Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Peter B Skramstad Jørstad

A comparison of sequence models for anomaly detection on process chains

Comparing sequence models for anomaly detection on process chains as a HIDS

Master's thesis in Information Security
Supervisor: Gudmund Grov
Co-supervisor: Lasse Øverlier
May 2023

NTNU
Norwegian University of Science and Technology

Peter B Skramstad Jørstad

A comparison of sequence models for anomaly detection on process chains

Comparing sequence models for anomaly detection on process chains as a HIDS

Master's thesis in Information Security
Supervisor: Gudmund Grov
Co-supervisor: Lasse Øverlier
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

A comparison of sequence models for anomaly detection on process chains

Peter Skramstad Jørstad

Chapter 1

Abstract

Abstract:

This study investigates the application of machine learning techniques in host-based Intrusion Detection Systems (HIDS), with a focus on the analysis of process relationships, an under-explored area in HIDS research. Through a rigorous literature review, we identified the current state of the art and potential research gaps in the field, facilitating the selection of suitable models and datasets for our investigation.

Our exploration centered on three prominent sequence models - Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Transformer models. The performance of these models was evaluated using the OpTC dataset, which represents a modern environment well-suited for process-tree based HIDS. Findings indicated that, among the three models, the Transformer model demonstrated marginally superior performance, although room for improvement was noted.

Further, our models were compared with state-of-the-art anomaly detection techniques. Despite outperforming several algorithms detailed in ProcAID, our Transformer model was inferior to ANUBIS, an anomaly detection approach that employs Bayesian Neural Networks. These results not only underscore the potential of Transformer models for anomaly detection tasks in HIDS, but also highlight the effectiveness of more sophisticated approaches such as Bayesian Neural Networks.

In conclusion, our study contributes to the ongoing discourse in HIDS research, revealing promising areas for future work and development to bolster anomaly detection performance.

Chapter 2

Sammendrag

Sammendrag

Denne studien undersøker anvendelsen av maskinlæringsteknikker i vertsbaserte Intrusion Detection Systems (HIDS), med fokus på analysen av prosessforhold, et underutforsket område i HIDS-forskning. Gjennom en grundig litteraturstudie identifiserte vi den nåværende toppmoderne teknikken og potensielle forskningshull i feltet, noe som muliggjorde valg av passende modeller og datasett for vår undersøkelse.

Vår utforskning sentrerte rundt tre fremtredende sekvensmodeller - Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) og Transformer-modeller. Ytelsen til disse modellene ble evaluert ved hjelp av OpTC-datasettet, som representerer et moderne miljø godt egnet for prosess-tre baserte HIDS. Funnene indikerte at blant de tre modellene, demonstrerte Transformer-modellen marginalt overlegen ytelse, selv om rom for forbedring ble bemerket.

Videre ble modellene våre sammenlignet med toppmoderne anomalideteksjonsteknikker. Til tross for at de utførte flere algoritmer detaljert i ProcAID, var vår Transformer-modell underlegen i forhold til ANUBIS, en anomalideteksjonstilnærming som bruker Bayesian Neural Networks. Disse resultatene understreker ikke bare potensialet til Transformer-modeller for anomalideteksjonsoppgaver i HIDS, men også effektiviteten til mer sofistikerte tilnærminger som Bayesian Neural Networks.

Til slutt bidrar studien vår til den pågående diskursen i HIDS-forskning, og avslører lovende områder for fremtidig arbeid og utvikling for å styrke anomalideteksjonsytelsen.

Contents

Figures

Tables

Code Listings

Chapter 3

Introduction

3.1 Topic covered by the project

An intrusion detection systems (IDS) is a system designed to detect and alarm on malicious activity on the system it is deployed in. IDS are divided based on what data they analyse and how they detect malicious behaviour. Network IDS (NIDS) analyses network traffic data, and Host based IDS (HIDS) analyses host data. Detection can be done either by looking for signatures (misuse detection) or anomalies (anomaly detection). Machine learning (ML) is a field within artificial intelligence, based on the concept that machines can learn from data, identify patterns and make decisions with minimal human intervention [1]. Traditionally, an IDS was based on manually constructed signatures for detecting malicious activity. Over the recent years, research on ML techniques for anomaly based detection has gained popularity. This project focuses on the area of applying machine learning algorithms to IDS utilizing host data (HIDS).

3.2 Keywords

cybersecurity, intrusion detection, anomaly detection, machine learning

3.3 Problem description

As the world has become more and more digitized, new attack surfaces and techniques has emerged. As misuse detection IDS depend on the signatures written for them, they are incapable of detecting unseen attacks. Anomaly based IDS are design around a baseline of normal activity. However, as real-world environments are dynamic and any abnormal behaviour will be triggered on, they are prone to a high false alarm rate [2][3][4]. As alarms are mostly reviewed manually it creates a lot of labor for the operators reviewing them. Further, since the amount of data a normal company conjures on a daily basis has grown large, even a low false alarm rate can create a huge number of false alarms.

Network IDS are mostly deployed outside the intranet of an organization which makes internal attack hard to detect [5]. Also, as network traffic is becoming increasingly encrypted it is potentially problematic for NIDS [6][5]. These issues points towards host based IDS. Most HIDS research are based on system calls, which are low-level artifacts of the system kernel. However, there is a lack of publicly available datasets for HIDS containing system call, and traditional datasets cannot represents a modern environment. There are several studies detailing the problems relating to currently available HIDS datasets [5]. For both HIDS and NIDS datasets, these problems include old data, redundant information, unbalanced numbers of categories, and insufficient data volume [4]. Further, a tracing tool is required to run on the host in order to collect system calls, which adds a computational cost, thus making system calls non-present in common log types. As system-call based HIDS has its challenges with datasets, process tree based HIDS should be considered.

Processes on a computer has parent-child relationships, and will form trees which has paths or chains. However, there is little research on HIDS based on process relationships. Indeed, a survey on HIDS from 2019 only had a small section for process-based HIDS, and only one research group had analyzed parent-child relationships [6]. This project will explore the potential of adopting NLP-techniques used in system call based methods on process tree based HIDS.

3.4 Justification, motivation and benefits

Several companies report on and predict an increasingly high cost of cyber crime. In the US, the Federal Bureau of Investigation's annual Internet Crime Complaint Center (IC3) reported a cost of \$6.9 billion in 2021 [7]. In 2020, Cybersecurity Ventures projected the total global cost of cyber crime to reach \$10.5 trillion USD annually by 2025 [8]. Accentures study on 11 countries found that the average cyber crime cost for a company in 2018 was \$13 million [9]. Worse still than common criminals, advanced persistent threats (APT) have the ability to launch sophisticated attacks to intrude an organization, and has been considered a serious global problem since the 2010s [10]. Therefore, the ability to detect and alert on malicious activity is of utmost importance.

By analyzing data from the operating system of a host, HIDS has the privilege of having access to context-rich data for processes and activities, which support the ability to detect internal attacks [6][5]. All which is needed to construct a process tree is the parent-child relationship between the processes, which is included in common log types (e.g. Sysmon and Microsoft Security Event Log). As datasets with just common logging enabled is easier to construct than ones with system call tracing, there are modern, sufficiently sized datasets available that are compatible for process tree based HIDS. Also, most enterprises are likely to have these log types enabled, which makes implementing this type of HIDS cost-efficient. Considering that system call traces has similarities with process chains in regards to both being sequential, machine learning techniques used on the one might be adaptable on the other. As a process tree can be seen as both a graph or as individual chains, different ML techniques may be applied. This in combination with dataset availability might give grounds for anomaly detection with an acceptable false alarm rate.

3.5 Research questions

The main research questions to be answered by this project are as follows:

1. What is the state of the art on sequential machine learning methods used on HIDS?
2. Are there any gaps in the current research?
3. Are these methods suitable to be used on process-chains?
4. How do these methods compare against each other and SOTA?

Which leads to the sub-questions:

a. Which datasets are available and most suitable for HIDS?
b. What is the SOTA on process trees?
c. How does mask rate affect performance?

3.6 Planned contributions

This project will examine the possibility of adopting machine learning techniques used for system call based HIDS to process tree based HIDS. More specifically, natural language processing (NLP) techniques, as these have been extensively used both on system calls sequences and network traffic. The project also seeks to review different machine learning approaches already used on process tree based HIDS, and what datasets are available.

For this, a proof of concept model will be built first, followed by one or two more advanced. Hopefully, we are able to do anomaly detection with an acceptable FAR. However, the overall goal is to complete a proof of concept and highlight the potential in process trees for HIDS.

Chapter 4

Background

4.1 Intrusion detection systems (IDS)

[[Intrusion detection, a critical component of computer security, is defined as the identification of any unauthorized activities potentially detrimental to an information system [11]. Intrusions can compromise the confidentiality, integrity, or availability of information and services to legitimate users, making systems unresponsive. These malicious activities or intrusions include break-ins, penetrations, and other forms of computer abuse [12].

Intrusion detection systems (IDS) are specifically designed to identify these intrusions. IDS can be either a software or hardware system and their primary purpose is to detect malicious actions that could not be identified by traditional firewalls [11]. These systems have three core components: data collection, conversion to select features, and a decision engine. They ingest various data types such as system logs or network flows, convert these into feature vectors or predefined units of data, and then use an algorithm to decide whether the given data represents an attack or not. They can be configured to alert a user or trigger an automated response system [6].

Intrusion detection systems can be broadly categorized into Signature-based Intrusion Detection Systems (SIDS) and Anomaly-based Intrusion Detection Systems (AIDS) [11]. Additionally, there are host-based IDS (HIDS), network-based IDS (NIDS). HIDS, in particular, have been gaining significant attention in the cybersecurity community due to their fine granularity and ability to detect internal attacks [5].

Unlike antivirus systems, which monitor all activities inside a system, HIDS collect and analyze specific system data to detect anomalies, including system call patterns, system events, and the status of the file system [12]. Anomaly detection techniques, which identify items, patterns, and events deviating from the normal behavior of system processes, are particularly useful in HIDS. These anomalies or outliers are commonly found in fraudulent activities, system health issues, fault detection, and cyber intrusions [12]. Furthermore, HIDS have the capability to monitor individual systems and provide robust tools for centralized audit policy management, statistical analysis, and evidentiary support [12].

Intrusion detection systems (IDS) are typically categorized into Signature-based Intrusion Detection Systems (SIDS) and Anomaly-based Intrusion Detection Systems (AIDS), with each having their strengths and weaknesses. [5, 6, 11, 12].

SIDS, also known as misuse detection, primarily rely on pattern matching techniques to identify known attacks by referencing a database of previously recorded attack signatures [5, 11, 12]. These signatures might include specific patterns, known malicious instruction sequences, or system vulnerabilities [12]. When a match between a recorded activity and an existing signature happens, an alert is triggered. Popular tools such as Snort and NetSTAT employ SIDS, and despite the simplicity and high detection accuracy for known intrusions, these systems has their challenges [5, 11].

One significant limitation of SIDS is its inability to detect novel or zero-day attacks, since these exploits do not have existing signatures in the database [5, 6, 11]. Advanced and polymorphic malware, coupled with the rising number of targeted attacks, also complicate SIDS performance, necessitating continuous updates and maintenance of the signature database [11, 12].

On the other hand, AIDS works by defining a baseline of normal behavior of a computer system using machine learning, statistical, or knowledge-based methods. It flags any significant deviation from this model as an anomaly, which might be malicious activity [5, 11, 12]. Notably, AIDS has the advantage of identifying zero-day attacks, as it is not reliant on a signature database [6, 11, 12]. It is also capable of detecting internal malicious activities thereby making harder for cybercriminals to bypass without generating an alert [11].

However, AIDS is not without its challenges. Firstly, it tends to produce a higher rate of false positives due to the inherent difficulty in distinguishing between normal activities and actual intrusions [6, 11, 12]. Secondly, making an accurate definition of normal behavior for the system can be complex and resource-intensive, requiring detailed knowledge of the system's operation, and this may lead to overlooking malicious behavior if it is considered within the normal baseline [5, 12]. Furthermore, attacks can often be obfuscated within the ambient data noise, especially if the training data has high variance, or if the attack patterns are present within the training data, resulting in a skewed baseline of normal behavior [6].

In conclusion, while SIDS and AIDS have their respective advantages and disadvantages, the selection of an appropriate IDS generally depends on the specific security needs of a system or network. Due to the nature of a dynamic threat landscape, and the ability of anomaly detection systems to detect novel attacks, this study revolves around anomaly detection on HIDS.

4.2 Past decade's evolution of Sequential Models

The past decade, sequential models has seen a lot of improvements. This period is marked by development and new applications of machine learning techniques in the field of Natural Language Processing (NLP). This chapter provides a brief overview of this this period, focusing on the major milestones.

4.2.1 Word Embeddings (2013-2014)

Word embeddings are dense vector representations of words that capture semantic meanings based on context. In 2013, Mikolov et al. [13] introduced Word2Vec, a model that used shallow neural networks to generate these vector representations. This model had a huge improvement in the efficiency of the training procedure compared to earlier models, which lead to better word representations. [14]

This year also saw an adoption of neural network models in NLP. One model in particular, recurrent neural networks [15] (RNNs; Elman, 1990), gained popularity for its ability in dealing with dynamic input sequences, which is in the nature of NLP. [16]

4.2.2 Sequence to Sequence Learning (2014-2015)

The next milestone was the introduction of sequence to sequence (seq2seq) learning by Sutskever, Vinyals, and Le in 2014 [17]. This allowed a neural network to map an input sequence, such as a sentence in one language, to an output sequence, such as a translation of the sentence in another language. [16]

In 20012, the attention mechanisms was introduced by Bahdanau et al. [18], to help improve upon seq2seq models. Attention allowed a model to focus on certain parts of the input sequence when generating an output, proving very successful and improving performance on sequence to sequence tasks. [14]

4.2.3 LSTM and GRU Networks (2015-2017)

Long Short-Term Memory (LSTM) networks were popularized in response to what's called vanishing gradient problem of conventional RNNs, which makes it difficult to learn long-range dependencies. LSTM networks introduced a form of memory into the network, enhancing their ability to retain information over long sequences. They were developed by Hochreiter and Schmidhuber back in 1997 [19], but only gaining foot-hold later. [16]

A similar solution came with Gated Recurrent Units (GRUs). Introduced by Cho et al. in 2014 [20], GRUs offered a simplified yet effective variant of LSTM-style gating mechanisms for handling sequential data. [21]

4.2.4 Transformer Models (2017-)

The Transformer model, proposed by Vaswani et al. in 2017 [22], was a breakthrough that marked a shift away from recurrence-based architectures like RNNs. They introduced a new type of attention, called self-attention mechanisms, which allowed the model to learn the relationships between all words in an input sequence. This lead to a significant increase in accuracy of NLP tasks such as machine translation. [23]

The success of Transformers lead to pre-trained models, which are model that are trained on a large text corpus, then fine-tuned for a specific task. Examples of such models are OpenAI's GPT (Generative Pretrained Transformer) and Google's BERT (Bidirectional Encoder Representations from Transformers). These models have shown to improve state-of-the-art methods over a wide range of tasks. [14]

4.3 Encoding Techniques in Natural Language Processing

Encoding techniques in machine learning are used to convert raw data into a format that can be interpreted by machine learning algorithms. There are several way to convert the raw data into numbers, some common encoding techniques used in machine learning are explained here.

4.3.1 One-Hot Encoding

One-hot encoding is a technique where each word in the vocabulary is represented as a binary vector with the same dimension as the size of the vocabulary. The vector consists of zeros, except at the index that corresponds to the specific word, which is marked as one. While straightforward and computationally efficient, one-hot encoding falls short in capturing any semantic relationships between words, and its high dimensionality can pose challenges with large vocabularies [24, 25].

4.3.2 Label (or Index) Encoding

This is a simple and common approach where each unique category value is assigned an integer value. For example, red could be 1, blue 2, and green 3. This method is straightforward but can lead to the algorithm mistakenly interpreting the indexes as a scalar, thereby assuming a numerical difference between the categories that does not exist [24].

4.3.3 Count Vectorization (Bag of Words)

Count Vectorization, also known as Bag of Words (BoW), represents a vector with a count for every token present in the document or text. It disregards the order of words, thus treating the document as a 'bag' of words. Although it's computationally efficient and easy to understand, BoW completely disregards the semantic relationships between words and the context of words in the document. [25, 26]

4.3.4 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is an enhancement over simple count-based methods, quantifying the importance of a word in a specific document relative to its frequency in the entire corpus. This approach assigns higher weigh to words that occur frequently in a one document but are less frequent in the entire, and vice versa. The idea is to weigh important words or tokens more. Despite its advantages, TF-IDF still does not capture semantic relationships between words [24–26].

4.3.5 Word2Vec

Word2Vec is a popular word embeddings technique that represents words as vectors in a continuous vector space. Unlike the above methods, Word2Vec captures the semantic relationships between words by placing semantically similar words close to each other in the vector space. In order to to this, Word2Vec employs a shallow neural network, using either of two methods:

Continuous Bag of Words (CBOW): This model predicts a target word given its context, as in surrounding words. The context is represented as the bag of words for simplicity, hence disregarding the order of words.

Skip-gram: This model does the opposite, it predicts the context words from a target word. For each context word, it generates skip-gram pairs between it and the target word.

The neural network is trained on these tasks usually using a large corpus of text. Once trained, it is the actual weights of the hidden layer of the neural network that serve as the word vectors, constructing what's called an embedding matrix.

The main purpose of Word2Vec is to capture the semantic meaning of words in a compact representation. However, it is not without its drawbacks. [24, 25, 27]

Transformer Encoders

Transformers, introduced in the paper "Attention is All You Need" by Vaswani *et al.* [22], primarily consist of an encoder and a decoder. However, we'll focus on the encoder part, which is applied by popular models such as for BERT[28].

A Transformer encoder processes the input data (such as a sentence in a natural language processing task) all at once rather than sequentially, which can help with efficiency and long-range dependencies [29]. Each encoder consists of a stack of identical layers. Each layer has two main components: *Self-attention layer (or multi-head attention layer)*: This layer computes an attention score for each word in the input with respect to every other word. This allows the model to consider other words in the sentence as it encodes a particular word. *Feed-forward neural network*: After the attention scores are used to compute a weighted representation of the input, this is passed through a feed-forward neural network for further processing. These components are connected with normalization and residual connections, which help with in training deep networks.

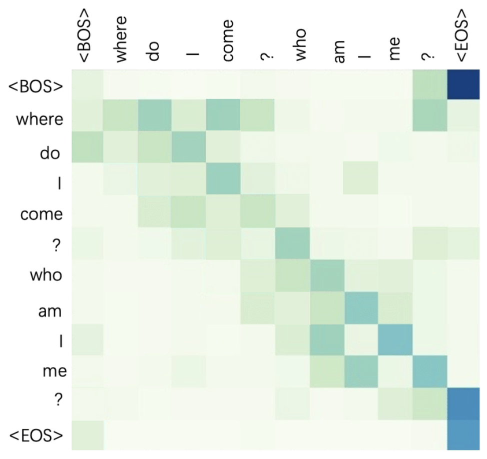


Figure 4.1: Self Attention Weights [30]

4.4 Process Trees

4.4.1 Process Create Event

Windows Event ID 4688 is a security event that tracks when a new process is created, and is generated by the operating system's auditing mechanism. The event provides information about the newly created process, the user account responsible for creating it, and other details.

Event ID 4688 includes the following information: *Security ID (SID)*: A unique identifier of the user account that initiated the process creation. *Account Name*: The user account name associated with the Security ID. *Account Domain*: The domain the user account is a part of. *Logon ID*: A semi-unique (unique between reboots) identifier for the logon session. *New Process ID (PID)*: A semi-unique (unique between reboots) identifier for the newly created process. *New Process Name*: The full path to the executable file associated with the new process. *Token Elevation Type*: Indicates the elevation level of the user's token (e.g., full administrative rights or limited user rights). *Mandatory Label*: The integrity level of the process, which is determined from the user integrity level and the file integrity level of the executable. *Creator Process ID*: The identifier of the parent process that initiated the creation of the new process. *Creator Process Name*: The full path to the executable file associated with the creator process.

[31] [32]

Sysmon (System Monitor) is a Windows system service that monitors and logs system activity to the Windows event log. It provides more detailed information about process creation events compared to the native Windows Event ID 4688. Following are some additional fields that are included in a Sysmon Event ID 1: *ProcessGuid*: A globally unique identifier (GUID) for the process making event correlation easier. *ParentProcessGuid*: The GUID of the parent process that created the new process. *CommandLine*: The full command line used to initiate the

```
A new process has been created.

Creator Subject:
        Security ID:            SYSTEM
        Account Name:           -
        Account Domain:         -
        Logon ID:               0x3E7

Target Subject:
        Security ID:            NULL SID
        Account Name:           -
        Account Domain:         -
        Logon ID:               0x0

Process Information:
        New Process ID:         0x1e4
        New Process Name:       C:\Windows\System32\services.exe
        Token Elevation Type:   TokenElevationTypeDefault (1)
        Mandatory Label:                Mandatory Label\System Mandatory Level
        Creator Process ID:     0x394
        Creator Process Name:   C:\Windows\System32\wininit.exe
        Process Command Line:
```

Code listing 4.1: EventID 4688

process, including arguments and parameters. *LogonGuid*: GUID of logon session associated with the event. *Hashes*: A set of cryptographic hashes (e.g., MD5, SHA1, SHA256, IMPHASH) for the process's executable file. [33] [34]

4.4.2 Process Tree

In operating systems, processes can spawn other processes, which in turn can spawn even more processes, forming a hierarchical structure. A process tree visualizes these relationships, with the root node representing the initial process and child nodes representing the descendant processes. A process chain represents a specific path in this tree, showing how a series of processes are related through parent-child relationships. A process chain, in this context, refers to a sequence of related processes in a process tree, where each process in the chain is an ancestor of the next process. In other words, it is a series of parent-child relationships between processes that originate from a single starting process, extending down to the leaf processes in the process tree.

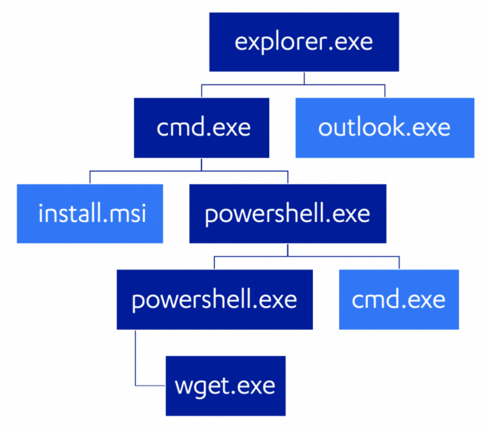


Figure 4.2: An example of a process tree, with one process chain in darker blue [35]

Chapter 5

Methodology

5.1 Introduction

The purpose of this methodology chapter is to outline the systematic approach employed to address the research objectives of this study. By providing a detailed description of the methodology we aim to ensure transparency and reliability in our research process. This chapter serves as a roadmap for readers, enabling them to understand the steps taken from start to end.

The methodology chapter is organized into five key sections. Firstly, the research design section explains the overall approach of the study, highlighting the mixed-method nature that combines both quantitative and qualitative methodologies. Following this, the literature review section describes the examination of existing research, aiming to identify relevant ML techniques and potential research gaps. The subsequent section focuses on model selection, where the most promising ML techniques for evaluation within the context of the study are chosen. Following is the dataset selection section which outlines the criteria and process used to identify the most suitable dataset for the study. The evaluation section explains the steps involved in assessing the selected models' performance and drawing meaningful conclusions. Lastly we reflect on ethical considerations.

5.2 Research Design

The present study adopts a mixed-method approach, combining both quantitative and qualitative research techniques. Focused on Intrusion Detection Systems (IDS) and machine learning methods, the design is primarily quantitative given the numeral metrics that can be applied to the analysis of machine learning methods. The IDS models are evaluated based on metrics such as loss and accuracy using labeled data, allowing us to make comparative judgments. Alongside this quantitative aspect, we also incorporate a literature review component, which employs a more qualitative methodology. Here, studies are evaluated using a combination of numerical and non-numerical metrics to assess their relevance and effectiveness.

Our primary objective is to explore the applicability of machine learning (ML) techniques to process sequences in a Host-based Intrusion Detection System (HIDS). To achieve this, we have crafted a research design encompassing the following stages:

1. **Literature Review**: This initial stage involves a comprehensive examination of existing literature to discern ML techniques that have been applied to HIDS. Here, we also aim to identify any gaps in the current body of research, thereby highlighting areas of potential exploration.
2. **Model Selection**: Following the literature review, we will select the most promising ML techniques for evaluation within a HIDS context. These selections could either be established methods that have proven their efficacy or promising areas of exploration unveiled by the identified gaps in the literature.
3. **Dataset Selection**: From the literature review, we will identify and select the most suited dataset based on some critera outlined below.

The quantitative component of our study follows a standard approach comprising the steps below:

1. **Data Collection**: The initial stage involves gathering relevant data to serve as the basis for our models.
2. **Data Preprocessing**: The collected data is processed to make it suitable for use by the models. This involves tasks such as handling missing data, normalization, and encoding categorical variables.
3. **Feature Selection and Engineering**: We identify the most relevant features for our models and engineer encodings to optimize performance.
4. **Model Training**: Our chosen models are trained using preprocessed and feature-engineered data.
5. **Model Validation and Evaluation**: Our trained models are then tested on unseen data, using suitable metrics to evaluate their performance.
6. **Results Interpretation**: Finally, we draw meaningful conclusions from the evaluation results.

The research design outlined above aligns with our research question, as it systematically enables the exploration of the utilization of ML techniques in IDS, more specifically, in HIDS. It further allows for a comparison between different models and the current state-of-the-art, illuminating the potential advantages and shortcomings of each.

In summary, this research design facilitates a comprehensive understanding of the topic through a thorough literature review and quantitative analysis. Its structure supports the identification and exploration of gaps in the literature, enabling us to contribute to the academic conversation in a meaningful and insightful way.

5.3 Literature Review

To better comprehend the current landscape of machine learning methods employed in HIDS, a literature review was done. The goal for the review was to identify, evaluate and synthesize the most relevant academic papers and other sources. The methodology comprised of several phases:

Identification of Sources. We started by searching the electronic databases Google Scholar, Oria (NTNU Universitetsbiblioteket, Norske fagbibliotek), IEEE Xplore and ACM Digital Library, using keywords relevant to our topic. On key papers, we also did a manual two-way search on both references from the paper and articles that cite this paper. This was done to uncover additional sources that might not have appeared in our initial search.

Inclusion and Exclusion Criteria. In order to filter out non-relevant sources, we defined criteria to decide which sources to include or exclude. The first criteria was relevance to our research questions, which was assessed by first pass screening of titles and abstracts. The age of the publication is important for finding contemporary articles, and the cutoff was set to 2010. The credibility of the paper is indicated by the publisher and where it is published, and whether it is peer-reviewed. A reputable academic journal or conference proceedings, published by a reputable association gives higher credibility.

Evaluation. We evaluated the source's methodological quality to ensure that we relied on sound research. We considered the research design, data collection and analysis methods, and the overall coherence of the paper, especially the discussions. This is assessed by second pass screening of the full text.

Synthesis. Lastly, we synthesized our findings to identify common themes, contrasts, and gaps in the existing literature. This is done to build an understanding of the state of the art and how our research can contribute.

5.4 Model Selection

Based on the literature findings, we narrowed down our selection to a few promising ML techniques for further evaluation. These techniques must have either shown promise in HIDS applications, or represent promising new directions.

The selected models will be subject to evaluation in order to determine their applicability and effectiveness in the context of HIDS. We will conduct a comparative analysis between the selected models, as well as against the state-of-the-art models in the field.

Our goal in this model selection phase is not only to identify the best performing ML techniques for HIDS but also to contribute to filling the gaps in the literature. This dual objective ensures our research provides valuable insights into the current state of IDS and ML intersection and offers novel perspectives for future exploration.

5.5 Dataset Selection

This section describes the process to select an appropriate dataset for our research. We examined several potential datasets identified in our literature review, assessing each one based on some criteria. These criteria include size, features, data quality, and the presence of labeled data.

Size. The performance of machine learning models can be heavily influenced by the size of the dataset they're trained on. As a rule of thumb, if there's not enough training data, the model's predictions won't be accurate. If the model has too many constraints, it won't fit the limited training data well, called underfitting. Conversely, a model with too few constraints may overfit the training data, causing it to perform poorly as well. Furthermore, if there's insufficient test data, the model's performance estimation could be overly positive and fluctuate significantly. [36] We therefore prioritized datasets of considerable size.

Features. We examined the features present in each dataset. The dataset must at a bare minimum include features that makes it possible to construct process chains, which are the fields ProcessID (PID) and Parent-PID (PPID). Other features that are relevant to our research and will enrichen the model training is also weighted.

Quality and Reliability. We looked for datasets with minimal missing or erroneous data, which could potentially skew our results or make extensive data cleaning necessary. Further, we looked for a data collection process that was systematic and unbiased, ensuring the reliability of the dataset. Real-world data would be preferable, as it would train the model to a realistic scenario. However, a synthetic dataset can do fine.

Labels. The dataset must include both benign and malicious data, as this is necessary for training and evaluation of the models. Even though models can be trained exclusively on normal data, malicious data is needed for doing proper evaluation of the model, ensuring it can discriminate between that and benign data. How well-made the labels are can vary greatly between dataset, and since bad or naive labelling of the data can lead to inaccurately training or evaluation, we ensure to weight how good the labels are.

Using these criteria, we will select a dataset that best met these conditions, providing a suitable foundation for our project. The specifics of the chosen dataset, including its source, size, features, and labels, will be described in more detail in later.

5.6 Evaluation

5.6.1 Metrics

Multiple quantitative metrics are used to evaluate an IDS and a machine learning model. In order to best describe and evaluate models against each other, several metrics are used [3]. A confusion matrix is a table with prediction results on the columns and ground truth on the rows, enumerating the possible classifications. For binary classification, this will be a 2*2 matrix [4], as shown in the table below .

Table 5.1: Confusion Matrix

	Classified Positive	**Classified Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

The four categories are: True Positive (TP): a sample correctly classified as positive, True Negative (TN): a sample correctly classified as negative, False Positive (FP): a sample wrongly classified as positive, False Negative (FN): a sample wrongly classified as negative. These categories are used to calculate the metrics.

Accuracy: Accuracy is the fraction of correct predictions our model made over the total number of predictions. It is most useful when target classes are well balanced.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: Precision, also known as positive predictive value, is the fraction of correctly marked positives among all the predicted positives, representing confidence in a TP.

$$\text{Precision} = \frac{TP}{TP + FP}$$

True Positive Rate (TPR), Recall or Sensitivity: TPR, also known as Recall or Sensitivity, is the fraction of correctly marked positives among all the actual positives. This is an important metric in an IDS as it reflects the ability to detect attacks.

$$\text{Recall} = \frac{TP}{TP + FN}$$

False Negative Rate (FNR): FNR is the ratio of the number of false negatives (predicted negatives that are actually positive) to the total number of actual positives.

$$\text{FNR} = \frac{FN}{TP + FN}$$

False Positive Rate (FPR): FPR is the ratio of the number of false positives (predicted positives that are actually negative) to the total number of actual negatives.

$$\text{FPR} = \frac{FP}{FP + TN}$$

True Negative Rate (TNR): TNR, also known as Specificity, is the ratio of the number of true negatives (predicted negatives that are actually negative) to the total number of actual negatives.

$$\text{TNR/Specificity} = \frac{TN}{FP + TN}$$

F1-score: The F1-score is the harmonic mean of precision and recall, giving both metrics equal weight. It ranges from 0 to 1, with 1 being perfect precision and recall, and it is a good metric to use when the positive class is the minority or when false positives and false negatives are equally important.

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Receiver Operating Characteristic (ROC): The ROC curve is a graphical representation of the contrast between true positive rates (TPR) and false positive rates (FPR) at various thresholds. It can be used to evaluate the trade-off between sensitivity (or TPR) and specificity (1 – FPR).

AUC: Area Under the ROC Curve: AUC represents the probability that a random positive example will be ranked more highly than a random negative example. AUC ranges from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0, one whose predictions are 100% correct has an AUC of 1.0.

Precision-Recall (PR) curve: The precision-recall curve is a graphical representation of the trade-off between precision and recall for different classification thresholds in a binary classification problem. PR curves are particularly useful when dealing with imbalanced datasets, where the positive class is much less frequent than the negative class.

Area Under the Precision-Recall Curve (AUC-PR) or Average Precision (AP): is a metric that quantifies the overall performance of a binary classification model based on the precision-recall trade-off. It represents the average precision across all possible recall levels.. A model with perfect precision and recall would achieve an AUC-PR value of 1. In the context of imbalanced datasets, AUC-PR can often provide more informative performance summaries than traditional metrics like accuracy or AUC-ROC.

Matthews Correlation Coefficient (MCC): is a metric measure the quality of binary classifications. It takes into account all four aspects of the confusion matrix and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction, and -1 indicates total disagreement.

5.6.2 Process

The first step in the evaluation process involves determining a threshold value based on the model's loss scores on the validation dataset. The validation dataset, which is a subset of the data that the model has not been trained on, provides a reasonable estimate of how the model might perform on unseen data. The threshold is set as a certain percentile of these validation losses. This percentile-based thresholding is an effective way to balance performance metrics as it is granular, and based on percentage rather than some fixed values.

The actual evaluation is based on the test datasets. We use two distinct test datasets - one comprising benign sequences and the other, malicious sequences. It's crucial to note that the model has never encountered these datasets before, ensuring an unbiased evaluation on new data.

The model's loss scores from the benign and malicious test datasets are then used to evaluate its performance. This evaluation is carried out by employing the set of metrics that we've previously presented in . This collection of metrics offers a comprehensive view of the model's performance.

5.7 Ethical Considerations

It is crucial to address the ethical considerations associated with the research. One important aspect to consider is data privacy and protection. As researchers, we must ensure that any personal or sensitive information is handled with care and in compliance with relevant data protection regulations.

To mitigate potential privacy concerns, the use of synthetic datasets can be employed. Synthetic datasets are artificially generated and do not include real data from real individuals. By using synthetic data, we eliminate the risk of exposing personal information or violating privacy rights. However, it is important to note that even when using synthetic datasets, it is necessary to adhere to ethical guidelines and ensure that the generated data does not inadvertently reveal private or sensitive information.

Chapter 6

Literature Review

6.1 Datasets

Available datasets with process parent-child relationships. From various sources, including Windows Security Event logs and Sysmon logs. There are many datasets that can be considered outdated as of today. The following dataset are the ones consider according to our criteria.

6.1.1 Splunk Attack Data

Splunk Attack Range is a detection development platform developed by Splunk. A user can build a small lab infrastructure, and use the range to perform attack simulations with engines such as Atomic Red Team or Caldera to simulate real attack data. The deployment consists of several Windows systems, a Kali machine, some Splunk servers and more. Several log sources are collected from the machines, including Windows Event Logs, Sysmon Logs and Sysmon for Linux Logs [37]. A repository of logs from the range from different contributors has been created, called Attack Data [38]. The datasets created by simulating atomic test are labeled with MITRE ATT&CK technique used, they are freely available, and can be downloaded from the Github repo. The files totals to above 9Gb.

6.1.2 The Security Datasets project

The Security Datasets project is an open-source project, that collects benign and malicious dataset from different systems. The project is open for everyone to contribute to the repository. The datasets are produced by running atomic red team test in a virtual environment, and labeled according to the technique used. The repository includes host and network logs from Windows, Linux and AWS. As of now, there are 91 datasets for Windows, 3 for Linux and 2 for AWS [39].

6.1.3 Operationally Transparent Cyber (OpTC) Data Release

Operationally Transparent Cyber (OpTC) was a pilot study with the objective of determining if DARPA Transparent Computing program technologies could scale up to a thousand machines while maintaining detection performance. A third-party instance acted as a red team and test coordinator. The OpTC system architecture was based on one used in TC program evaluations, and was evaluated in a well instrumented facility. Virtual clients were programmed to complete basics tasks to mimic generic daily user activity. Every client ran an endpoint sensor that sends real time system-level data to a translator server. The translator compiles co-related events into aggregate messages, and undergo additional processing ending up in a format called eCAR. This evaluation underwent for two weeks, with the first period used for benign record generation, and the last three days for a red team operation. The clients ran Windows, and data was collected from 500 clients for the evaluation period. The data collection totals up to approximately a terabyte in a compressed JSON compatible format. A red team report contains the ground truth from the red team operation to assist in evaluation. The dataset is divided into ecar-bro containing flow-start event, ecar containg endpoint data, and bro containing data from a bro sensor. Data for the benign period and for the evaluation period are separated. The eCAR format builds on MITRE's CAR, and include objects such as PID, PPID, image path, and command line [40].

6.2 System Call Based Methods

System calls provides the interface for which a process can request services from the operating system. The services of the operating system reside in the kernel space, and are delivered through an API by the use of systems calls. This is the only entry point for the kernel space [41]. Modern operating systems range from having a couple of hundred different systems calls to a couple of thousand. For example, Linux has listed above 300 different in its manual page [42], while Windows has almost 2000 [43]. Since system calls are pure in the sense that they are not interpreted, filtered or processed in a way that can obfuscate the event, they are a popular choice for HIDS Creech and Hu [44]. Whenever a process runs, it will produce a sequence of system call called a trace. Often, the trace for a process is collected in a given time frame. Because of the sequential nature of system call traces, natural language processing techniques are suited for processing them. Taking it further, Creech and Hu [44] imagined system call as letters, a string of connected systems calls thereby a word, and a combination of these as phrases. The most popular methods are n-gram, sliding window algorithm, bag-of-words, term frequency-inverse document frequency (TF-IDF) and Markov models Bridges *et al.* [6]Liu *et al.* [5]. The definition of n-gram in this context is a continuous sequence of n system calls from a system call trace within a specified time frame. With the sliding window algorithm, a window of size n passes along a complete system call trace to produce a number of n-grams. It is also possible to use multiple-length windows to produce multiple length *n*-grams, as done by Creech and Hu [44] following Forrest *et al.* [45].

Liu *et al.* [5] surveyed the usage of various types of NN in HIDS research, including multilayer perceptron, self-organizing maps NN, radial basis function-based NN, extreme learning machine, and self-structuring confabulation network. The survey also states that deep learning has proven to be good at finding deep features within big data and therefore is a promising technique for HIDS. However, it points out the disadvantage that training a deep NN is time consuming and requires strong hardware.

Creech and Hu [44] used the sliding window algorithm to produce the training data from the KDD98 and ADFA-LD datasets, with both validation and attack data. They used this data to train an Extreme Learning Machine (ELM), a training scheme for a feedforward neural network with a single hidden layer Huang *et al.* [46]. The design was implemented on every host and required a large amount of computational power, taking up weeks to train. However, the classification part was fast, and the produced accuracy results were good.

Anandapriya and Lakshmanan [47] also conducted research on anomaly-based HIDS with ELM. Several works [3][48][49][4] have popularized LSTM and GRU for HIDS.

Tong *et al.* [50] suggested a hybrid model of RBF NN combined with an Elman NN for both misuse detection and anomaly detection. The RBF NN was used for real-time classification and the Elman NN was used to handle the memory for previous events. The DARPA Intrusion Detection Evaluation Data Sets of 1999 was used for training and testing of the network Cunningham *et al.* [51]. The tests resulted in a 93% detection rate and and 2.6% FAR for anomaly detection, and 95.3% detection rate and 1.4% FAR for misuse detection. However, the classification itself is done by the non-recurrent RBF NN, and therefore does not look for the sequential features we are interested in.

In Kim *et al.* [52],proposes an architecture with two parts, with the first being an LSTM NN doing language modeling of system calls and the second part doing anomaly prediction based on ensemble of thresholding classifiers derived from the first part. The goal of the ensemble method is to build a "strong normal" model, i.e. a sequence of system calls which is highly likely normal, and thereby reducing false positives. This is done by composing several classifiers into one. Evaluation with the KDD98 and ADFA-LD dataset yielded a 100% detection rate with a 2,3% FAR and AUC of 0.994, and a 100% detection rate with 50-60% FAR or AUC of 0.928, respectively.

Chawla et al. suggests a combined CNN/RNN architecture for anomaly based HIDS Chawla *et al.* [53]. The first part of the architecture consists of an embedding layer and several CNNs that works as a pre-processing step, making the system call sequences smaller resulting in faster training. As this is 1D data, the convolution action done by the CNN consists of filters sliding across sequences to extract a feature map for local sub-sequences. By adjusting the number of stacked CNNs one can control the effect of this part. The second part consists of a RNN with

GRU units and a Time Distributed Layer. The authors built five independent model based on this arcitecture, with different hyperparameters. The NN is trained on normal sequences from the ADFA-LD dataset to produce a probability distribution for the next call in a sequence, similar to a language model. The authors were able to achieve a detection rate of 100% with a FAR of 60%, or AUC of 0.81. Compared to [52] the results were not as good, but with faster training times.

6.3 Process Tree Based Methods

Processes on a host has parent-child relationships, and will form process trees. These trees has "paths" or "chains". Some works explore treating the tree as a graph and use methods like link prediction, gradient boosted trees, and autoencoder to detect malicious process creations. PROBE is a Host based intrusion prevention system (HIPS) proposed by in Kwon *et al.* [54]. It uses the parent-child relationships of processes to detect and block abnormal behaviour. The mechanisms of PROBE has three parts; *Tree Builder* for constructing the process trees, *Path checker* that inspect the process relationships, and *Process controller* that blocks abnormal processes. *Tree Builder* uses the identifiers PID (ProcessID) and PPID (ParentProcessID) of a process to make the process tree for running processes on a host. Each node in the tree is a process and it has edges to its parent and child processes, if any. The *Path checker* checks if something abnormal occurs based on information from operating system system objects—attributes, modification time, and more. Advantages of this method is that it lightweight and does not require much computing power. However, disadvantages of this method is that it relies on manually defined rules, and might not scale very well when the amount of data grow large.

In Read [55], Read proposes an anomaly based HIDS that treats process trees as graphs and uses unsupervised link prediction to find anomalies.

In Patel [35] the researcher presents an anomaly detection method focused on analyzing process creation chains. The hypothesis being that rare process creation chains might indicate malicious activity. To recognize similarities between various file paths, the researchers treated the paths as a natural language processing problem. They collected about 22 million log entries and used tokenization to obtain "sentences" (file paths) and "words" (directory and file names). They encoded the directory and file names by training a word2vec-style model (FastText) [56], on the tokenized data, which generated word vectors capturing similarities between related file and directory names. They used the DBSCAN algorithm to cluster word vectors by cosine distance and identified common patterns in file and directory naming. They normalized and reduced the initial dataset by generating regular expressions for the clusters. Then they trained the FastText model using the new, normalized paths. To distinguish anomalous process creation chains from benign ones, the researchers used an autoencoder. They hypothesized that the model's reconstruction error would be lower for common process chains than for rare ones. They trained a bidirectional RNN autoencoder on process chain sequences using the vectors obtained from the previous steps. After training, they tested the autoencoder's ability to detect anomalous process chains by measuring its reconstruction error. They set a threshold value for the reconstruction error, above which a process chain can be considered anomalous.

In Filar and French [57] the researcher present ProblemChild, a graph-based analytic framework designed to identify anomalous parent-child process chains. They focuses on process events from Microsoft ETW data. A directed acyclic graph is constructed from the logs, where nodes represent objects (e.g. process names), edges represent actions taken by objects (e.g. process creation, termination), and metadata describes nodes or edges (e.g., process ID, command line arguments, timestamps). The following features are used to generate a weight for a given edge between nodes (u, v) in the graph: t between process creation and termination, one-hot encoding of u.child and v.parent (100 processes), process-signature, -elevation and -integrity information, if process is running as system, parent-child user mismatch, entropy of process name and of command line, and lastly TF-IDF (n-grams) of the command line arguments. These features form vector that is passed to XGBoost Chen and Guestrin [58], a gradient boosted trees model, to assign edge weights based on the maliciousness of a given parent-child pair. Louvain community detection is then applied to segment the weighted graph, providing a structure for identifying rare process chains and group attacks. ProblemChild was trained on multiple datasets, both malicious and benign, to establish a normal baseline. The model was then tested on real-world and simulated data, with the goal of determining an ideal threshold for identifying malicious communities.

6.4 System Log Based

DeepLog Du *et al.* [59] is a technique that views each parameter value vector sequence in a log key as a separate time series, employing a Long Short-Term Memory (LSTM) based approach to construct individual LSTM networks for each unique log key value's parameter value vector sequence. The goal of the LSTM model's training process is to minimize the error between the prediction and the observed parameter value vector using mean square loss. Anomalies are detected by comparing the predicted and observed parameter value vectors using mean square error (MSE). This method is particularly effective in detecting a variety of performance anomalies, as parameter values in log messages often capture crucial system state metrics.

Building on the concept of using machine learning for anomaly detection, *LogBERT* Guo *et al.* [60] is a framework designed to detect anomalies in log sequences. Inspired by BERT, a widely used transformer model, LogBERT is trained through self-supervised tasks to learn the patterns of normal log sequences. The authors propose two self-supervised tasks for training LogBERT: Masked Log Key Prediction (MLKP) and Volume of Hypersphere Minimization (VHM). The idea behind anomaly detection in LogBERT is that, being trained on normal log sequences, it can predict masked log keys with high accuracy if a test sequence is normal. If the true log key is not in the predicted set, the key is considered anomalous.

In *LogFiT*, Almodovar *et al.* [61] applies recent developments in Deep Learning and Natural Language Processing (NLP) to detect anomalies in system logs. It relies on "foundation models," pre-trained on large, multi-modal datasets. The foundation model used is Longformer, an advanced version of the RoBERTa model, which surpasses BERT. LogFiT learns from normal system logs to understand their linguistic patterns and properties. During training, it predicts masked tokens in these logs, minimising the difference between its predictions and actual tokens using cross-entropy loss. For training, LogFiT uses a self-supervised approach, using masked token prediction. In the application phase, LogFiT uses its understanding of normal logs to identify anomalies.

For these methods, each method uses different preprocessing, training, and optimization techniques to detect anomalies in system logs. While DeepLog uses LSTM networks and a time-series approach, LogBERT and LogFiT leverage Transformer-based models with various training and optimization strategies to understand log sequences and identify anomalies.

6.5 Summary

The literature review provides an overview on the evolution and application of sequence models from RNNs to LSTMs, and more recently, to Transformer models in Host-based Intrusion Detection Systems (HIDS). These models have been employed extensively in System Call Based methods, and have started to find their way into System Log Based methods. Notably, the research community has made significant strides in implementing various neural networks architectures in anomaly detection tasks, achieving impressive results.

The transition from traditional RNNs to LSTMs was driven by the latter's ability to mitigate the vanishing gradient problem, thereby facilitating the learning of long-term dependencies in sequence data. This is particularly important for system call sequences where an anomalous event might be better identified considering a broader context.

Recently, the application of Transformer models is gaining attention, particularly in the context of system log analysis. Models such as BERT, LogBERT and Longformer have shown promise in understanding the patterns in system logs for anomaly detection. They offer an advantage in terms of their attention mechanism that allows for parallel computation and better handling of long-range dependencies, which might be a limiting factor in RNNs and LSTMs.

However, despite the advantages offered by these models, their application to Process Tree Based methods appears to be a largely unexplored area. The current methods for process tree analysis predominantly rely on graph-based techniques, PID and PPID relationships, and clustering-based algorithms. As identified in the literature, these methods might not scale well when dealing with large amounts of data and rely heavily on manually defined rules.

The proposed research aims to explore the application of Transformer models to Process Tree Based methods. Given their proven success in sequence and log data, Transformer models could potentially improve the efficiency and accuracy of process tree based anomaly detection. The proposed research will be a significant contribution to the field as it intends to bridge this gap in the current state-of-the-art.

Moreover, the proposed research will also include a comparative study with the existing RNN/LSTM based methods. This will provide valuable insights into the relative performance and trade-offs between these different approaches. It will also serve as a comprehensive reference for future studies seeking to optimize anomaly detection methods.

In conclusion, this research proposal presents a promising direction for improving HIDS. It has the potential to significantly advance the field by leveraging state-of-the-art Transformer models in a new application domain.

6.5.1 Data Gathering Preprocessing

The most suitable dataset was identified based on the literature review. The OpTC dataset was chosen based on the type of logs, the large size, both in numbers of client and amounts of data, and that the data is nicely structured in a JSON format. The dataset was then collected from the repository. The initial step involves filtering out irrelevant data. In this study, we focus on Process Creation events, which can be found in both Windows Security Event logs and Sysmon logs, or other logging applications. This was done by simply filtering on two fields that correspond to a process create event.

Chapter 7

Experiments and Results

7.1 Dataset Preprocessing

7.1.1 Obtaining OpTC

The dataset contains three folders: ecar-bro, ecar, and bro. For this project, we are only interested in the ecar data. Benign data from the first period, 17-18 September, and the last period, 20-23 September, is downloaded. Evaluation data from Day 1 is also downloaded. The total size on disk for the benign data is was around 1.6TB, and the total size for the evaluation data was 179 GB. However, the amout of data was drastically reduced when extracting the features.

7.1.2 Extract benign and malicious process create events

The first thing that needs to be done is extracting the events we're interested in. These are process creation events, i.e. events where action is CREATE and object is PROCESS.

For benign processes, we can simply collect all that are present in the dataset. However, the dataset is not directly labeled, but rather split into days of normal activity and days of attacks. For the dataset from the attack-days, the red team has described and reported the process. The "ground truth" report specifies which hosts are attacked during that day, and include the malicious agent running on the host, malicious process IDs and IP-addresses. In order to extract the malicious events, we filter on the attacked hosts and PIDs for the respective host. This malicious data is put into its own dataset and processed in similar fashion as the benign data. However, this dataset is only used for evaluation.

The logs are nicely structured in the JSON format, but because of the large file size, the files need to be parsed with ijson c[62], an iterative JSON parser. The files are read iteratively in “slices”, and for each slice the function appends all process-create events to a new list. This list is then normalized and flattened, as a flattened JSON object is easier to work with than a nested one. Some columns need to be renamed as punctuation marks causes problems in downstream tasks.

Lastly, the objects are grouped by hostname, which is needed for building process relationships as PIDs are unique only per host.

7.1.3 Building trees and chains

We are interested in the process parent-child relationships, but they must be constructed. This is done using a python library from Microsoft Threat Intelligence called msticpy, which is for InfoSec investigation and hunting [63]. In the library, a root process is defined as a top-level process, which only has children. A leaf process is at the end of the tree and only has parents. Branch processes are in between root and leaves.

Code listing 7.1: Schema for the OpTC dataset, used for msticpy

```
optc_schema = {
        "process_name": "image_path",
        "parent_name": "parent_image_path",
        "process_id": "pid",
        "parent_id": "ppid",
        "host_name_column": "hostname",
        "time_stamp": "timestamp",
        "cmd_line": "command_line",
        "logon_id": "actorID"
    }
```

The function takes as input the process create events and their schema, which must be adapted. Firstly, the function *build process tree* constructs the process tree, from which process chains can be constructed. The function iterates over each level of the process tree, starting from the bottom. For each level, every process residing at this level is identified and iterated over. If a process is a leaf node, its ancestors are retrieved using *get ancestors*. For every ancestor of the process, it extracts the process name, image path, and command line, and appends these as a list which represents the process chain.

7.1.4 Cleaning the file paths

The function takes a string as input and processes it using a series of operations: convert the string to lowercase, remove the '.exe' extension if it exists, replace spaces with underscores, and several regular expressions generalizes specific patterns in the string, such as file paths, version numbers, and others. For example, usernames are converted to "user." Every operation is tried on the string, passing if error occurs.

Normalization

The process paths include variables such usernames and version numbers. The name of the user who ran the program is not of interest in this study, as it does not carry the semantic information we are looking for. Similarly to how SIDs were excluded in [64] and usernames normalized in [57], we chose to normalize the usernames. In an environment, there can be several version of the same applications running. We regard the specific version numbers as not important to the relationships between the processes, on the contrary, training a model on too many tokens could result in weak learning. So, similar to the preprocessing in [35], we use Regex to perform manual stemming of the process paths. Stemming is a technique to reduce words to their base. With these two techniques, we conduct normalizing and dimensional reduction. by normalizing, the goal is to improve performance of the models, as they should generalize and recognize pattern more easily. By reducing the vocabulary, memory requirements are lowered and should result in faster processing. However, its important not to overstem or understem [65], which can lead to information loss or incomplete normalization, respectively. Therefore, we chose to stem the version numbers down to the major version, as this seemed like a good balance. Aslo, as handling N/A fields can be a challenge for some functions, all N/A fields are replaced with a blank value.

7.1.5 Building the datasets

Data splitting is done so that we have a sets for different purposes. Typically, the data in a machine learning model is divided into three subsets: the training set, the validation set, and the test set. The *training set* is used to train the model, allowing it to learn and optimize its parameters. The *validation set*, also known as the cross-validation or model development set, is used to adjust learning process parameters. It ranks the model's accuracy and aids in model selection. The *test set* evaluates the final model by comparing it with previous data sets, thus providing an assessment of the model and its algorithm. Data splitting helps prevent data leakage, which occurs when information from the evaluation or test set unintentionally influences the training process. To ensure unbiased evaluation and accurate estimation of model performance, it is crucial to keep the training, validation, and test sets independent and avoid any overlap between them. The splitting also helps the model to generalize better to new data and not just memorize the training data. It provides aims to prevent overfitting and gives a more accurate measure of how the model will perform in real-world scenarios. To ensure a high amount of training data, data sets are often split using an 80-20 or 70-30 ratio of training to testing data. [66] There are several methods for implementing a data split. One simple being random sampling, which is a data sampling method that prevents bias towards various data characteristics. However, this method may lead to uneven data distribution. In this study, an 80-10-10 split with random sampling was chosen as the data splitting strategy.

We utilize sklearn's *train test split* [67] to first shuffle the data, with the *random state* value controlling the shuffling in order to have a reproducible output. Since the funtion splits the data in two, it is applied twice. The final dataset er 80% for training, 10% for validation and 10% for testing. All datasets are batched for better performance.

7.2 Implementation

7.2.1 Environment

Our choice landed on Tensorflow [68] and Keras [69] for the construction of our machine learning models. This choice was based on their solid documentation and user-friendly nature, which significantly simplify the implementation process. These libraries offer intuitive high-level APIs, making it easy to build and experiment with different models, which was particularly important given our focus on a variety of sequential architectures.

As for the execution of the code, we decided on Google Colab [70] due to several factors. First, Google Colab provides an environment of simplicity, reducing the need for extensive setup or configuration, thus allowing us to concentrate on the machine learning aspects of our project. Second, Google Colab offers substantial computational resources, including access to GPU processing power, which is critical when training the models. Finally, Tensorflow, being developed by Google, integrates seamlessly with Google Colab, which should make for a smooth workflow.

It should be noted that while our models are not overly complex or large in scale. Thus, the combination of Tensorflow, Keras, and Google Colab forms an ideal solution, providing us with the required tools and resources to successfully implement our project without unnecessarily complex infrastructure or computational requirements.

7.2.2 Feature Extraction and Engineering

Process Path

The available features for the process create events are many, as described in *Process Create Event* chapter. For this project, the full process path was chosen as the sole feature, similar to the approach taken in (Patel et al. 2020) [35]. The reason for this is that we mainly want to study the semantic relationship between processes, and for that purpose the paths will suffice. Using only process names would be overly simplistic, as malicious programs could easily change their names to avoid detection (Patel et al. 2020) [35]. By considering the full path, we also take into account the directory where the executable file is located, allowing us to distinguish between processes run from different directories, such as system32 and temp, even if they share the same name.

Tokenization and Vectorization

A relatively simple tokenization technique was employed to represent the input data. Each path is treated as a single token and assigned a unique index or token ID. Unlike a wordpiece tokenizer, which breaks words into subtokens, we chose not to split the paths into smaller units.

While wordpiece tokenizers have proven to be effective for natural language processing tasks, their usefulness relies on the assumption that similar words carry similar meanings. This assumption does not hold true for processes, as the name of a process does not necessarily reflect its actual function. By using a straightforward tokenization approach, we ensure that each process is treated as an individual unit without introducing potential inaccuracies based on shared subtokens.

```
vectorize_layer = tf.keras.layers.TextVectorization(
 max_tokens=1000,
 standardize='lower',
 split='whitespace',
 output_mode='int',
 output_sequence_length=32,
)
```

Code listing 7.2: Tokenizer

A keras.layers.TextVectorization is used for vectorization and tokenization. The text is lowered and split on whitespaces. This layer is used to build the vocabulary, which maps tokens to token-IDs, which is a technique not far from one-hot encoding. The layer can then convert process chains to sequences of token IDs. The sequences are also padded to handle their variable length. This is done to ensure the model gets a consistent input. The padding token is a "0" , which is masked or ignored by the later layers.

Embedding Techniques

In order to enhance the representation of the input text, we experimented with word2vec embeddings instead of relying solely on index encoding. The Word2vec vectors should capture the semantic similarities and relationships of the tokens. We hypothesized that using word2vec embeddings would improve the performance of the model by providing richer features. We used the Gensim [56] framework to generate the word2vec embeddings because it offers a fast and easy way to create and manipulate word vectors.

```
model = gensim.models.Word2Vec(sentences=train_list, vector_size=8, min_count=1)
model.train(corpus_iterable=train_list, total_examples=model.corpus_count, epochs=5)
)
```

Code listing 7.3: Word2Vec

Code listing 7.4: w2v example for "powershell"

```
Vector Dimension: 8
Most Similar Tokens to 'powershell':
[('\\device\\harddiskvolume1\\program_files\\windows_defender\\msmpeng',
  0.781205415725708),
('\\device\\harddiskvolume1\\program_files_(x86)\\google\\chrome\\application\\
chrome',0.7426756620407104),
 ('\\device\\harddiskvolume1\\windows\\system32\\compattelrunner',
  0.731886088848114),
 ('\\device\\harddiskvolume1\\windows\\system32\\conhost', 0.7241727113723755),
 ('\\device\\harddiskvolume1\\windows\\system32\\devicecensus',
  0.7036294937133789)]
Vector representation of 'powershell':
array([-1.4884729 , -1.7543703 , -1.7553613 ,  0.30112946, -2.1499376 ,
       -2.0736372 ,  1.7052546 , -2.9479523 ], dtype=float32)
```

7.2.3 The Models

RNN and LSTM

We employed two types of Recurrent Neural Networks (RNNs) - SimpleRNN and Long Short-Term Memory (LSTM) - to carry out sequence analysis for the task of next-token prediction. This task is used for anomaly detection in a binary class, using the loss as an anomaly score. Given that our sequences are typically short, ranging from 2 to 16 in length, and that our dataset consists of 1.6 million such sequences, RNNs are suitable due to their ability to handle sequential data and capture temporal infromation.

Both models are implemented using Keras' Sequential API [71], which allows for linear stacking of layers, making it highly suitable for these relatively straightforward model architectures.

SimpleRNN Model The architecture of the SimpleRNN model begins with an Embedding layer. This layer transforms the integer-encoded vocabulary into fixed-size dense vectors. The input dimension is set to the size of the vocabulary, while the output dimension is determined by a tunable hyperparameter which ranges from 8 to 64 in steps of 8. The 'mask_zero' parameter is set to True, allowing the model to handle sequences of variable length.

Following the Embedding layer, we introduce a SimpleRNN layer. The number of units, is a hyperparameter that varies between 8 and 64 in steps of 8. We also apply a dropout rate of 0.5 to prevent overfitting, which aids in model generalization.

The last layer is a Dense layer that uses a softmax activation function. The number of units equals the size of the vocabulary, making the output a probability distribution across all potential next tokens.

Model summary after hyperparameter tuning:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 32, 32)            5824

 simple_rnn (SimpleRNN)      (None, 64)              4160

 dense (Dense)               (None, 182)               11830

```

```
=================================================================
Total params: 21,814
Trainable params: 21,814
Non-trainable params: 0
_________________________________________________________________
```

LSTM Model The LSTM model also begins with an Embedding layer and ends with a Dense layer identical to the SimpleRNN model.

The middle layer consists of an LSTM unit. LSTMs, a variant of RNN, are particularly for learning long-term dependencies, that also might be a beneficial property even in the context of relatively short sequences. The number of LSTM units, is a tunable hyperparameter that ranges from 8 to 64 in steps of 8. Again, a dropout rate of 0.5 is applied.

The tunable hyperparameters allow for model optimization based on the characteristics of the specific task and dataset, enabling the models to balance model complexity and computational efficiency, a crucial consideration given the sizable dataset of 1.6 million sequences.

Model summary after hyperparameter tuning:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 32, 40)            7280

 lstm (LSTM)                 (None, 128)              53760

 dense (Dense)               (None, 182)               23478

=================================================================
Total params: 84,518
Trainable params: 84,518
Non-trainable params: 0
_________________________________________________________________
```

Transformer

First a set of hyperparameters are define, which are used in the layers. These include the number of transformer layers, the dimensions of the model, the dimension of the intermediate layer in the transformer, and the number of attention heads in the multi-head attention mechanism of the transformer. These parameters are subject to optimization via hyperparameter tuning, with appropriate ranges and steps defined for each. Dropout rate is fixed at 0.5 to prevent overfitting. A norm epsilon value is also defined for the layer normalization process.

We start by defining an input tensor with shape set to 32. This allows for a flexible handling of our sequences which usually range from 2-10 tokens, but can be padded up to a length of 32.

The first layer is a TokenAndPositionEmbedding layer. This layer first embeds the token ids into continuous vectors and then adds positional encoding to the embeddings, providing positional context to the model. The layer also apply a mask to ignore padding tokens.

Following the embedding layer, we apply layer normalization and dropout to regularize the model and prevent overfitting. The epsilon value in the layer normalization is a hyperparameter.

Subsequently, we add a number of transformer encoder layers as defined by a hyperparameter. Each encoder layer includes multi-head self-attention and a feed-forward network, both of which have residual connections and layer normalization. The dimensionality of the feedforward layer is a hyperparameter, and another hyperparameter defines the number of attention heads in the self-attention mechanism.

This completes the encoder which takes token ids as input and returns the transformer-encoded tokens.

Finally, we attach a masked language model head on top of the encoder model. This model receives the encoded tokens and outputs predicted token probabilities for masked positions in the sequence.

The transformer model is a suitable choice for sequence analysis due to its ability to handle dependencies between tokens regardless of their distance in the sequence, which is an advantage over traditional RNNs or LSTMs. Moreover, its attention mechanism can offer insight into the relationships between tokens, potentially aiding in the detection of anomalies.

Code listing 7.5: Transformer Encoder after hyperparameter tuning

```
________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer                 [(None, 32)]               0

 TokenAndPositionEmbedding  (None, 32, 48)            10272

 LayerNormalization         (None, 32, 48)            96


 Dropout                    (None, 32, 48)             0

 TransformerEncoder x5      (None, 32, 48)            10806

=================================================================
Total params: 64,398
Trainable params: 64,398
Non-trainable params: 0
```

Code listing 7.6: Transformer model after hyperparameter tuning

```
_________________________________________________________________
Model: "model_1"
______________________________________________________________
 Layer (type)                   Output Shape         Param #
==============================================================
 input_2 (InputLayer)           [(None, 32)]         0

 model (Functional)             (None, 32, 48)       64398
 input_3 (InputLayer)          [(None, 16)]         0

 masked_lm_head (MaskedLMHead)  (None, 16, 182)      11366

=================================================================
Total params: 67,028
Trainable params: 67,028
Non-trainable params: 0
_________________________________________________________________
```

7.2.4 Embedding adaption

The adaption of the models to using pre-trained word2vec embeddings requires a modification of the initial embedding layer. Instead of learning the weights during the training process, we initialize the embedding layer with an embedding matrix, containing the weights for the pre-trained embeddings. The 'trainable' parameter is set to False, meaning that these weights are kept fixed and are not updated during training. This is done in order to better measure the impact of using these embeddings. The embedding matrix has shape vocabulary-size*vector-size, where the vector size is the dimension of the embeddings. This is set to a conservative size of 8, to be able to learn good embeddings even with little data.

To use the word2vec embeddings with the transformer, the TokenAndPositionEmbedding layer had to be split up into a a token embedding layer (which uses the pre-trained embeddings) and the positional embedding used for the attention mechanism. These two embeddings are then simply summed together. For this model, both setting the 'trainable' parameter to False and True was attempted.

However, one potential disadvantage of this approach is that since the embeddings are not updated during training, this part of the model is not able to adapt it further. In this context, though, the pre-trained embeddings are trained on the same corpus, so they should fit the data well already.

Code listing 7.7: Word2Vec Embedding layer, equal for all models

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 32, 8)             1456
```

7.2.5 Hyperparameter Optimization

We leverage the Keras Tuner, a powerful and easy-to-use hyperparameter tuning library, to optimize the performance of the models.

We use the Hyperband tuning algorithm provided by Keras Tuner. It is an adaptation of the random search algorithm with a more efficient resource allocation strategy. It should help us in finding the optimal hyperparameters in fewer iterations, thereby speeding up the tuning process.

Model Tuning We instantiate the Hyperband tuner by specifying the model-building function, the objective to maximize (in this case, validation sparse categorical accuracy), the maximum number of epochs for each model to be trained, and the reduction factor of the number of epochs and models for each bracket in Hyperband.

To prevent overfitting and reduce unnecessary computations, we introduce an early stopping callback that halts the training process when the validation loss has not improved for a number of epochs, called patience. We chose to set this to 2, as the models are relatively small, and should converge fast.

By leveraging this strategy, we can find an optimal set of hyperparameters that would lead to better model performance in terms of validation sparse categorical accuracy. This in turn improves the effectiveness of our anomaly detection task by increasing the model's prediction accuracy.

7.2.6 Training

Training our models encompasses several techniques tailored to our specific task of anomaly detection in sequences. This section will discuss the main strategies and techniques we employ, specifically focusing on the training objective, the optimizer, and the loss function.

Training Objective

The training process differs slightly for the two types of models implemented, the RNNs and the transformer.

For the RNN and LSTM models, we use next-token prediction. In this approach, the model is trained to predict the next token in a sequence given the preceding tokens. The aim is to allow the model to learn the temporal dependencies inherent in the sequences. It can capture normal patterns in the data, and anomalies can then be detected as deviations from these patterns.

On the other hand, for the Transformer model, we utilize a masked language model (MLM) technique. Using this technique, certain tokens in the sequence are masked out, and the model's objective is to predict these masked tokens based on the context provided by the unmasked tokens. This training technique, is particularly effective for Transformer models as it allows the model to leverage the attention mechanism, where all tokens in the sequence make up the context. It also provides a more granular view of sequence structure and has been shown to be successful in sequence analysis tasks, specifically for anomaly detection [60, 61, 72, 73]. In our training, we set the mask rate to 0.25 and max number of masks to 16. This means that every token has a 25% of being masked, and the maximum value we set equal to the longest sequence observed in training.

Optimizer

All our models utilize the Adam optimizer for training. Adam, short for Adaptive Moment Estimation, is a popular optimization algorithm in deep learning due to its efficiency and low memory requirements. It combines the benefits of two other extensions of stochastic gradient descent: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in online and non-stationary settings [74]. This makes Adam well-suited for our task, as we are dealing with multiclass data with many sparse values, or sequences with padding.

Loss Function

The models are trained using the Sparse Categorical Cross-Entropy loss function [75]. This loss function is suitable for our task as we are dealing with multi-class classification problems (predicting the next token or masked token out of a vocabulary). The 'sparse' variant is particularly useful when the classes are mutually exclusive, i.e., each token can belong to one and only one class, which is the case in our task.

Anomaly Detection

Anomaly detection in our models is based on the concept of reconstruction loss. Given that our models are trained to predict the next token (for the RNNs) or masked tokens (for the Transformer), the loss value can serve as a measure of how well the model 'understands' a given sequence. Normal sequences should have lower loss values as they align with the patterns learned during training, while anomalous sequences will yield higher loss values as they deviate from these patterns. Therefore, by monitoring the loss value, we hypothesize that we can identify anomalies in the sequences, as others also have [61, 72, 76, 77].

7.2.7 Evaluation

7.3 Results

7.3.1 Dataset

Benign Class

In Table 7.1, some statistics for every sequence position is shown, these include the count of tokens at that position, unique tokens at the position, and the top token and its frequency. Here, it is apparent that the number of sequences decrease as the length grows. In the 'top' field the paths are shortened to just a backslash. The longest sequence is of 16 tokens, and is sits alone at the top.

In Figure 7.1 and Figure 7.2, the distribution for tokens and top 250 chains are shown, respectively. Without doing heavy statistics, it looks like the distributions has a classic 'head' and 'long tail', where a few tokens and chains account for

Table 7.1: Per Position Statistics

index	count	unique	top	freq
0	1618967	124	\svchost	602017
1	1618967	161	\cmd	418026
2	1273445	149	\mantra	369230
3	1084312	145	\python	384698
4	926123	138	ping	278459
5	587551	141	\python	122529
6	353226	127	\cmd	107504
7	203694	117	reg	62040
8	81273	113	ping	34272
9	36035	93	ping	13486
10	11670	43	\python	5034
11	6421	36	\cmd	3239
12	3886	22	\qwinsta	770
13	855	15	netstat	684
14	37	6	mstsc	22
15	1	1	netstat	1

Table 7.2: Chain Statistics

attribute	value
count	1618967
unique	8558
top	\svchost \cmd \mantra \python ping
freq	272923

most of the events, and a large number of them only has 1 count. The number of chains included in the graphics had to be cut for from 8558 to 250. Top and bottom tokens in the vocabulary is shown in Table 7.3. Note that this is for the entire benign dataset, not the vocabulary used for training, the difference though is only one token. In Table 7.2, the total count of chains are shown, and how many are unique. Also the top chain is shown along with its count. Again, the path here is shortened.

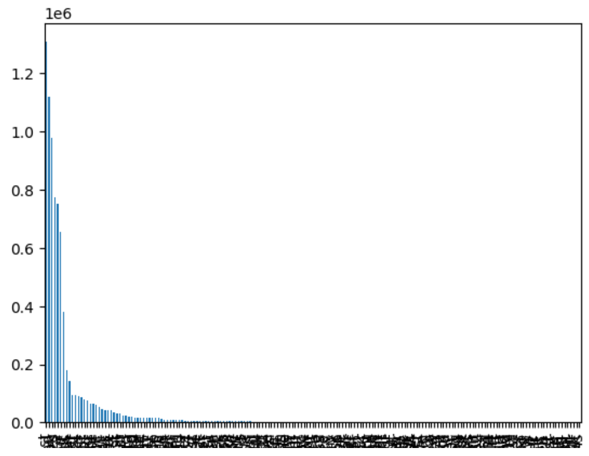


Figure 7.1: Token Distribution

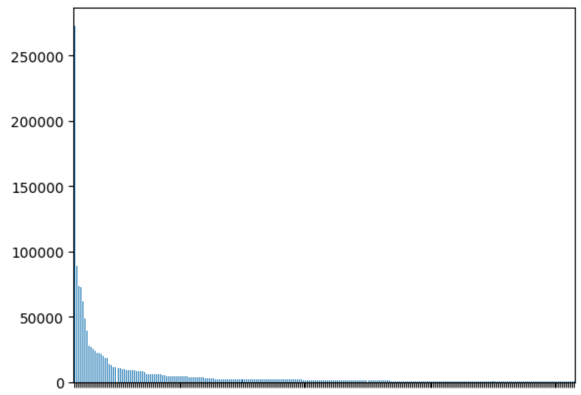


Figure 7.2: Chains Distribution (Top 250)

Malicious Class

In Table 7.4, the same statistics as in Table 7.1 is shown. The number of chains seems to have a significant drop off after length of 4 tokens. We can also note the high number of chains that end with Powershell.

In Figure 7.3 and Figure 7.4, the distribution for the tokens and chains are plotted. The shapes of the plots are not too different from the benign distribu-

Table 7.3: Vocabulary Statistics

Token	Count
\device\harddiskvolume1\windows\system32\svchost	1307667
\device\harddiskvolume1\windows\system32\cmd	1118480
\device\harddiskvolume1\python27\python	977564
\device\harddiskvolume1\windows\system32\services	774236
\device\harddiskvolume1\ncr\mantra\runtime\scripts\mantra	753334
...	...
\device\harddiskvolume1\windows\system32\secedit	1
powerpnt	1
\device\harddiskvolume1\windows\system32 askmgr	1
\device\harddiskvolume1\windows\system32\werfault	1
\device\harddiskvolume1\windows\system32\cloudnotifications	1
Length: 181	

Table 7.4: Per Position Statistics - Malicious

index	count	unique	top	freq
0	605	10	\svchost	308
1	605	13	powershell	261
2	603	16	\powershell	257
3	308	16	\ping	255
4	39	8	\cmd	17
5	27	7	powershell	15
6	24	8	\powershell	11
7	17	7	\powershell	9
8	12	6	\whoami	6
9	2	2	\mmc	1
10	1	1	mmc	1

Table 7.5: Chain Statistics - Malicious

attribute	value
count	605
unique	45
top	\wmiprvse powershell \ping
freq	254

tions, however these statistics are from a significantly smaller sample. We also note that only 2 chains account for most events. Vocabulary statistics for the malicious class are shown in Table 7.6, and we note that 5 tokens account for most of the events. The number of and percentage of Out-Of-Vocabulary tokens is shown at the bottom. This value is important, as it tells us how many of the tokens is received as UNK for the model. The table is a bit misleading, as the number (13) is unique tokens, and the percentage (23.6%) is the number of instances that the OOV-tokens makes up of all instances. We note that this value is a significant portion the tokens. In Table 7.5, statistics about the chains are shown. We note that these numbers are very small compared to the benign class.

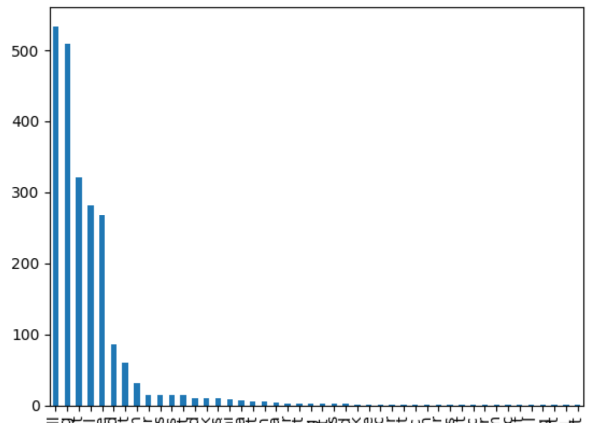


Figure 7.3: Token Distribution - Malicious

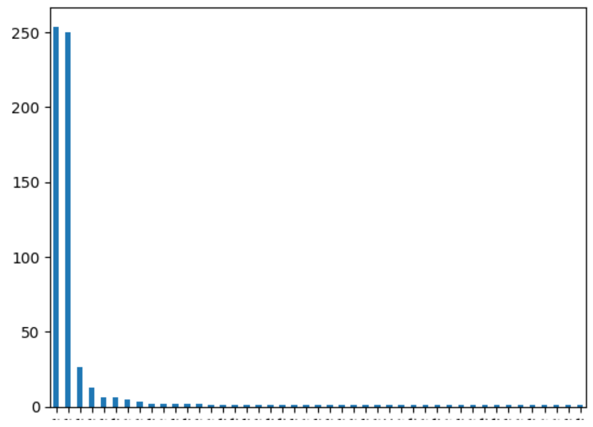


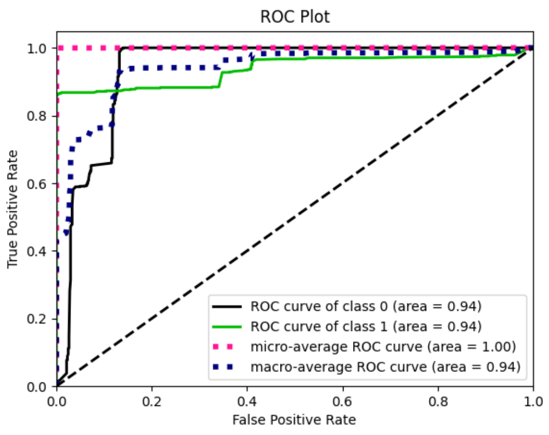
Figure 7.4: Chains Distribution - Malicious

Table 7.6: Vocabulary Statistics - Malicious

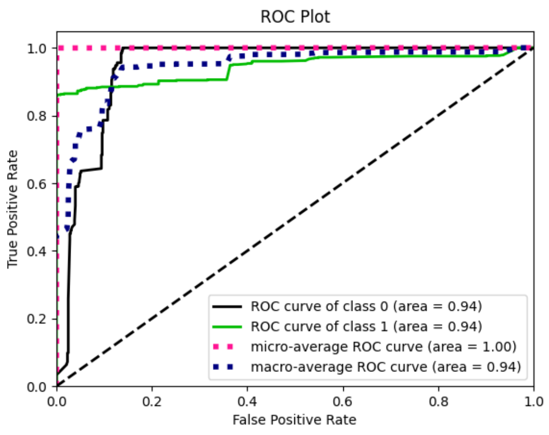
Token	Count
\device\harddiskvolume1\windows\system32\windowspowershell\v1.0\powershell	534
\device\harddiskvolume1\windows\system32\ping	509
\device\harddiskvolume1\windows\system32\svchost	321
powershell	282
\device\harddiskvolume1\windows\system32\wbem\wmiprvse	268
\device\harddiskvolume1\windows\system32\cmd	85
...	...
taskkill	1
reg	1
\device\harddiskvolume1\windows\system32\wscript	1
net1	1
tasklist	1
OOV	13 (23.6%)

Table 7.7: Performance metrics at two percentiles for RNN - Index and Word2Vec

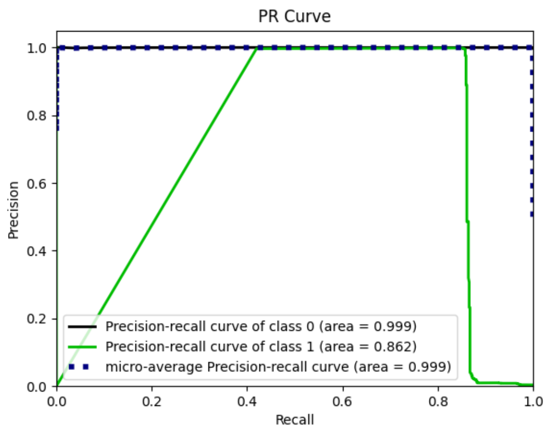
	RNN-Index		**RNN-Word2Vec**	
Metric	**95th %**	**99th %**	**95th %**	**99th %**
ROC AUC Score	0.937	0.937	0.943	0.943
Avg Precision	0.862	0.862	0.864	0.864
F1 Score	0.116	0.38	0.118	0.382
Precision	0.062	0.244	0.063	0.245
Recall	0.868	0.866	0.873	0.863
MCC	0.225	0.456	0.227	0.457
TN	153988	160270	154078	160292
FP	7909	1627	7819	1605
FN	80	81	77	83
TP	525	524	528	522



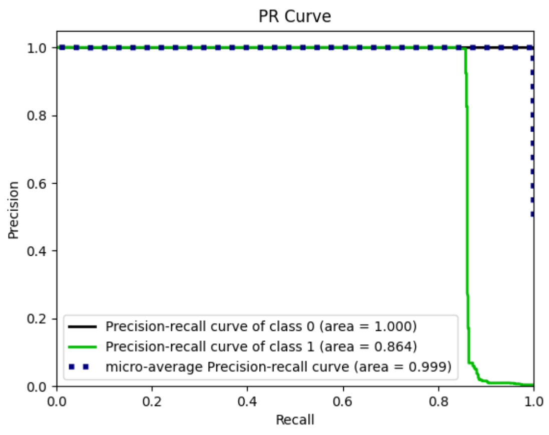
(a) RNN - Index ROC



(b) RNN - W2V ROC



(c) RNN - Index PR plot



(d) RNN - W2V PR plot

Figure 7.5: Plots for ROC and PR for RNN Index and Word2Vec

7.3.2 Models

RNN

LSTM

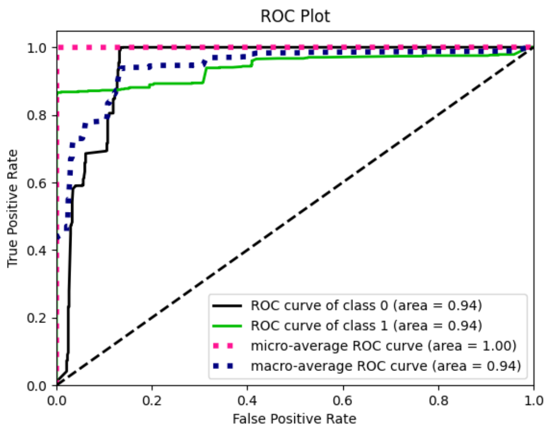
Table 7.8: Performance metrics at two percentiles for LSTM - Index and Word2Vec

	LSTM-Index		**LSTM-Word2Vec**	
Metric	**95th %**	**99th %**	**95th %**	**99th %**
ROC AUC Score	0.941	0.941	0.947	0.947
Avg Precision	0.866	0.866	0.836	0.836
F1 Score	0.12	0.381	0.116	0.382
Precision	0.064	0.244	0.062	0.245
Recall	0.869	0.868	0.881	0.864
MCC	0.229	0.457	0.227	0.457
TN	154231	160270	153880	160287
FP	7666	1627	8017	1610
FN	79	80	72	82
TP	526	525	533	523

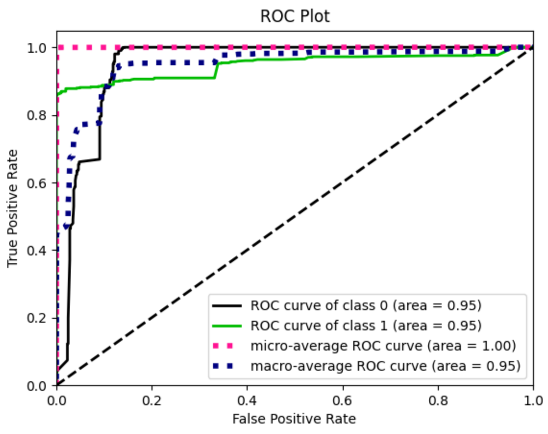
Transformer

Table 7.9: Performance metrics at two percentiles for Transformer - Index and Word2Vec

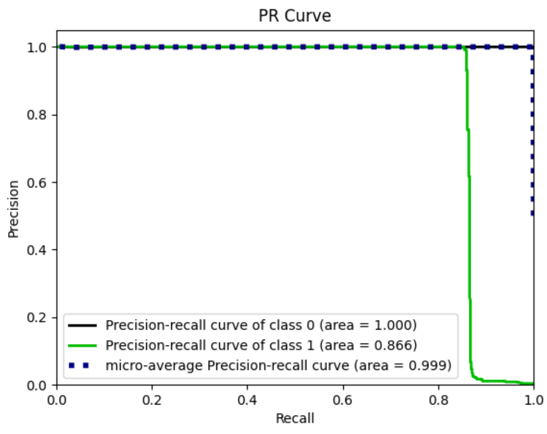
	Transformer-Index		**Transformer-Word2Vec**	
Metric	**95th %**	**99th %**	**95th %**	**99th %**
ROC AUC Score	0.957	0.957	0.95	0.95
Avg Precision	0.546	0.546	0.239	0.239
F1 Score	0.117	0.391	0.116	0.248
Precision	0.063	0.251	0.062	0.163
Recall	0.894	0.883	0.883	0.519
MCC	0.229	0.468	0.227	0.286
TN	153807	160305	153861	160280
FP	8090	1592	8036	1617
FN	64	71	71	291
TP	541	534	534	314



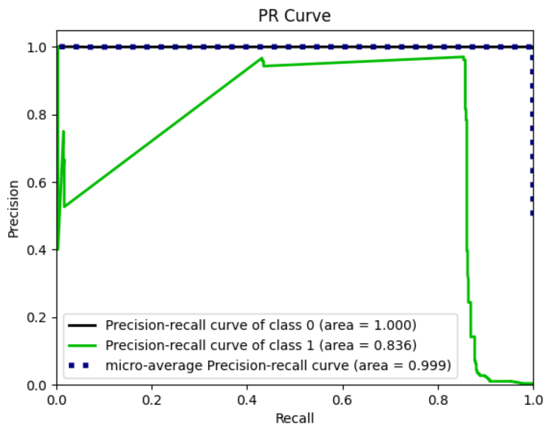
(a) LSTM - Index ROC



(b) LSTM - W2V ROC



(c) LSTM - Index PR plot

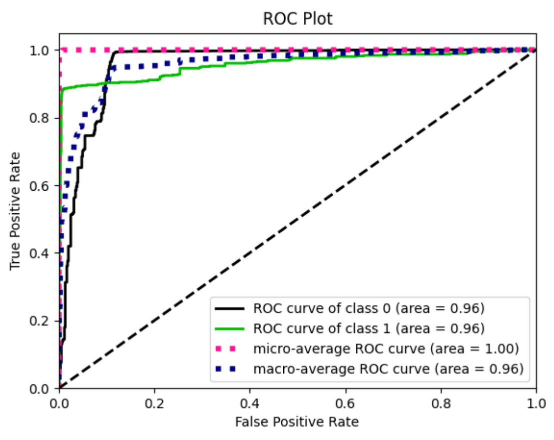


(d) LSTM - W2V PR plot

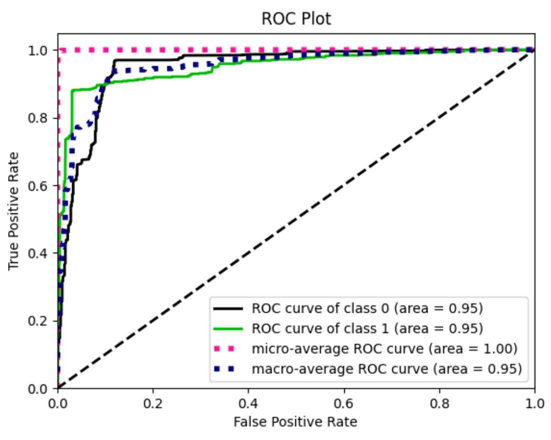
Figure 7.6: Plots for ROC and PR for LSTM Index and Word2Vec

Table 7.10: Transformer - Index (Maskrate=0.25)

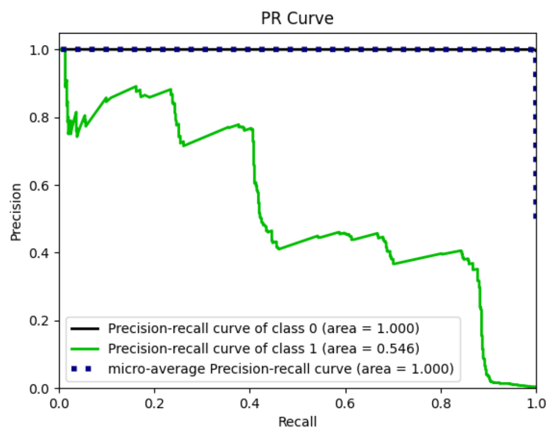
percentile	95, 99
roc_auc_score	0.92
avg_precision	0.41
f1_score	0.098, 0.296
precision	0.052, 0.194
recall	0.737, 0.623
mcc	0.188, 0.344
tn	153821, 160329
fp	8076, 1568
fn	159, 228
tp	446, 377



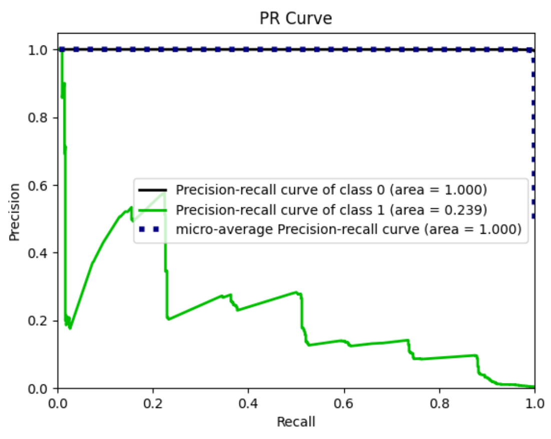
(a) Transformer - Index ROC



(b) Transformer - W2V ROC



(c) Transformer - Index PR plot



(d) Transformer - W2V PR plot

Figure 7.7: Plots for ROC and PR for Transformer Index and Word2Vec

7.3.3 Embedding

To plot the embedding, we first used the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm to identify clusters in the set vectors. After performing the DBSCAN algorithm, the we constructs a 2D t-SNE (t-Distributed Stochastic Neighbor Embedding) plot for visualization. This plot helps reveal the structure of the data and show how the points are grouped into clusters. Each point on the t-SNE plot represents a data vector, and the location of the point reflects the similarity of that vector to the others. Points that are closer together are more similar. The points are colored according to the cluster assignment from the DBSCAN algorithm, making it possible to visually identify the clusters. Annotations (labels) are added to the five points in each cluster that are closest to the cluster's centroid, providing representative labels for the clusters. The legend outside the plot area helps identify which color corresponds to which cluster.

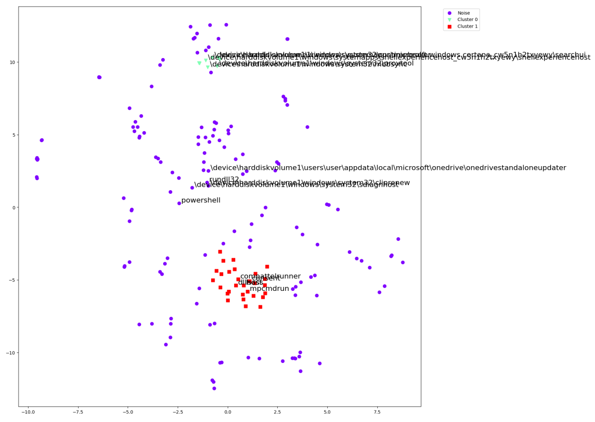


Figure 7.8: TSNE plot for Word2Vec embeddings

Chapter 8

Discussion

8.1 Methodology

We regard the methodology as being a good fit for the research question. Question 1-2 are qualitative, and a literature review was conducted to answer them. Question 3-4 are quantitative, and numerical measurements, as in model evaluation, were used to answer them. The subquestions vary a bit in nature, but they were answered throughout the research. However, more time and resources could have been invested in analysing the result, as this would have given a better picture of model performance, both in terms of comparison between developed models and also against SOTA.

Research questions:

1. What is the state of the art on sequential machine learning methods used on HIDS?
2. Are there any gaps in the current research?
3. Are these methods suitable to be used on process-chains?
4. How do these methods compare against each other and SOTA?

Which leads to the sub-questions:

a. Which datasets are available and most suitable for HIDS?
b. What is the SOTA on process trees?
c. How does mask rate affect performance?

8.2 Results Discussion

8.2.1 Dataset

The unbalanced dataset likely contributes to the somewhat bad results, and it may be beneficial to use techniques such as oversampling, undersampling, or synthetic sample generation (like SMOTE) to address this issue.

8.2.2 Vocabulary

Since the amount of different processes running on a system is non defined, unlike syscalls, a list of processes must be created. This can be done in two ways: either by building the vocabulary from the training set, or by pre-defining a static list of processes of interest, or a combination of both. For example, Powershell instances might be of interest whereas Spotify instances might not be. One way to go about this is to create a list of top-x processes on the system. Malicious actors are moving towards "Living off the Land" (LotL), which is a technique where the actor uses programs already existing in the targeted environment, instead of relying on custom-made software. This makes the malicious behaviour harder to detect for standard antivirus software [78][79]. However, it makes the task of creating a priority list of processes easier. A project called the LOLBAS (Living Off The Land Binaries, Scripts and Libraries) [80] has listed 157 binaries, scripts and libraries that can be used for LotL techniques. By merging a top-x process list with this list we can construct a decent list of processes of interest. However, for this project the chosen method was to build a vocabulary from the training set.

8.2.3 Models

8.2.4 RNN, LSTM and Transformer - Index Encoding

This evaluation presents the performance of the three models - the Transformer Table 7.9, the LSTM Table 7.8, and the RNN Table 7.7 - for anomaly detection based on Sparse Categorical Crossentropy loss. The results are for two thresholds, the 95th and 99th percentile.

Starting with the ROC AUC score, which provides an overall measure of the model's ability to distinguish between classes, we see that the Transformer outperforms the LSTM and RNN with a score of 0.957. The LSTM follows with a score of 0.941, while the RNN lags slightly behind with a score of 0.937.

Average precision, which considers the trade-off between precision and recall, is substantially higher for the LSTM (0.866) and RNN (0.862) models as compared to the Transformer model (0.546). Despite having a higher ROC AUC, the significantly lower average precision of the Transformer model suggests that it may not balance precision and recall as well as the LSTM and RNN models.

In terms of the F1 score, which is the harmonic mean of precision and recall, the Transformer model slightly outperforms the LSTM and RNN models at the 99th percentile, but they perform comparably at the 95th percentile.

Looking at precision, which measures how many of the flagged anomalies are actual anomalies, all models show low scores at the 95th percentile, while at the 99th percentile, the Transformer model has the highest precision. This suggests that the Transformer model may have fewer false alarms at the 99th percentile, but at the 95th percentile, it tends to misclassify normal instances as anomalies more often.

The recall scores, which measure the proportion of actual anomalies that were correctly detected, are similar across all models, with the Transformer model slightly outperforming the others. This suggests that the Transformer model is slightly more effective at catching actual intrusions.

The Matthews Correlation Coefficient (MCC), a balanced measure that is useful even if classes are of very different sizes, is highest for the Transformer model at the 99th percentile, indicating it as the better performing model at this threshold. At the 95th percentile, the MCC is similar for all three models.

Looking at the actual counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), the Transformer model identifies slightly more actual intrusions (TP) and fewer missed intrusions (FN), but has more false alarms (FP) at the 95th percentile. However, at the 99th percentile, the Transformer model has fewer false alarms and more correctly identified normal instances (TN) compared to the LSTM and RNN models.

Summary Overall, while the Transformer model shows a higher ROC AUC score and slightly better performance in terms of F1 score, recall, and MCC at the 99th percentile, its significantly lower average precision score compared to the LSTM and RNN models suggests that it may not balance the trade-off between precision and recall as effectively, particularly at lower thresholds.

The choice between these models would thus depend on the operational needs and constraints of the intrusion detection system. Further improvements might involve optimizing these models for better precision without significantly sacrificing recall, potentially through techniques such as adjusting the decision threshold, using cost-sensitive learning, or applying oversampling or undersampling techniques to handle the imbalance in the dataset.

8.2.5 Embedding impact

Comparing the performance metrics between the models with and without pre-trained Word2Vec embeddings, it's clear that the embeddings have a significant impact on performance.

Transformer The Transformer model shows a slight decrease in the ROC AUC score from 0.957 to 0.95 at both percentiles after applying the Word2Vec embeddings.

Interestingly, average precision for the Transformer model significantly decreased from 0.546 to 0.239. This is a substantial drop, suggesting that the use of Word2Vec embeddings in this model negatively impacts its ability to balance precision and recall.

The F1 score at the 99th percentile decreased significantly (from 0.391 to 0.248) due to lower precision and recall. This suggests that while the model was able to maintain a good balance between precision and recall without embeddings, introducing Word2Vec significantly impaired its performance.

The Matthews Correlation Coefficient also decreased, indicating a decrease in the quality of binary classifications.

LSTM For the LSTM model, the ROC AUC score increased slightly from 0.941 to 0.947 at both percentiles when using Word2Vec embeddings.

However, the average precision score decreased from 0.866 to 0.836. This suggests that while Word2Vec embeddings helped the LSTM model to improve overall class separation, they also slightly reduced its ability to balance precision and recall.

In terms of F1 score, precision, and recall, the LSTM model with Word2Vec embeddings performs similarly to the original LSTM model at the 99th percentile, indicating that the embeddings did not have a substantial effect on the performance of this model in these respects.

The Matthews Correlation Coefficient remained the same, suggesting a consistent quality in binary classifications.

RNN The RNN model showed a minor improvement in the ROC AUC score from 0.937 to 0.943 at both percentiles when using Word2Vec embeddings.

The average precision for the RNN model increased slightly from 0.862 to 0.864. This suggests that Word2Vec embeddings improved the RNN model's ability to balance precision and recall, albeit slightly.

The F1 score, precision, recall, and Matthews Correlation Coefficient for the RNN model remained nearly the same, suggesting that the Word2Vec embeddings had little to no impact on these aspects of the RNN model's performance.

Summary The use of Word2Vec embeddings had varying impacts on the performance of the models. It seemed to negatively impact the Transformer model significantly while only slightly impacting the LSTM model and the RNN model. This suggests that while Word2Vec embeddings can be beneficial, their effectiveness may depend on the specific architecture of the model, and might not be a good fit for a transformer. This could be due to a mismatch with the advanced attention mechanism, which utilizes both token and positional embedding. The underlying reason for this effect could be an interesting point for further investigation.

8.2.6 Transformer - ANUBIS

ANUBIS Anjum *et al.* [64] , is an anomaly detection method utilizing based on a Bayesian Neural Network. Their approach was not covered in the literature review, however, they base their work on the same dataset as this, namely OpTC. Their work can be considered similar, as they utilize a neural network for anomaly detection, on the same dataset. The sampling might be different, though.

```
Transformer
f1_score        0.391
```

```
precision        0.251
recall           0.883

ANUBIS
f1_score        0.998
precision       0.997
recall      1.00
```

The F1 score for the ANUBIS model is significantly higher (0.998) than the Transformer model's score (0.391). This suggests that the ANUBIS model is much more accurate in correctly classifying instances as anomalous or benign.

Precision: Again, ANUBIS outperforms the Transformer, with a precision of 0.997 compared to the Transformer's 0.251. This indicates that when the ANUBIS model classifies an instance as anomalous, it is very likely to be correct. On the other hand, when the Transformer model classifies an instance as anomalous, it is correct only about 25% of the time.

Recall: The ANUBIS model has perfect recall (1.00), meaning it correctly identifies all actual anomalous instances. The Transformer model, while performing decently with a recall of 0.883, falls short of perfect detection and misses about 12

In conclusion, the ANUBIS model significantly outperforms the Transformer model in all the compared metrics. However, a definite comparison cannot be made without more metrics, and the fact that the models are based on slightly different samples.

8.2.7 Transformer - Other algorithms

From the ProcAID paper Read [55], an approach looking at processes, Read did a comparison of different algorithms on the OpTC dataset. Below are the collected result for the hosts, 0201 and 0501. "Stage One" represent ProcAIDs own algorithm.

Code listing 8.1: Transformer, multiple hosts

```
Transformer
f1_score         0.391
precision        0.251
recall   0.883
```

Code listing 8.2: Algorithm Comparison for Host 0201

```
Algorithm Precision Recall F1-Score
Stage One 33.871 100.00 50.602
NewEdge 0.000 0.000 0.000
K-Means 2.365 53.846 4.531
HBOS 17.241 23.809 20.000
One-Class SVM 6.410 23.810 10.101
```

Code listing 8.3: Algorithm Comparison for Host 0501

```
Algorithm Precision Recall F1-Score
```

```
Stage One 11.852 84.211 20.779
NewEdge 0.000 0.000 0.000
K-Means 2.571 56.250 4.918
HBOS 0.000 0.000 0.000
One-Class SVM 1.546 37.500 2.970
```

The Transformer model outperforms all other models with an F1 score of 0.391. The F1 score, indicates that the Transformer model has a good balance between precision and recall compared to the other models. In contrast, the next best algorithm (Stage One) has a significantly lower F1 score of 20.779.

The precision of the Transformer model is the highest precision among all the models, indicating that it provides the fewest false positives. The next best precision is from the Stage One algorithm, but it is significantly lower, and three of the models (NewEdge, HBOS, One-Class SVM) have very low precision scores, indicating they might often misclassify normal instances as anomalies.

The recall of the Transformer model, is the second-highest among all algorithms, indicating that it catches a high proportion of the actual intrusions. The Stage One algorithm has a slightly higher recall, which means it might detect a higher percentage of anomalies, but considering its lower precision, it's also likely to produce more false positives.

In summary, the Transformer model demonstrates superior performance over the algorithms in terms of precision and F1 score and has a high recall. This indicates that it balances the trade-off between identifying as many intrusions as possible (high recall) and minimizing false alarms (high precision), leading to the highest F1 score among the compared models.

However, the algorithms were only tested on two hosts that had been attacked, while the transformer was tested on a much larger part of the dataset, also including data from these hosts. This skews these result. Still, this indicates that there is potential for sequence model in process analysis.

8.2.8 Transformer Mask Rate 0.5 / 0.25:

In this evaluation, two configurations of the same Transformer model are compared. The only difference is the masking rate during evaluation, with one model using a mask rate of 0.5 and the other 0.25. The hypothesis under consideration is that a higher mask rate (0.5) provides more granular loss data, which leads to better discrimination between the two classes.

From the given metrics, it appears that the hypothesis holds true. The Transformer model with a higher mask rate of 0.5 generally performs better across all metrics when compared to the one with a lower mask rate of 0.25.

The ROC AUC score of the model with a mask rate of 0.5 is significantly higher than that of the model with a mask rate of 0.25 at both percentiles (0.957 vs 0.92). This indicates that the model with a higher mask rate is better at distinguishing between the two classes.

Average Precision: Similarly, the average precision of the model with a mask rate of 0.5 is substantially higher than that of the model with a mask rate of 0.25 at both percentiles (0.546 vs 0.41). This suggests that the model with a higher mask rate is better at balancing precision and recall, leading to fewer false positives and false negatives.

The F1 score, is also higher in the model with a mask rate of 0.5 as compared to the model with a mask rate of 0.25 at both percentiles.

Precision:The model with a mask rate of 0.5 also outperforms the other model in terms of precision.

Recall:The recall of the model with a mask rate of 0.5 is considerably higher than that of the model with a mask rate of 0.25 at both percentiles. This suggests that the model with a higher mask rate is more capable of identifying true positives, thus lowering the chance of false negatives.

Matthews Correlation Coefficient: Is again higher for the model with a mask rate of 0.5, suggesting that this model provides better binary classifications.

Confusion Matrix (TP, FP, TN, FN): From the confusion matrix, the model with a mask rate of 0.5 produces fewer false negatives and more true positives, suggesting that it is better at correctly identifying actual anomalies.

In conclusion, the comparison suggests that the Transformer model with a higher mask rate of 0.5 performs better in anomaly detection. Thus, this supports the initial hypothesis that a higher mask rate provides more granular loss data, thereby improving the model's ability to discriminate between classes.

Chapter 9

Conclusion and Future Work

In conclusion, this study has explored the performance of three popular sequence models - RNN, LSTM, and a Transformer - in the task of anomaly detection on the OpTC dataset. Our findings indicate that while the Transformer model demonstrated slightly superior performance compared to the RNN and LSTM models, it still leaves considerable room for enhancement.

Furthermore, we compared the Transformer model with state-of-the-art models in anomaly detection. While the Transformer exhibited superior performance to several algorithms presented in ProcAID Read [55], it was outperformed by ANUBIS Anjum *et al.* [64], a Bayesian Neural Network-based anomaly detection approach.

These findings not only underscore the utility of Transformer models in anomaly detection tasks but also highlight the potential of more advanced techniques such as Bayesian Neural Networks. This indicates a promising direction for future research and development to improve the efficacy of anomaly detection systems.

9.1 Future Work

Future research efforts in this domain can pursue several potential directions, which are outlined in the following subsections:

9.1.1 Feature Engineering

A deeper exploration into feature engineering could offer beneficial insights and improve model performance. As Filar and French [57] suggests, a comprehensive analysis of various features including, but not limited to, process name, parent process name, commandline arguments, process path, event subtype, and integrity level could be undertaken. Other potentially meaningful features worth investigating might include normalized process path, whether the process is signed, and the trust level of the signer.

Furthermore, specific attention could be paid to the commandline arguments . As Cocea [81] demonstrated, utilizing the BERT embedding for commandline arguments could yield useful insights. The application of advanced language model embeddings like BERT might unveil complex patterns within the commandlines that simpler feature extraction methods might overlook.

Investigating different types of embeddings can reveal more about the nature and structure of the data. For example TF-IDF that could add additional weights to the process events.

Feature importance can also be assessed to identify the most informative features contributing to the detection of anomalies. This can be achieved by employing various statistical methods or by using machine learning models that provide feature importance scores.

In summary, embedding and feature engineering present promising areas for future research that could significantly enhance the performance and interpretability of intrusion detection systems. Future work in these areas can contribute to the development of more robust and effective IDS models.

Bibliography

[1] S. Institue, *Machine learning what it is and why it matters*, [Online; accessed 18-May-2022], 2022. [Online]. Available: `https://www.sas.com/en_us/insights/analytics/machine-learning.html`.

[2] J. Lansky, S. Ali, M. Mohammadi, M. K. Majeed, S. H. T. Karim, S. Rashidi, M. Hosseinzadeh and A. M. Rahmani, ‘Deep learning-based intrusion detection systems: A systematic review,’ *IEEE Access*, vol. 9, pp. 101 574–101 599, 2021.

[3] H. Liu and B. Lang, ‘Machine learning and deep learning methods for intrusion detection systems: A survey,’ *applied sciences*, vol. 9, no. 20, p. 4396, 2019.

[4] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou and C. Wang, ‘Machine learning and deep learning methods for cybersecurity,’ *Ieee access*, vol. 6, pp. 35 365–35 381, 2018.

[5] M. Liu, Z. Xue, X. Xu, C. Zhong and J. Chen, ‘Host-based intrusion detection system with system calls: Review and future trends,’ *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[6] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent and Q. Chen, ‘A survey of intrusion detection systems leveraging host data,’ *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–35, 2019.

[7] digitalguardian, *Cybercrime cost u.s. $6.9 billion in 2021*, [Online; accessed 18-May-2022], 2022. [Online]. Available: `https://digitalguardian.com/blog/cybercrime-cost-us-69-billion-2021`.

[8] cybersecurityventures, *Cybercrime to cost the world $10.5 trillion annually by 2025*, [Online; accessed 18-May-2022], 2020. [Online]. Available: `https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/`.

[9] accenture, *Ninth annual cost of cybercrime study*, [Online; accessed 18-May-2022], 2019. [Online]. Available: `https://www.accenture.com/_acnmedia/pdf-96/accenture-2019-cost-of-cybercrime-study-final.pdf`.

[10] Y. Tsuda, J. Nakazato, Y. Takagi, D. Inoue, K. Nakao and K. Terada, 'A lightweight host-based intrusion detection based on process generation patterns,' in *2018 13th Asia Joint Conference on Information Security (AsiaJCIS)*, IEEE, 2018, pp. 102–108.

[11] A. Khraisat, I. Gondal, P. Vamplew and J. Kamruzzaman, 'Survey of intrusion detection systems: Techniques, datasets and challenges,' *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[12] S. Jose, D. Malathi, B. Reddy and D. Jayaseeli, 'A survey on anomaly based host intrusion detection system,' in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1000, 2018, p. 012 049.

[13] T. Mikolov, K. Chen, G. Corrado and J. Dean, 'Efficient estimation of word representations in vector space,' *arXiv preprint arXiv:1301.3781*, 2013.

[14] F. Chiusano, *A brief timeline of nlp*, 2022. [Online]. Available: `https://medium.com/nlplanet/a-brief-timeline-of-nlp-bc45b640f07d`.

[15] J. L. Elman, 'Finding structure in time,' *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[16] A. Louis, *A brief history of natural language processing — part 2*, 2020. [Online]. Available: `https://medium.com/@antoine.louis/a-brief-history-of-natural-language-processing-part-2-f5e575e8e37`.

[17] I. Sutskever, O. Vinyals and Q. V. Le, 'Sequence to sequence learning with neural networks,' *Advances in neural information processing systems*, vol. 27, 2014.

[18] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho and Y. Bengio, 'Attention-based models for speech recognition,' *Advances in neural information processing systems*, vol. 28, 2015.

[19] S. Hochreiter and J. Schmidhuber, 'Long short-term memory,' *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, 'Empirical evaluation of gated recurrent neural networks on sequence modeling,' *arXiv preprint arXiv:1412.3555*, 2014.

[21] S. Kostadinov, *Https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be*, 2017. [Online]. Available: `https://www.guru99.com/system-call-operating-system.html`.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, 'Attention is all you need,' *Advances in neural information processing systems*, vol. 30, 2017.

[23] Deva, *Evolution of natural language processing(nlp)*, 2021. [Online]. Available: `https://medium.com/analytics-vidhya/evolution-of-natural-language-processing-nlp-ac941b6523e9`.

[24] R. Silipo and K. Melcher, *Text encoding: A review*, 2019. [Online]. Available: https://www.kdnuggets.com/2019/11/text-encoding-review.html.

[25] S. Gupta, *Word vector encoding in nlp (make machines understand text)*, 2020. [Online]. Available: https://www.enjoyalgorithms.com/blog/word-vector-encoding-in-nlp.

[26] B. Bose, *Nlp — text encoding: A beginner's guide*, 2020. [Online]. Available: https://medium.com/analytics-vidhya/nlp-text-encoding-a-beginners-guide-fa332d715854.

[27] tensorflow, *Word2vec*, 2023. [Online]. Available: https://www.tensorflow.org/tutorials/text/word2vec.

[28] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding,' *arXiv preprint arXiv:1810.04805*, 2018.

[29] S. Cristina, *The-transformer-model*, 2022. [Online]. Available: https://machinelearningmastery.com/the-transformer-model/.

[30] J. Zhan, Q. Chen, B. Chen, W. Wang, Y. Bai and Y. Gao, *Non-autoregressive translation with dependency-aware decoder*, Mar. 2022.

[31] microsoft, *Event-4688*, [Online; accessed 1-Apr-2023], 2022. [Online]. Available: https://learn.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4688.

[32] ultimatewindowssecurity, *Eventid=4688*, [Online; accessed 1-Apr-2023], 2022. [Online]. Available: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventid=4688.

[33] sysinternals, *Sysmon*, [Online; accessed 1-Apr-2023], 2023. [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon.

[34] ultimatewindowssecurity, *Sysmon event id 1*, [Online; accessed 1-Apr-2023], 2023. [Online]. Available: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventid=90001.

[35] A. Patel, *Detection of anomalous process creation chains using word vectorization, normalization, and an autoencoder*, [Online; accessed 18-May-2022], 2020. [Online]. Available: https://blog.f-secure.com/process-creation-chains/.

[36] J. Brownlee, *Impact of dataset size on deep learning model skill and performance estimates*, Aug. 2020. [Online]. Available: https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/.

[37] Splunk, *Splunk attack range*, [Online; accessed 10-May-2022], 2022. [Online]. Available: https://github.com/splunk/attack_range.

[38] Splunk, *Attack data repository*, [Online; accessed 10-May-2022], 2022. [Online]. Available: `https://github.com/splunk/attack_data/`.

[39] R. Rodriguez, *Securitydatasets*, [Online; accessed 10-May-2022], 2022. [Online]. Available: `https://securitydatasets.com/introduction.html`.

[40] DARPA, *Operationally-transparent-cyber-optc*, [Online; accessed 10-May-2022], 2019. [Online]. Available: `https://ieee-dataport.org/open-access/operationally-transparent-cyber-optc`.

[41] L. Williams, *System call in os (operating system): What is, types and examples*, [Online; accessed 1-May-2022], 2022. [Online]. Available: `https://www.guru99.com/system-call-operating-system.html`.

[42] M. Kerrisk, *Syscalls(2) — linux manual page*, [Online; accessed 1-May-2022], 2021. [Online]. Available: `https://man7.org/linux/man-pages/man2/syscalls.2.html`.

[43] M. Jurczyk, *Windows win32k.sys system call table*, [Online; accessed 1-May-2022], 2022. [Online]. Available: `https://j00ru.vexillium.org/syscalls/win32k/32/`.

[44] G. Creech and J. Hu, 'A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns,' *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 807–819, 2013.

[45] 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*, IEEE, 1996, pp. 120–128.

[46] G.-B. Huang, Q.-Y. Zhu and C.-K. Siew, 'Extreme learning machine: A new learning scheme of feedforward neural networks,' in *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*, Ieee, vol. 2, 2004, pp. 985–990.

[47] M. Anandapriya and B. Lakshmanan, 'Anomaly based host intrusion detection system using semantic based system call patterns,' in *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, IEEE, 2015, pp. 1–4.

[48] A. Aldweesh, A. Derhab and A. Z. Emam, 'Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues,' *Knowledge-Based Systems*, vol. 189, p. 105 124, 2020.

[49] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah and F. Ahmad, 'Network intrusion detection system: A systematic study of machine learning and deep learning approaches,' *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, e4150, 2021.

[50] X. Tong, Z. Wang and H. Yu, 'A research using hybrid rbf/elman neural networks for intrusion detection system secure model,' *Computer physics communications*, vol. 180, no. 10, pp. 1795–1801, 2009.

[51] R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod and M. A. Zissman, 'Evaluating intrusion detection systems without attacking your friends: The 1998 darpa intrusion detection evaluation,' Massachusetts Inst Of Tech Lexington Lincoln Lab, Tech. Rep., 1999.

[52] G. Kim, H. Yi, J. Lee, Y. Paek and S. Yoon, 'Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems,' *arXiv preprint arXiv:1611.01726*, 2016.

[53] A. Chawla, B. Lee, S. Fallon and P. 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.

[54] M. Kwon, K. Jeong and H. Lee, 'Probe: A process behavior-based host intrusion prevention system,' in *International Conference on Information Security Practice and Experience*, Springer, 2008, pp. 203–217.

[55] A. Read, 'Procaid: Process anomaly-based intrusion detection,' Ph.D. dissertation, The George Washington University, 2022.

[56] gensim, *Fasttext*, [Online; accessed 1-Apr-2023], 2023. [Online]. Available: `https://radimrehurek.com/gensim/models/word2vec.html`.

[57] B. Filar and D. French, 'Problemchild: Discovering anomalous patterns based on parent-child process relationships,' *arXiv preprint arXiv:2008.04676*, 2020.

[58] T. Chen and C. Guestrin, 'Xgboost: A scalable tree boosting system,' in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[59] M. Du, F. Li, G. Zheng and V. Srikumar, 'Deeplog: Anomaly detection and diagnosis from system logs through deep learning,' in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[60] H. Guo, S. Yuan and X. Wu, 'Logbert: Log anomaly detection via bert,' in *2021 international joint conference on neural networks (IJCNN)*, IEEE, 2021, pp. 1–8.

[61] C. Almodovar, F. Sabrina, S. Karimi and S. Azad, 'Logfit: Log anomaly detection using fine-tuned language models,' 2023.

[62] ICRAR, *Ijson*, 2023. [Online]. Available: `https://github.com/ICRAR/ijson`.

[63] microsoft, *Processtree*, [Online; accessed 10-Aug-2022], 2023. [Online]. Available: `https://msticpy.readthedocs.io/en/latest/visualization/ProcessTree.html`.

[64] M. M. Anjum, S. Iqbal and B. Hamelin, 'Anubis: A provenance graph-based framework for advanced persistent threat detection,' in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 1684–1693.

[65] H. Heidenreich, *Stemming? lemmatization? what?* [Online; accessed 1-Apr-2023], 2018. [Online]. Available: `https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8`.

[66] A. S. Gillis, *Data splitting*, [Online; accessed 1-Apr-2023], 2022. [Online]. Available: `https://www.techtarget.com/searchenterpriseai/definition/data-splitting`.

[67] scikit-learn, $_split$, 2023. [Online]. Available: `https://github.com/scikit-learn/scikit-learn/blob/364c77e04/sklearn/model_selection/_split.py#L2463`.

[68] tensorflow, *Tensorflow*, 2023. [Online]. Available: `https://www.tensorflow.org/`.

[69] keras, *Keras$_n$lp*, 2023. [Online]. Available: `https://keras.io/keras_nlp/`.

[70] google, *Colab.research.google*, 2023. [Online]. Available: `https://colab.research.google.com/`.

[71] keras, *Api/models/sequential*, 2023. [Online]. Available: `https://keras.io/api/models/sequential/`.

[72] Y. Lee, J. Kim and P. Kang, 'Lanobert: System log anomaly detection based on bert masked language model,' *arXiv preprint arXiv:2111.09564*, 2021.

[73] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang and Z. Luan, 'Hitanomaly: Hierarchical transformers for anomaly detection in system log,' *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[74] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' *arXiv preprint arXiv:1412.6980*, 2014.

[75] tensorflow, *Sparsecategoricalcrossentropy*, 2023. [Online]. Available: `https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy`.

[76] S. Tuli, G. Casale and N. R. Jennings, 'Tranad: Deep transformer networks for anomaly detection in multivariate time series data,' *arXiv preprint arXiv:2201.07284*, 2022.

[77] Z. Chen, D. Chen, X. Zhang, Z. Yuan and X. Cheng, 'Learning graph structures with transformer for multivariate time-series anomaly detection in iot,' *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9179–9189, 2021.

[78] P. Alto, *What-are-fileless-malware-attacks*, [Online; accessed 11-May-2022], 2022. [Online]. Available: `https://www.paloaltonetworks.com/cyberpedia/what-are-fileless-malware-attacks`.

[79] frsecure, *Living-off-the-land-attacks*, [Online; accessed 11-May-2022], 2021. [Online]. Available: `https://frsecure.com/blog/living-off-the-land-attacks/`.

[80] O. Moe, J. Bayne, C. Richard, C. ’. Spehn, Liam and Wietze, *Lolbas-project*, [Online; accessed 11-May-2022], 2022. [Online]. Available: `https://github.com/LOLBAS-Project/LOLBAS/blob/master/README.md`.

[81] S.-B. Cocea, *Bert embeddings: A modern machine-learning approach for detecting malware from command lines*, [Online; accessed 10-Apr-2023], 2022. [Online]. Available: `https://www.crowdstrike.com/blog/bert-embeddings-new-approach-for-command-line-anomaly-detection/`.