTE

Hypothese: Threadinfos bringen was

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

4.6 METRIKEN

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

6

FOLGERUNGEN

BIBLIOGRAPHY

[1] Konstantinos Benidis et al. *Neural forecasting: Introduction and literature overview*. 2020. arXiv: 2004.10240 [cs.LG].

[2] 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.

[3] 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.

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

[5] 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.

[6] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen." In: (Apr. 1991).

[7] 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.

[8] 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.

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

[10] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. "MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks." In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*. Ed. by Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis. Cham: Springer International Publishing, 2019, pp. 703–716. ISBN: 978-3-030-30490-4.

[11] 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.

[12] 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.

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

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

[15] 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.

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

[17] 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.