Context-Aware Malware Detection Using Topic Modeling

A thesis submitted to the Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

Master of Science

in the Department of Electrical Engineering and Computer Science
of the College of Engineering and Applied Science

by

Wayne Stegner
B.S. Computer Engineering, University of Cincinnati, 2020

September 2021

Committee Chair: Rashmi Jha, Ph.D.

Abstract

Detecting whether or not a software is malicious is a difficult task. The context under which software is executing plays an important role in understanding malicious behavior. Current malware detection strategies have shown high classification accuracy, but they lack contextual considerations. The objective of this thesis is to address the development of a context-aware malware detection system. A definition of context and how it pertains to malware detection is discussed. Based on this definition, two proof-of-concept context-aware models utilizing Latent Dirichlet Allocation are developed to address different aspects of context. These models provide insight into the challenges of including context in malware detection models, and future work to improve the contextual aspects of the models is discussed.

Acknowledgements

There are many people I must thank who have made this journey possible. First and foremost, I would like to thank Dr. Rashmi Jha for helping to connect me with exciting research opportunities and guiding me along the way. I am also very appreciative of her research group for helpful discussions over the research, with a special thank you to Tyler Westland for helping me learn how to use Linux properly.

Thank you to Dr. David Kapp and Dr. Temesguen Kebede from the Air Force Research Laboratory for great research discussions, brainstorming, and helping me iterate through my ideas.

Of course, I would like to thank my friends and family for their support throughout my college career. Finally, big thanks to my cat, Pixel, for keeping my spirits up and keeping me company, especially over the past year during Covid.

This work was supported by the Air Force Research Laboratory under the DAGSI-SOCHE Award No. RY8-UC-20-1.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Cyber security is an ever increasing need, especially as society relies increasingly upon online services. In recent times, several notable cyber attacks have demonstrated the need to further our knowledge of cyber threats. In the SolarWinds security breach, an adversary was able to spread malicious software to an estimated 100 companies and several agencies of the U.S. federal government [1]. It is estimated that this attack could cost the U.S. government alone hundreds of millions of dollars [2]. In a separate cyber attack, a major U.S. oil pipeline was compromised, leading to a temporary shutdown of the pipeline [3]. With such severe consequences, it is vital to increase our ability to detect and mitigate cyber threats.

Detecting whether or not a software is malicious is a difficult task. The context under which software is executing plays an important role in understanding malicious behavior. A given sequence of opcodes or system calls is not necessarily malicious by nature because malice is relative to the desired outcome of running the software. If a specific opcode or system call was malicious, it would simply not be built into the system in the first place.

Take, for example, an autonomous drone with an onboard camera; turning the camera on and off is not inherently malicious, otherwise the camera would be hard-wired on. There

can be some times when turning the camera off is a desired, benign action, such as saving power on the drone or preserving privacy in sensitive locations. That being said, a malicious software can cripple the functionality of a drone by switching off the camera at undesired times. The determining factor on whether or not turning the camera off is malicious is the context in which the software chose to turn it off.

1.2 Outline

Chapter 2 outlines the prerequisite background topics, including malware analysis and the machine learning techniques used in this work. Next, a brief literature review of relevant efforts in malware detection and context-aware software analysis is given.

Chapter 3 details our methodology in developing a context-aware malware detection model. We discuss the central point of how we should define context, as well as the rationale for several proof-of-concept context-aware model iterations.

Chapter 4 presents the results of the models explained in Chapter 3, including a discussion of how well context is addressed.

Finally, Chapter 5 discusses the main takeaways from our work, including a summary of areas of improvement for future work.

Chapter 2

Background

2.1 Prerequisite Topics

First, we will explain some prerequisite knowledge which will be necessary for full understanding of this research.

2.1.1 Overview of Malware Analysis

Malware analysis exists in two categories: static analysis and dynamic analysis [4]. Static analysis refers to any technique in which a file is examined without running the code, whereas dynamic analysis involves actually running the code and observing its behavior.

Static analysis of binary files is difficult because they are designed to be read by computers and not humans. To solve this issue, it is common to use a disassembly tool, such as IDA Pro [5] or Ghidra [6]. Disassembly is the process of taking an executable file and extracting the assembly code and any annotations which are present. The disassembly can then be analyzed for patterns, such as specific sequences of opcodes (known as signatures), or further feature extraction can be done. Other interesting features in static analysis include looking for system calls, hard-coded data such as strings, and control-flow features. Static analysis is generally safe because no malicious software must be run. However, it may be difficult to

statically examine code if it has been obfuscated.

Dynamic analysis involves running software samples and observing the resulting behavior. This analysis is typically done in a sandboxed environment, such as a virtual machine, to contain any malicious behaviors. In dynamic analysis, interesting features to collect include the dynamic assembly sequence, system call sequence, and system changes such as file modifications or registry edits. Dynamic analysis is advantageous because even if code is obfuscated, malicious actions can be observed which may not have been apparent through static analysis. However, some malicious code can detect that it is in a sandbox and mask its malicious behavior to avoid detection.

2.1.2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a generative statistical model used in natural language processing (NLP) to model data, such as a corpus of documents [7]. LDA is used to learn topics from the corpus which can be used to describe the documents, such that each document is a mixture of those topics, and each topic is a distribution over the vocabulary of the corpus. In other words, LDA aims to learn which words in the vocabulary are related to each other and group related words into topics, and then it assigns each document a weighting of each topic. Please note that in the literature, the number of topics is often called k. This naming interferes with k from k-Nearest Neighbors (k-NN). To resolve this issue, we will refer to the number of topics simply as "number of topics" and the k-NN parameter as k.

Before training an LDA model, the corpus must be translated into a bag-of-words (BoW) model. A BoW model counts the frequency of words in a document. For example, given a document "cat dog mouse cat cat dog", the BoW representation is $\{\text{“}cat\text{”} : 3, \text{“}dog\text{”} : 2, \text{“}mouse\text{”} : 1\}$. BoW models are common in NLP and machine learning because they turn documents into vectors of equal length which can be easily compared. While BoW is sometimes used directly for classification, we will be using it as preprocessing for LDA.

An extension of LDA is hierarchical LDA (hLDA) [8]. Similarly to LDA, the goal of

hLDA is to learn latent topics from a corpus of documents. However, in hLDA, the topics are organized in hierarchically. At the top of the hierarchy is the most general topic, then as the topics go deeper in the hierarchy, they become more specific. The hierarchical organization is beneficial because topics are naturally arranged hierarchically and can be broken into more specific topics. However, LDA is much simpler to use because the resultant data is simply a vector.

To evaluate the quality of an LDA model, there are two primary methods: intrinsic and extrinsic evaluation [9]. Intrinsic evaluation of a set of topics generally uses measures such as perplexity or topic coherence. These measures are advantageous because they are unsupervised measures, and therefore do not depend on labeled data. On the other hand, extrinsic evaluations based on classification tasks can be more convenient if the end goal is to use the LDA model for classification.

2.1.3 k-Nearest Neighbors

k-Nearest Neighbors (k-NN) is a machine learning algorithm which can be used for classification [10]. The algorithm is simple to use, as typically the only parameter is k. In k-NN, first a labeled training set is presented to the model, which the model stores for later comparison. The model is then used to assign classes to new data points. The class of the new data point is determined by a vote among the nearest k neighbors (typically determined by Euclidean distance, but any distance can work). A common modification to k-NN is weighting the vote of each neighbor by the inverse of its distance, making closer neighbors are more influential in the vote. The k-NN classifier is a useful tool because it requires no training iterations and has few parameters to tune (typically just k).

2.2 Malware Classification with LDA

LDA has been applied to the field of malware classification [11]–[13].

Sundarkumar et al. [11] collected application programming interface (API) calls from a set of programs and then trained an LDA model on those API calls. The LDA features were then classified using various machine learning classifiers with a maximum accuracy of 98.61%.

Greer [12] utilized LDA and hLDA to model both static and dynamic opcode sequences of sorting and searching programs. The LDA and hLDA results were both shown to differentiate between sorting and searching program classes. The analysis of the topic similarity was done by hand and not computationally, though these results did show promise in LDA to distinguish between program behavior based on low-level features.

Djaneye-Bounjou et al. [13] extended Greer's work by applying LDA to static opcode sequences to classify malware. They classified the Microsoft Malware Classification Challenge (BIG 2015) malware dataset [14] using a k-NN classifier on the topic distributions with an accuracy of 97.2%.

2.3 Context in Software Analysis

Fernandez et al. [15] proposed a model for context-based access control policy focusing on mobile devices. The idea was to limit access to certain sensitive resources based on the context of the system. Context is comprised of elements, such as the physical context (i.e., location data), and the logical context, which includes a device profile, a user profile, and a task profile. The premise of using physical specifications to help define the context was of interest to us, but the overall model was fairly user-centric, making it incompatible with autonomous vehicles (one of the motivations for this research).

Shebaro et al. [16] also worked on context-based access control policies for mobile devices, but this study was more application focused. Their access control policy was similar to that of [15], but the context was mostly based on location. The study features an experiment where certain features of the mobile device were restricted at certain locations. This type of access control can work for situations with a small, pre-defined setting, such as an office

building. However, it requires manual definition of acceptable context zones, which is less desirable for scalability.

Shrestha et al. [17] developed a different approach to implementing context in a mobile platform. Instead of basing context on the location, they define their context based on physical gestures done by the user. These gestures are intended to ensure that the user was the one who triggered certain sensitive actions to prevent automated attacks by malware. Because the context is defined solely based on user interaction, it does not satisfy our needs.

Narayanan et al. [18] also aimed their study at mobile devices, focusing on context-aware malware detection for Android phones. They utilize program representation graphs, such as system call graphs, to represent applications running on the devices. By examining the entry point to sensitive functions, such as accessing position data, the context is defined according to whether the user was aware or unaware of the action. Again, while this definition of context works well for mobile devices, it does not generalize to autonomous systems.

The existing contextual methods discussed in this section have provided some insight into how to define context. However, they all target mobile devices, such as smart phones, and typically require on some user interaction to define the context. There is still a need to pursue the topic of context-aware malware detection.

Chapter 3

Methodology

3.1 Threat Model

A vital part of addressing any cyber security threat is defining the threat model. Without a specific threat model, the problem space is difficult to define. In our threat scenario, a malicious actor causes some software from a 3rd party vendor to perform some malicious action. We require a context-aware malware analysis model to detect such malicious software before it is run on an autonomous drone. A diagram of our threat model is shown in Figure 3.1.

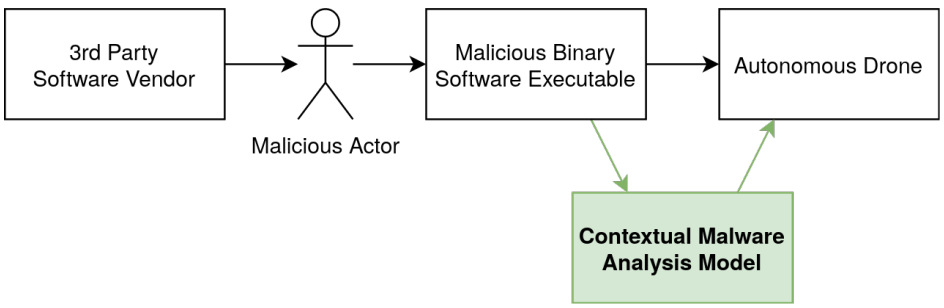


Figure 3.1: Threat model diagram.

3.2 Context Definition

One of the biggest challenges in this work has been defining context. In Chapter 2.3, existing literature regarding context in software analysis is discussed. However, these works do not define context in a way which aligns with the goals of this work. In order to properly address context, we want to address the following questions:

1. What is the physical context of the system (e.g., location and altitude)?
2. How does the actual behavior compare to the expected behavior?
3. Why is the software making certain decisions?

The physical context is an important component of context, especially for mobile systems, such as autonomous drones. For instance, if a drone performs a specific action, it may be difficult to tell if that action is benign or malicious without considering factors such as its position, altitude, or external weather conditions to name a few. Recall the autonomous drone example from Chapter 1.1; turning the camera off in some areas may be beneficial for preserving sensitive data, while turning the camera off in other areas can be malicious if the purpose is to obstruct the gathering of intelligence. The simple act of turning the camera off is not inherently malicious, but the physical conditions which caused the camera to turn off can indicate whether the act was malicious or benign.

Understanding the expected behavior of a program is foundational for developing context because it defines what actions are expected. For example, it makes sense for a text editor to read and write files on the system because that is part of the expected behavior, and modifying files fits the context. It would probably be unexpected for a text editor to start recording from a webcam because that behavior does not fit the context of a text editor. In contrast, a program designed for video calls would be expected to access the web cam because that fits the context of a video call software. The act of accessing the webcam cannot be labeled as malicious or benign without knowledge of what the program should be doing.

Perhaps the most fundamental question in determining the context of a program is why it makes certain decisions. This question is much broader and more difficult to address precisely than the previous two. When a human expert examines a piece of software to determine whether or not it is malware, they are looking not only at what behaviors are present, but looking at what conditions will cause such behaviors to occur. The human expert can then rely on domain knowledge and previous experience to determine if the actions are malicious or benign.

3.3 Cross Validation

Due to the random nature of fitting LDA models, we observed high variation in the accuracy of classifiers which use the LDA features as their primary features. To tune the parameters of the models, it is important to know whether differences in classification accuracy are due to randomness in the fitting process or if the parameters are actually better. To get a better evaluation of the classifiers, we applied k-fold cross validation [10]. The k-fold cross validation process is described by Algorithm 1.

Algorithm 1: k-Fold Cross Validation

Data: Dataset D

Randomly shuffle D;

Split D evenly into $fold_0$, $fold_1$, $\ldots fold_{k-1}$;

for $i \leftarrow 0$ **to** $k-1$ **do**

  Initialize a new model;

  Fit model on $D \setminus fold_i$;

  Evaluate model accuracy on $fold_i$;

end

Compute mean and standard deviation accuracy;

An alternative validation strategy is holdout validation. In this strategy, only a single model is fitted with set testing and training partitions, which takes less time than cycling

through all of the folds. However, k-fold cross validation is advantageous over holdout validation in our case because multiple models are trained, allowing us to observe the mean classification accuracy and standard deviation. By testing the models on multiple distinct testing sets, we are able to make stronger claims about the performance of our models.

3.4 Datasets

3.4.1 Das Malwerk Dataset

For evaluating our first context-aware model, we wanted a simple dataset for preliminary testing. We also had an initial requirement to use live files to collect dynamic features. This requirement was later dropped from the model, but the dataset was chosen under constraint of needing live files. To meet these requirements, we collected malicious files from Das Malwerk [19] and benign files from a default installation of Windows 7. This dataset was chosen because of its small size and the fact that it contains live executable files, meaning it met our requirements. This dataset contains 576 malicious files and 646 benign files.

To get the dataset into a usable state, there were several preprocessing steps. First, the static assembly commands were obtained using `objdump` [20]. Given an input file `IN_FILE` and a target output file `OUT_FILE`, `objdump` can generate the disassembly using the `-d` flag as follows (on Linux systems):

```
objdump -d ${IN_FILE} > ${OUT_FILE}
```

Next, the static assembly commands must be filtered down to documents consisting of sequences of opcodes. To achieve this, several command line utilities are piped together as a series of string stream filters. The first is `cat` [21], which concatenates files and dumps the output to standard output of the terminal. Next, `sed` [22] filters the incoming stream based on regular expressions and puts the filtered stream in standard output. Lastly, the `cut` [23] utility is used to isolate the column of the remaining output which contains the opcodes. The process of taking an input file `IN_FILE` (the output from `objdump`), filtering

out the opcode sequence, and outputting it to the output file `OUT_FILE` is as follows (again, on Linux systems):

```
cat ${IN_FILE} | sed '/[^\t]*\t[^\t]*\t/!d' | cut -f 3 | sed 's/ .*$//' \
  | sed '/\(bad\)/d' > ${OUT_FILE}
```

The resulting sequences of opcodes are used as documents for the models discussed in this study.

3.4.2 Microsoft BIG 2015 Dataset

After preliminary experimentation, we found several drawbacks with the Das Malwerk dataset we wanted to fix for further testing. The biggest drawback is that the Das Malwerk dataset consists of only two classes, and each class contained a very broad mix of types of software. The benign class had all of the executable files found on a default Windows 7 installation, which includes web browsers, text editors, media players, and many others. Furthermore, the malicious class includes a mixture of unsorted malware programs, and sorting them into more specific groups would have been a manual process prone to error. With such drastically different functionality, it does not make sense to assign a single acceptable context to each class. The scenario was just too simple.

To improve our methodology, we needed a better dataset. The new dataset must have multiple classes which are separated by specific program type, and not just into malicious and benign classes. We no longer had the constraint of needing live files because the dynamic analysis components had been set aside, which gave us more dataset options. To meet these requirements, the second dataset we used was the Microsoft Malware Classification Challenge (BIG 2015) malware dataset [14]. This dataset has become a benchmark for malware detection algorithms, and it has been widely studied. The details of the class distribution of the BIG 2015 dataset are shown in Table 3.1.

One of the interesting challenges about this dataset is that the number of samples varies by orders of magnitude between some classes. We do not explicitly address this issue and

Table 3.1: Details of the BIG 2015 dataset class distributions.

Class Name	Class ID	Number of Samples
Ramnit [24]	1	1541
Lollipop [25]	2	2478
Kelihos_ver3 [26]	3	2942
Vundo [27]	4	475
Simda [28]	5	42
Tracur [29]	6	751
Kelihos_ver1 [26]	7	398
Obfuscator.ACY [30]	8	1228
Gatak [31]	9	1013

leave it to future work, though several existing studies have addressed the issue [32]–[34].

Preprocessing for the BIG 2015 dataset was much simpler than for the Das Malwerk dataset because BIG 2015 was distributed in disassembly form. Therefore, it only required filtering it down to a sequence of opcodes. The opcodes were filtered out using regular expressions similarly to the Das Malwerk dataset.

3.5 Context-Free Model

Before developing the context-aware models, we demonstrated that LDA modeling is able to extract useful features from the dataset. To do this, we tested a context-free malware classification model using LDA features in a k-NN classifier. The context-free model was also used to explore parameters of the LDA and k-NN models, namely the number of topics in LDA and k in k-NN. Assuming we have each dataset such that each document is a sequence of opcodes (done via preprocessing, described in Chapter 3.4), the process for creating the context-free model is as follows:

1. Transform all documents into BoW documents.

2. Fit LDA model on the training partition.

3. Transform all BoW documents into topic distributions.

4. Fit k-NN classifier on the topic distributions of the training partition.

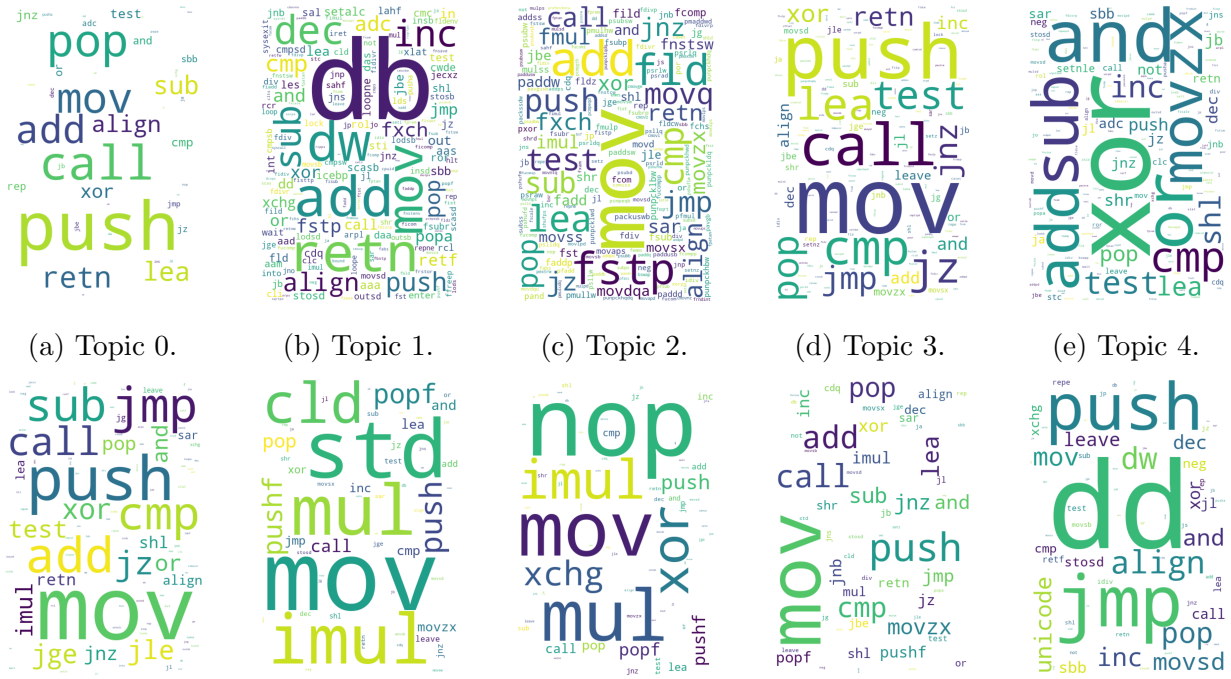
5. Evaluate k-NN classifier on the topic distributions of the test partition.

To thoroughly evaluate this model, we use 5-fold cross validation as described in Chapter 3.3. We limited the number of folds to five for model training time. In the LDA models, we explored the effect of the number of topics, testing values from 5 to 95 at multiples of 5. For k-NN, we tested values for k taking the odd numbers from 1 to 49.

To help understand the features generated by LDA, a 10 topic LDA model was fit on the BIG 2015 dataset, and the learned topics are visualized as word clouds in Figure 3.2. Each topic is a probability distribution over the vocabulary of the corpus, and the probability of each word is shown by the size of the word in the word cloud. The larger words should be interpreted as more significant to the topic, while the smaller words are less significant. Note how some topics, such as Figure 3.2a, seem to have only a few words. These topics still span the entire vocabulary, but the probability distribution is heavily skewed toward a few of the words, and the rest of the words are so insignificant to the topic that they are not large enough to be visible in the word cloud.

3.6 Context Bit Model

In developing our first context-aware model, we started with a high-level idea of how the data and features should be used. Our original idea was to take several different sets of features from different data collection strategies and combine them with contextual information in an ensemble classifier. First, the static disassembly of a file would be collected and filtered to a sequence of static opcodes and fed through an LDA model. Next, a dynamic assembly trace would be collected from each file and similarly filtered to a sequence of dynamic opcodes and fed through an LDA model. The final feature is a set of high level behavioral features. The contextual information in this model is based on the physical context of the system, which is derived from environmental information from sensor data. This data would then



(a) Topic 0. (b) Topic 1. (c) Topic 2. (d) Topic 3. (e) Topic 4.

(f) Topic 5. (g) Topic 6. (h) Topic 7. (i) Topic 8. (j) Topic 9.

Figure 3.2: Word cloud representations of each topic for a 10 topic LDA model fit on the Microsoft BIG 2015 dataset. The size of each opcode indicates its weighting in the topic.

be interpreted by an environmental context model and fed into the ensemble classifier. A diagram of this context-aware model is shown in Figure 3.3.

While the ensemble system could provide more diverse features for classification, there were limitations. The biggest limitation was the speed at which the dynamic features could be collected. Because the dynamic features were not necessary to explore context in this model, the ensemble classifier was removed, as with the behavioral analysis and the dynamic assembly trace. The simplified context-aware model is shown in Figure 3.4.

Generating the sensor data and developing an environmental contextual model would be difficult to generalize for our proof-of-concept model. For this reason, we black-boxed the sensor data and context model and simplified it to be a single bit to represent "good context" (0) and "bad context" (1). A context bit was randomly generated for each data point with a 50% chance of being either a 1 or a 0. The context bit was then appended to the LDA features to form the feature vector. For example, if the LDA model has 15 topics, the feature

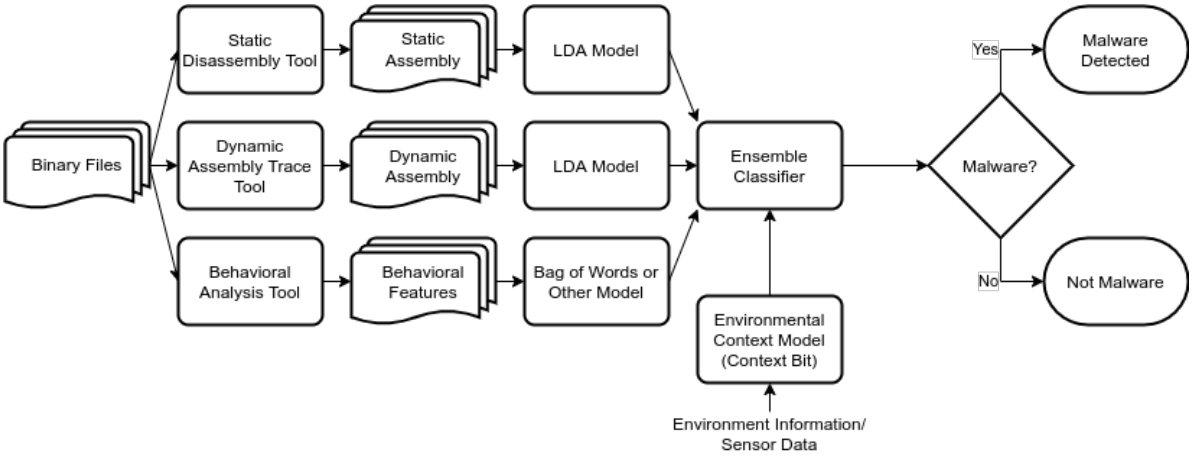


Figure 3.3: Ensemble context-aware model with context represented as a context bit.

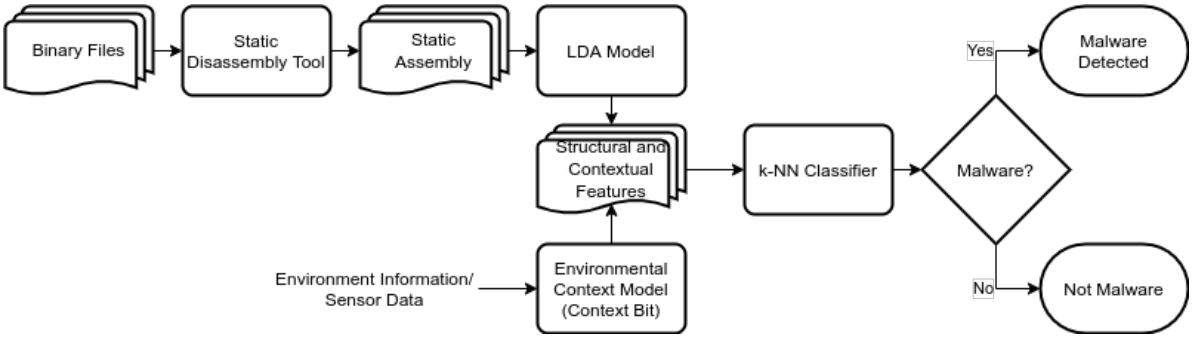


Figure 3.4: Simplified context-aware model with context represented as a context bit.

vector would be 16-dimensional; the first 15 elements are be the 15 topic weights, and the final element is the context bit.

The data was relabeled after applying the context bit. For this process, we must take a different perspective on what the labels mean. In the context-free model, each file was assigned either malicious or benign with no contextual information. However, when we consider the context of the system, whether or not a file is malicious is dependent on the context the file is running in. Instead, the files should be thought of as whether or not they are operating in their expected context. If a benign file is operating in a good context (context bit is 0), that should be labeled as operating within proper context. Likewise, if a malicious file is operating in a bad context (context bit is 1), it should also be labeled as operating within proper context. A file is labeled as violating its expected context if the nature of the file does not match the context, meaning a benign file in a bad context or

a malicious file in a good context. Files operating within proper context were labeled 0, and files operating outside of proper context were labeled 1. These new features were then classified using a k-NN classifier. Because this model relies on only having two classes, we used the Das Malwerk dataset described in Chapter 3.4.1.

3.7 Expected Behavior Model

While the context-aware model in Chapter 3.6 addresses physical context, we also wanted to attempt to address other areas of context. For this model, we compared the expected behavior of a file with its actual behavior. Expected behavior of a program can be difficult to quantify. For this case, we are considering the class label to indicate which type of software we are expecting. To ensure that all of the files within a specific class are actually similar in functionality, we used the BIG 2015 dataset to test this model.

Like the model in Chapter 3.6, this model processes the sequence of static opcodes through an LDA model to get the topic distributions. The difference here is in how the context data is acquired. Because the class label indicates the expected behavior of the program, if we had a program which was mislabeled, that would represent a context violation because the label does not match the actual software. Referring back to the threat model, if the 3rd party software vendor gives us a piece of software and says it's a video call program, we would expect it to operate within similar context to other video call programs we have seen in the past. If the vendor gives us a program and says it is a video call program, but it turns out to be a file sharing program, we should detect that the program is not matching the behavior we would expect for a video call program and realize that it is not operating within the expected context.

To simulate receiving a software which is not the type we are expecting, we took the BIG 2015 dataset and changed half of the class types to an incorrect type. For programs which had the original type, we labeled those as operating within proper context. If the type was

changed to an incorrect type, we labeled those programs as violating their expected context.

The feature vectors for this dataset were formed similarly to the features of the context bit model. Instead of adding the LDA features and the context bit together, we added the LDA features and the class type (with half of them changed to the wrong type). Because the magnitude of the class type is arbitrary, we represented the class identifier as a one-hot encoded vector. The one-hot encoded vector ensures that all class identifiers are represented with the same magnitude. If the LDA model has 15 topics, the feature vectors would be 24-dimensional; the first 15 elements are the LDA topic weights, and the last 9 elements are from the one-hot encoded vector for the class identifier.

Chapter 4

Results

4.1 Context-Free Model

The context-free model was trained on both the Das Malwerk and the BIG 2015 datasets. The purpose of the context-free model was to test that the LDA models were producing useful features for comparing the different classes.

The context-free classifier for the Das Malwerk dataset for various numbers of LDA topics is shown in Figure 4.1. While we tested from 5 topics to 95 topics in multiples of 5, we only plotted a handful of different numbers of topics against each other for visual clarity. We have also excluded the error bars for the same reason.

In Figure 4.1, there is a clear trend of maximal accuracy at $k = 1$. To compare all of the LDA models at their peak performance, we plotted their accuracies with k fixed at 1, shown in Figure 4.2. The 5 topic LDA model is excluded because its performance was significantly lower than the others, making the differences between the other models harder to notice. The maximal performance occurs with 85 LDA topics at $k = 1$, with a classification accuracy of 95.99%.

The context-free model for the BIG 2015 dataset for various numbers of LDA topics is shown in Figure 4.3. However, this time the trend appears where the maximal value for

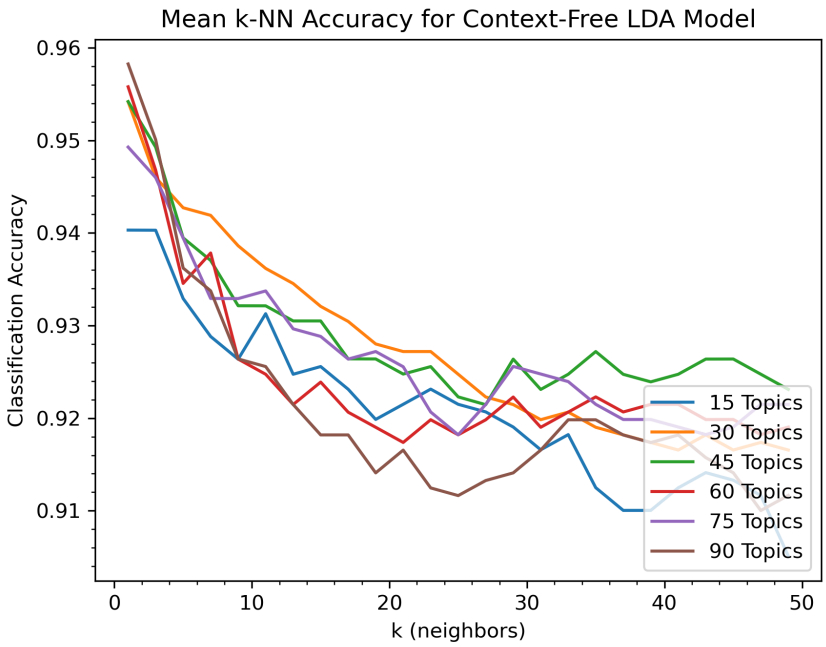


Figure 4.1: Context-free k-NN accuracy vs k for Das Malwerk. For visual clarity, not all numbers of topics are shown and error bars are not included.

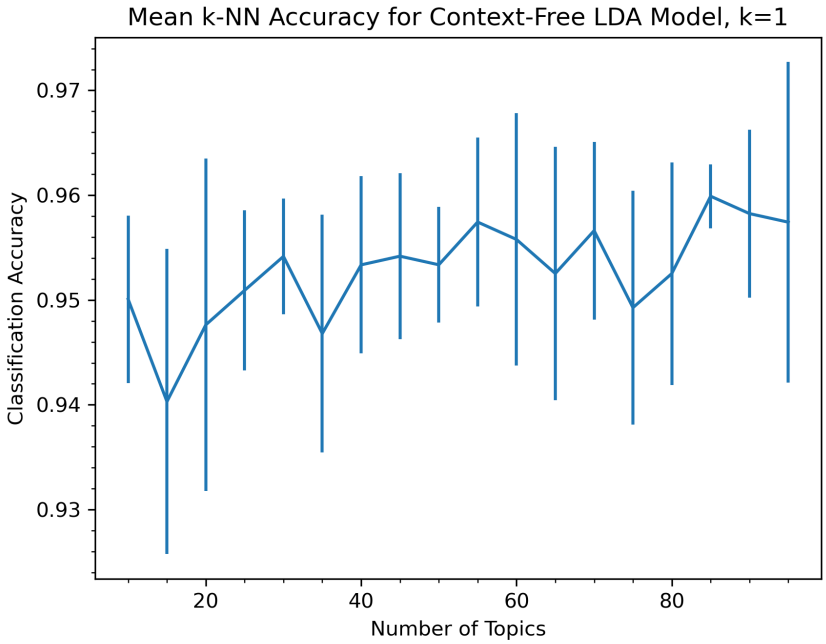


Figure 4.2: Context-free model k-NN accuracy vs number of topics for Das Malwerk fixed at $k = 1$. For visual clarity, the 5 topic model is excluded.

each LDA model occurs at $k = 5$, with the exception of the 15 topic model where $k = 1$ is a bit higher. We plotted all of the LDA models with k fixed at 5 to compare their best performances, shown in Figure 4.4. In this case, the best performing model is with 90 LDA topics at $k = 5$, with a classification accuracy of 97.03%.

4.2 Context Bit Model

The context bit model (trained on Das Malwerk) for various numbers of LDA topics is shown in Figure 4.5. As with the context-free model for Das Malwerk, the context bit model exhibits a trend where the maximum classification performance occurs at $k = 1$. To better compare the performance of all of the LDA models (except for 5 topics), we plotted their performance with k fixed at 1, shown in Figure 4.6. The maximal performance occurs with 45 LDA topics at $k = 1$, with a classification accuracy of 94.92%.

4.3 Expected Behavior Model

The expected behavior context model (trained on BIG 2015) for various numbers of LDA topics is shown in Figure 4.7. As with the context-free model for BIG 2015, all of the models have their best accuracies at $k = 5$. A plot showing all of the LDA models with k fixed at 5 is shown in Figure 4.8. The best performance occurs with 75 topics at $k = 5$, with a classification accuracy of 97.72%.

4.4 Discussion

The context-free model shows good performance on both the Das Malwerk and the BIG 2015 datasets, showing performance on par with other works utilizing LDA for malware detection. Therefore, we have shown that LDA features are useful for distinguishing the different classes from each other. The error bars are large and overlapping for many of the LDA models due to

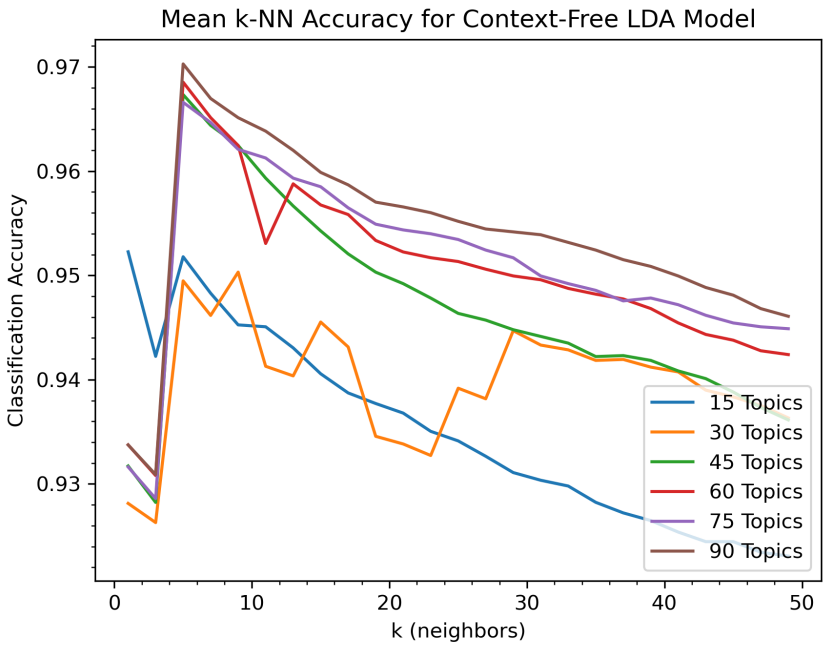


Figure 4.3: Context-free model k-NN accuracy vs k for BIG 2015. For visual clarity, not all numbers of topics are shown and error bars are not included.

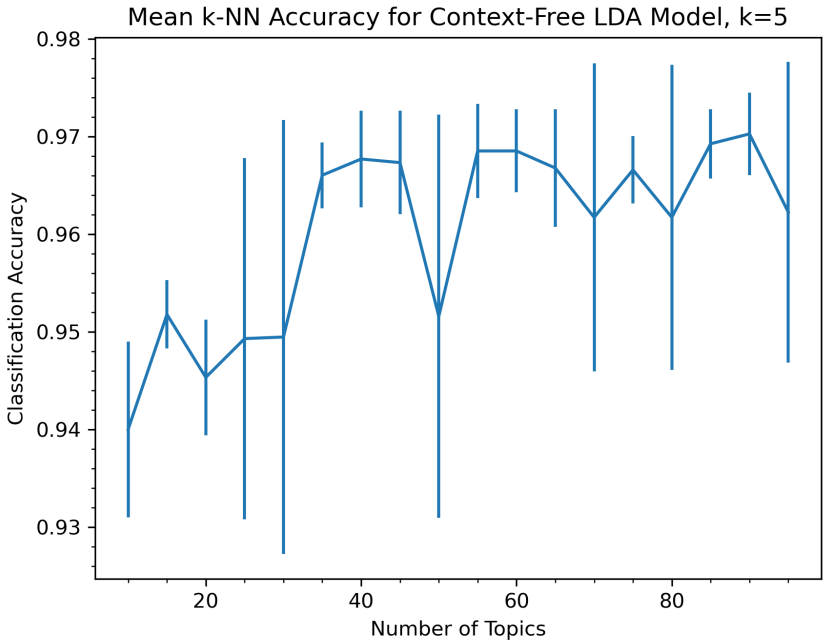


Figure 4.4: Context-free model k-NN accuracy vs number of topics for BIG 2015 fixed at $k = 5$. For visual clarity, the 5 topic model is excluded.

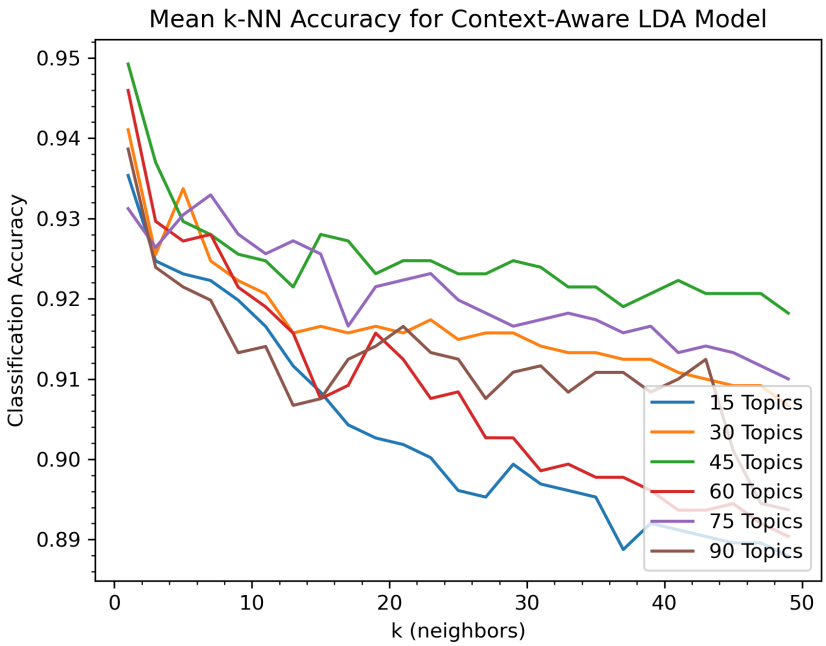


Figure 4.5: Context model bit k-NN accuracy vs k. For visual clarity, not all numbers of topics are shown and error bars are not included.

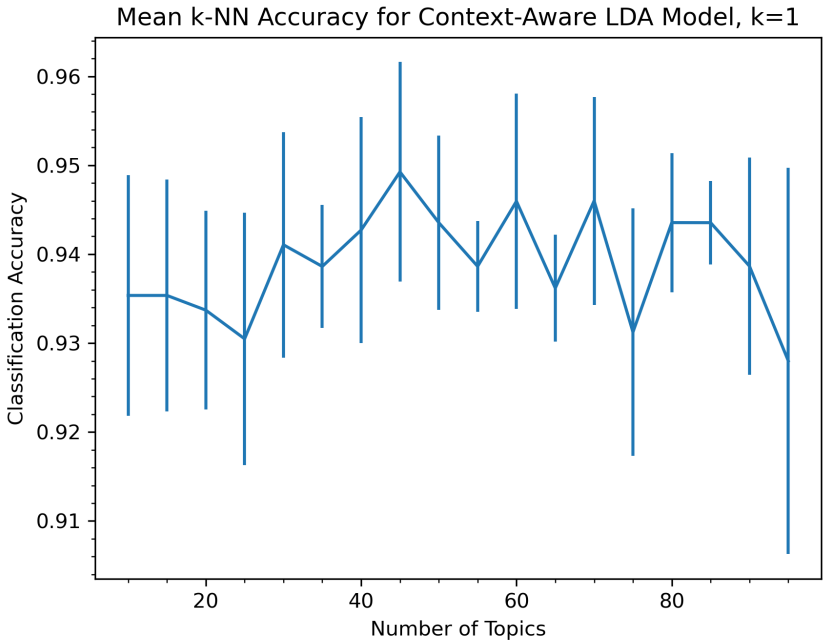


Figure 4.6: Context bit model k-NN accuracy vs number of topics fixed at $k = 1$. For visual clarity, the 5 topic model is excluded.

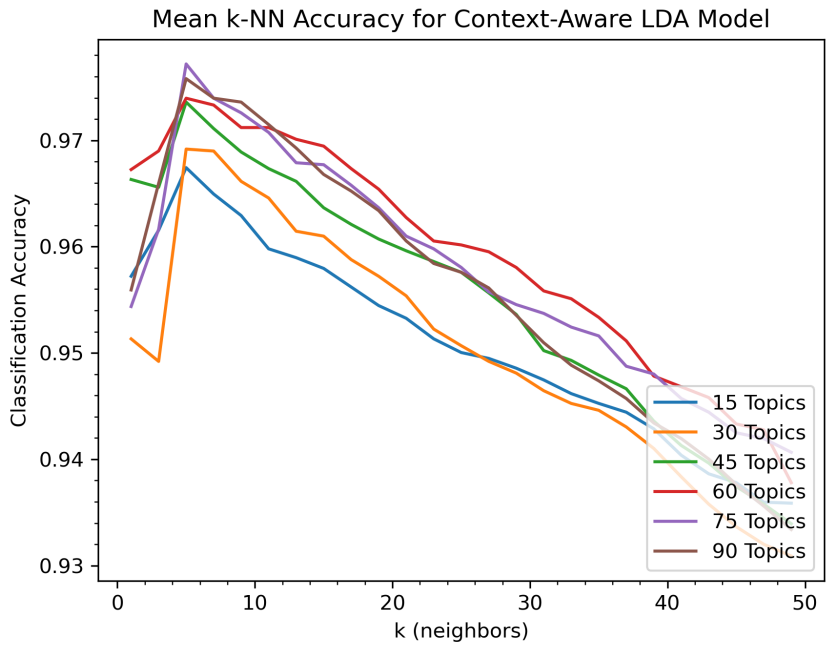


Figure 4.7: Expected behavior model k-NN accuracy vs k. For visual clarity, not all numbers of topics are shown and error bars are not included.

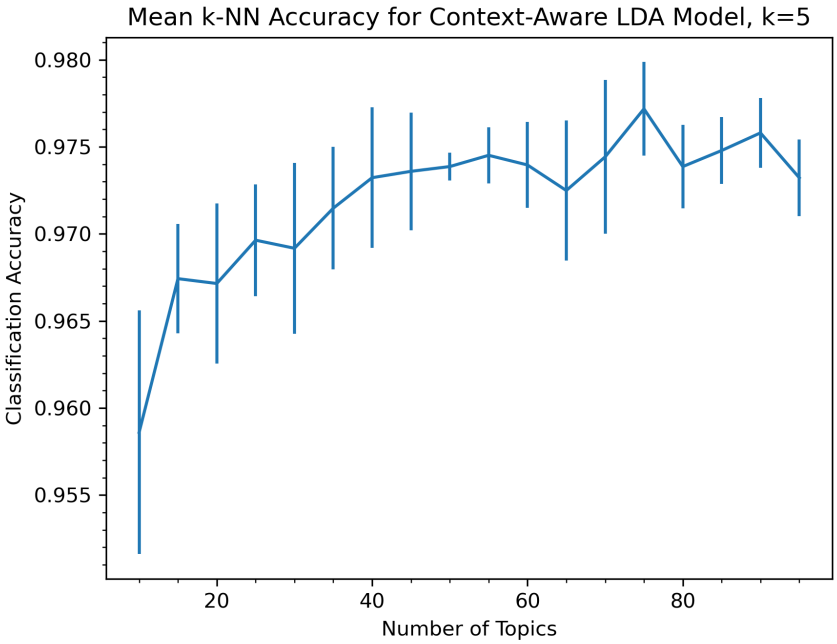


Figure 4.8: Expected behavior model k-NN accuracy vs number of topics fixed at $k = 5$. For visual clarity, the 5 topic model is excluded.

the randomness involved in fitting the LDA models. While the performance generally stayed above 90%, such large error bars make it difficult to be confident that the best parameters we found are actually the best and not just a fluke. Regardless, the performance is good enough to justify the use of LDA features for the context-aware models.

One thing that LDA lacks in this application is transparency for explanations. When LDA is used for actual natural language documents, the topics can be interpreted intuitively. For example, if a topic has prominent words such as "resistor", "capacitor", and "voltage", a human viewer could interpret this category to be "electronics". However, when the topics are made up of opcodes, it is difficult to intuitively understand what exactly each category means. Due to this lack of understanding of the topics, it can be challenging to understand explanations for the given decisions. Furthermore, the LDA and BoW models are irreversible operations, so the explanations cannot easily propagate past those features back to the source code.

An interesting pattern was that the Das Malwerk dataset always had the best classification accuracy with $k = 1$, while the BIG 2015 dataset always had its best classification accuracy with $k = 5$. This trend was present in virtually all numbers of topics for both the context-free and context-aware models.

The context bit model shows good performance for the task we designed. The performance was a bit lower than the context-free system, which was expected because this implementation is essentially equivalent to injecting noise into the system. The issue is that the task was oversimplified. For this model, the physical context is reduced to a single binary value. In reality, the physical context will be a much more complicated issue. The takeaway from this model is that the focus should be on the portion which we black-boxed for this study, which is the model which interprets the physical data into a context. Additionally, the physical context alone does not form a complete context, and it requires inclusion of other aspects of context.

The expected behavior context model performs its intended task very well. One area

which might impact the performance is the way the labels are generated. If the class identifier is changed, the program is marked as a context violation, regardless of what class it was switched to. However, if two classes are similar to each other, it is possible that their LDA topic distributions are expected to be similar. This issue was considered, but the performance is high enough to demonstrate the desired concept.

Chapter 5

Conclusion

In this research, we have explored the definition of context in malware detection and two proof-of-concept context-aware models. These models use LDA to extract structural features from the code, combining those features with contextual features to detect context violations. The first model utilizes the system's physical context, while the second compares the expected and actual behavior of the programs. These models are successfully able to detect their respective context violations.

The primary takeaway from this work is that it is difficult to define precisely what context is in malware analysis. In fact, a significant endeavor on this research was just determining how to frame context for software. We framed the idea of context as a series of questions which we should answer, but they do not translate directly into a computational model. The proof-of-concept models we presented showed strong performance at their designated tasks, but they do not make up a complete picture of context, leaving room for a multitude of future research directions.

5.1 Future Work

5.1.1 Practical Example

The example scenarios presented in this work were at an abstract level and do not extend well to practical examples. To continue working on context-aware models, a more concrete, practical example must be explored. One of the biggest barriers in testing a practical example is the nature of publicly available datasets. These datasets are not designed with context in mind, so they do not include the necessary information required to establish context. Ideally, each file in the dataset should be labeled with contextual information, such as the intended behavior of the file. Such a dataset would prove invaluable to the development of future context-aware models.

5.1.2 Dynamic Analysis

In this work, we only utilized static analysis techniques. For future work, it would be advantageous to include dynamic analysis features and use them in conjunction with the static analysis. Combining the two types of features could improve the robustness of the models. Our initial model included dynamic assembly trace features and high-level behavioral features, but these features were not included in the final model due to time limitations in collecting the data. While the static LDA features were shown to be effective at classifying the datasets, taking both dynamic assembly sequences and high-level behavioral features would contribute a more complete picture of the programs. Additionally, high-level behavioral features are more interpretable than the LDA topics because they convey tangible actions as opposed to the structure of the underlying code. The static opcode analysis could be bypassed if the attacker has knowledge of the detection strategy, but high-level features must occur in some form to actually perform a given action.

Another limitation of the static analysis is that it does not convey any temporal relationships between events. In reality, actions will happen in a time sequence over the duration

of program execution. Having time-series data of high-level behaviors with correlating environmental information will help to develop a more complex contextual model.

5.1.3 Biological Inspiration

A huge area to improve on this work is looking at biological inspiration for the formation of context. So far, we have relied on statistical methods to implement the context-aware models. While we were able to accomplish small pieces of the context problem this way, it is becoming apparent that statistical methods alone will not be able to completely solve the problem of context. Instead, it would be interesting to look for some biological inspiration into how the brain creates context. When a reverse engineer looks at a program, they are able to use context they have developed from past experience as well as knowledge of the overall system to determine whether an action is malicious or benign. The end-goal would be to develop a model which can mimic the way a human expert can assess the context of a program. To aid with the biological inspiration, neurologically inspired computational models should be investigated.

5.1.4 Physical Context Model

One of the most obvious areas for improvement is in the physical context model, as this work leaves that system as a black box. Testing should be done to try different kinds of models to see how to most effectively utilize the data. In our system, we had the physical context model separated from the rest of the model, but it may be a better idea to use a model which can directly learn which actions are allowed in which context. Exploring neural networks or another model which actually "learns" would be an ideal starting point.

5.1.5 Combining Context Models

At this point, the physical context and expected behavior context is evaluated in two separate models. However, it is necessary to consider both of these elements to determine if a given program is well-behaved given the context. There are several options, each with varying levels of implementation difficulty and robustness. The simplest solution would be to keep the models separate and just say that a program must pass either individual context check. However, this solution may not be robust enough, as it considers the context separately and not wholistically.

5.1.6 Parameter Tuning

In this work, some basic parameter exploration was done on the LDA and k-NN algorithms. However, the randomness of LDA fitting caused large error bars for the k-NN classification accuracy which have significant overlap. It is possible that further refinement of the LDA parameters aside from just the number of topics is required to get more consistent results over this type of data. Future work should be done to try to refine the parameter tuning.

Bibliography

[1] D. Temple-Raston, *A 'worst nightmare' cyberattack: The untold story of the SolarWinds hack*, en. [Online]. Available: https://web.archive.org/web/20210706101341/https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack (visited on 07/07/2021).

[2] C. Nolan and A. Fixler, "The economic costs of cyber risk," en, Foundation for Defense of Democracies (FDD), Tech. Rep.

[3] M. Peñaloza, *Cybersecurity attack shuts down colonial pipeline*, May 2021. [Online]. Available: https://web.archive.org/web/20210508181849/https://www.npr.org/2021/05/08/995040240/cybersecurity-attack-shuts-down-a-top-u-s-gasoline-pipeline (visited on 07/07/2021).

[4] S. Gadhiya and K. Bhavsar, "Techniques for malware analysis," en, *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 4, pp. 972–975, Apr. 2013.

[5] *IDA Pro*, en. [Online]. Available: https://www.hex-rays.com/products/ida/ (visited on 12/18/2020).

[6] NSA, *Ghidra*. [Online]. Available: https://ghidra-sre.org/ (visited on 12/18/2020).

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, Mar. 2003, ISSN: 1532-4435.

[8] D. Blei, T. Griffiths, M. Jordan, and J. Tenenbaum, "Hierarchical topic models and the nested chinese restaurant process," *Advances in Neural Information Processing Systems*, vol. 16, May 2004.

[9] L. Dietz, *Topic model evaluation: How much does it help?* en.

[10] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, en, Second Edition. Springer. [Online]. Available: https://web.stanford.edu/~hastie/ElemStatLearn/.

[11] G. G. Sundarkumar, V. Ravi, I. Nwogu, and V. Govindaraju, "Malware detection via API calls, topic models and machine learning," en, in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, Gothenburg, Sweden: IEEE, Aug. 2015, pp. 1212–1217, ISBN: 978-1-4673-8183-3. DOI: 10.1109/CoASE.2015.7294263.

[12] J. Greer, "Unsupervised interpretable feature extraction for binary executables using LIBCAISE," en, Masters Thesis, University of Cincinnati, 2019. [Online]. Available: https://etd.ohiolink.edu/apexprod/rws_olink/r/1501/10?p10_etd_subid=180585&clear=10 (visited on 12/18/2020).

[13] O. Djaneye-Boundjou, T. Messay-Kebede, D. Kapp, J. Greer, and A. Ralescu, "Static analysis through topic modeling and its application to malware programs classification," in *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, Jul. 2019, pp. 226–231. DOI: 10.1109/NAECON46414.2019.9057876.

[14] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv:1802.10135 [cs]*, Feb. 2018. arXiv: 1802.10135 [cs].

[15] E. B. Fernandez, M. M. Larrondo-Petrie, and A. E. Escobar, "Contexts and context-based access control," in *2007 Third International Conference on Wireless and Mobile Communications (ICWMC'07)*, Mar. 2007, pp. 73–73. DOI: 10.1109/ICWMC.2007.30.

[16] B. Shebaro, O. Oluwatimi, and E. Bertino, "Context-based access control systems for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 2, pp. 150–163, Mar. 2015, ISSN: 1941-0018. DOI: 10.1109/TDSC.2014.2320731.

[17] B. Shrestha, D. Ma, Y. Zhu, H. Li, and N. Saxena, "Tap-wave-rub: Lightweight human interaction approach to curb emerging smartphone malware," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 11, pp. 2270–2283, Nov. 2015, ISSN: 1556-6021. DOI: 10.1109/TIFS.2015.2436364.

[18] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "Context-aware, adaptive, and scalable android malware detection through online learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 3, pp. 157–175, Jun. 2017, ISSN: 2471-285X. DOI: 10.1109/TETCI.2017.2699220.

[19] R. Svensson, *Das malwerk*. [Online]. Available: https://www.dasmalwerk.eu/ (visited on 12/21/2020).

[20] *Objdump(1) - linux manual page*. [Online]. Available: https://man7.org/linux/man-pages/man1/objdump.1.html (visited on 07/08/2021).

[21] *Cat(1) - linux manual page*. [Online]. Available: https://man7.org/linux/man-pages/man1/cat.1.html (visited on 07/08/2021).

[22] *Sed, a stream editor*. [Online]. Available: https://www.gnu.org/software/sed/manual/sed.html (visited on 07/08/2021).

[23] *Cut(1) - linux manual page*. [Online]. Available: https://man7.org/linux/man-pages/man1/cut.1.html (visited on 07/08/2021).

[24] *Win32/Ramnit*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Ramnit (visited on 07/08/2021).

[25] *Adware:Win32/Lollipop!MSR*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Lollipop!MSR (visited on 07/08/2021).

[26] *Win32/Kelihos*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Kelihos (visited on 07/08/2021).

[27] *Win32/Vundo*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=Win32%2FVundo (visited on 07/08/2021).

[28] *Win32/Simda*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Simda (visited on 07/08/2021).

[29] *Win32/Tracur*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FTracur (visited on 07/08/2021).

[30] *VirTool:Win32/Obfuscator.ACY*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/Obfuscator.ACY (visited on 07/08/2021).

[31] *Win32/Gatak*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=Win32/Gatak (visited on 07/08/2021).

[32] T. Messay-Kebede, B. N. Narayanan, and O. Djaneye-Boundjou, "Combination of traditional and deep learning based architectures to overcome class imbalance and its application to malware classification," in *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, Jul. 2018, pp. 73–77. DOI: 10.1109/NAECON.2018.8556722.

[33] Y. Zhang, Q. Huang, X. Ma, Z. Yang, and J. Jiang, "Using multi-features and ensemble learning method for imbalanced malware classification," in *2016 IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 965–973. DOI: 10.1109/TrustCom.2016.0163.

[34] S. Yue, "Imbalanced malware images classification: A CNN based approach," *arXiv*, Aug. 2017. [Online]. Available: http://arxiv.org/abs/1708.08042 (visited on 07/08/2021).