

Union College Union | Digital Works

Honors Theses

Student Work


6-2015

Classifying System Call Traces using Anomalous Detection

William Doyle

Union College - Schenectady, NY

Follow this and additional works at: <https://digitalworks.union.edu/theses>

 Part of the [Computer and Systems Architecture Commons](#), [Digital Communications and Networking Commons](#), and the [Information Security Commons](#)

Recommended Citation

Doyle, William, "Classifying System Call Traces using Anomalous Detection" (2015). *Honors Theses*. 295.
<https://digitalworks.union.edu/theses/295>

This Open Access is brought to you for free and open access by the Student Work at Union | Digital Works. It has been accepted for inclusion in Honors Theses by an authorized administrator of Union | Digital Works. For more information, please contact digitalworks@union.edu.

Classifying System Call Traces using Anomalous Detection

By

William J. Doyle III

Submitted in partial fulfillment
of the requirements for
Honors in the Department of Computer Science

UNION COLLEGE

June, 2015

Abstract

DOYLE, WILLIAM J., III Classifying System Call Traces using Anomalous Detection. Department of Computer Science, June 2015.

ADVISOR: Cass, Aaron

We used data mining techniques to detect intrusions among system call traces and have outlined our results. Recent work at the intersection of security and machine learning has lead to better understanding of anomalous intrusion detection. There is a need to more thoroughly understand how anomaly detection can be used because of its potential applications and advantages over current standard methods. In this thesis, we report on a new approach of anomalous detection using system call traces. Our goal is to be able to create a system that can accurately detect hacking attacks by analyzing the sequences of system calls the operating system is performing. We will look at how this data can be processed to achieve correct detection of intrusions on a system. In the end, we will outline ways in which system call traces can be leveraged as well as what we can do and learn from these results.

Contents

1	Introduction	1
2	Background and Related Work	2
2.1	Previous Work	2
2.2	Network Intrusion Detection	5
2.3	kNN	6
2.4	kMC	7
2.5	SVM	7
2.5.1	Dictionary Method	7
2.5.2	Short Sequences	8
2.6	ELM	9
2.7	Previous Conclusions	10
3	Methods and Design	11
3.1	Processing the Data	12
3.1.1	Overview of the ADFA-LD	12
3.1.2	N gram Approach	15
3.2	Reducing the Attributes	15
4	Early Results	16
4.1	SVM	17
4.2	kNN	17
4.3	Early Results Discussion	18
5	Reduction Results	18
5.1	Analyzing a small subset of the data	18
5.2	Using all of the data	20

6 Conclusion and Discussion	20
7 Acknowledgments	22

List of Figures

1	kNN using Euclidean Space Accuracy against False Positive Rate[9]	6
2	kMC using Cosine Space Accuracy against False Positive Rate[9]	7
3	SVM and ELM using the Dictionary Method[8]	8
4	SVM using the Short Sequences[11]	9
5	ROC curve for compressed traces (subset)	18
6	ROC curve for uncompressed traces (subset)	19
7	ROC curve for compressed traces (full)	21
8	ROC curve for uncompressed traces (full)	21

List of Tables

1	Accuracy and False Positive Rates	2
2	Part of a System Call Table for Linux 2.6.38	11
3	Attacks on the System	12
4	ADFA-LD Structure	12
5	Early Results	17

1 Introduction

Computer security becomes ever more important as we build more advanced computer systems. Vulnerabilities in a computer system are starting to become more easily exploited as the types of attacks evolve over time. To this end, research is turning towards understanding trends within system data to be able to detect intrusions quicker and more efficiently. [10]

Consider a scenario where someone is inside of a cafe on an open network. Unknowing to them they have become the victim of an attacker who plans to perform malicious actions on the machine. Being able to detect these types of intrusions before the attacker performs their malicious actions is critical to the safety of the data in a modern computing world. One way we can go about doing this is to “look” for specific parts of the attack happening. Perhaps, in the cafe network example, the attacker uses the file system in a way that is different from the way a typical user would. We can use this piece of information to create a detection system that will look for these types of anomalies on the computer. Some examples of these anomalies are unusual read/write behavior, multiple attempts to log-in to the machine, or being on an unsecured connection.

Host-based Intrusion Detection Systems (HIDSs) [1] have been shown to be an effective means of protection on modern systems. Such systems are able to detect intrusions on an individual host by analyzing the information that is available on the machine (e.g. system calls, network traffic, etc.). Two common ways of implementing a host-based intrusion detection system are using a signature-based [17] or an anomaly-based [18] approach. A signature-based system will attempt to match current behaviors occurring on the system against a known database of intrusion-type behavior. In this method, what the specific attacks “look like” are known before the intrusion occurs.

On the other hand, an anomaly-based system has a constructed model of normal behavior. The detection system indicates an attack when the observed behavior statistically deviates from the modeled normal behavior. Signature-based approaches tend to have higher detection rates but they perform poorly when introduced to a new attack unknown to the detection system. For this reason, intrusion detection research is turning towards understanding anomaly-based systems. [11] [1]

The rest of this paper is organized as follows. Within Section 2 and will outline previous work that has

Method	Accuracy	FPR	Authors
kNN	50%	50%	Miao Xie et al.
kMC	60%	20%	Miao Xie et al.
SVM sequences	70%	20%	Miao Xie et al.
SVM dictionary	80%	15%	Gideon Creech et al.
ELM	90%	15%	Gideon Creech et al.
Genetic Alg.	95%	30%	M. Hoque et al.

Table 1: Accuracy and False Positive Rates

been done on the problem of anomaly detection with system calls as well as some other approaches. In Section 3 will begin to discuss our methods and what we hope to accomplish as well as an overview of the ADFA-LD, the dataset we used during our experiments. Section 4 will describe our experiment setup and results. Finally, we will draw conclusions from these results and describe what we hope can be done next.

2 Background and Related Work

2.1 Previous Work

Before we begin to discuss some of the work done by others in the field we explain some of the key terms involved in the discussion. A system call is a fundamental interface between a program on a machine and the operating system. The system call is a way for a user program to request an operation to be completed by the operating system. For example, a program may want to open a file and then copy it to a new directory. These tasks can be accomplished by using some sequence of system calls. As a result, programs running on the machine will produce sequences of system call, also known as system call traces. In later sections we will go into more detail about system calls and system call traces for now we can think of them in this fundamental way.

There are a few algorithms used within this section that we outline here:

kNN The k nearest neighbor (kNN) algorithm will classify data based on a distance metric. After a new data point is given to the model it will calculate how close in distance it is to other data points that already have been classified. How large of a distance area the algorithm looks at depends upon the

value of k . In the end, the new data point is given the class of the majority of the other points in the distance area.

kMC The k means clustering (kMC) algorithm takes a set of already classified data points and partitions them into “clusters” of data based on their features. The algorithm then creates a “prototype” for each of its clusters. One way the algorithm could accomplish this task is by taking an average of the data points already in the cluster. These prototypes are used to classify a new data point introduced to the model. The prototype that most closely resembles the new point will be the cluster the point is classified under.

SVM A support vector machine (SVM) will classify data by constructing a high-dimensional space which separates the data points it is given. We can think of a support vector machine as a way of separating data that is very close to each other, however, instead of attempting to make a one-dimensional cut, a straight line, in the data a SVM will make a multi-dimensional cut, for example a hyperplane.

ELM An extreme learning machine (ELM) uses a single hidden-layer neural-network implementation to randomly choose its hidden nodes and determine the weights of its output layer in the neural-network. This algorithm performs very fast when learning is performed with new set of data.

GA Genetic algorithms (GA) are a type of heuristic search that attempts to mimic the process of natural selection. The heuristic is generally interested in generating the optimal solution to the given problem. The algorithm accomplishes this optimization by taking an initial population of data, mutating and creating new data to add to the population. In the end, those left in the population are the optimal solutions to the problem given to the genetic algorithm.

Next, we review the results of other researchers and how we will evaluate future performance. The evaluation outlined in this section was performed by other researchers as outlined in Table 1. The primary way these people examined their results was through looking at ROC (Receiver Operating Characteristic) curves within the literature, and will be what we present here. A ROC curve is a graph of the accuracy of a method plotted against its false positive rate, see Figures 1, 2, 3, and 4 below for some examples. When reading the graphs we are looking at what happens when the accuracy rates go up. If the false positive rate

increases uniformly with the accuracy we know we have a poor classifier. On the other hand, if the false positive rate were to increase at a much slower rate than the accuracy then we would have an acceptable classifier.

The second section will cover k nearest neighbor (kNN), the third covers k means clustering (kMC), the next section will be about using support vector machines (SVM) and the last section will use a semantic approach with an Extreme Learning Machine (ELM). The results of these methods can be seen in Table 1.

These four methods were published within three papers. In the first paper, M. Xie et. al. [9] were interested in seeing how changing the way the distance was calculated affected classifications which involved distance. More specifically, they were trying to minimize the frequency vectors used during classification. Frequency vectors are vectors where each position is a weighted average of how often that system call would appear in any given sequence for each possible system call. Hence, these vectors are of length 324, because there are that many possible system calls in the data see Table 2, with each slot being the frequency of that system call appearing divided by the total number of system calls in that sequence. They then chose to pick two of the most common frequency-based algorithms in which they could vary how distance was calculated: kNN and kMC. The second paper M. Xie et. al. [11], attempts to use the data in a non-frequency based approach. They make this decision because they believed that their results from the kNN and kMC techniques to have poor performance. Instead, they use a one-class SVM and short sequences (shortened frequency vectors) from the dataset to classify anomalies. The third paper, using a semantic approach to classifying anomalies using multiple dictionaries as inputs to an ELM and SVM. This technique showed amazing results for this dataset but it came with a computational heavy cost. As mentioned before, the kNN and kMC experiments use many different types of distance metrics depending on which algorithm was being used. For kNN Euclidean, standardized Euclidean, Minkowski, Cosine and Mahalanobis space were used. While kMC used Euclidean, Minkowski, Cosine and Correlation space.

In addition to techniques analyzing system call traces we will look at some work not involving this approach. One such technique involves network based intrusion detection.[14] This system monitors and analyzes network traffic to protect a machine from network-based attacks. It will attempt to detect malicious actions such as a denial-of-service attack or port scans. The network based intrusion detection system

scans network traffic packets close to real time in order to detect abnormal behavior. Moreover, we will examine how some researchers have been using genetic algorithms techniques to create a network-based intrusion detection system.

2.2 Network Intrusion Detection

There is a large amount of data a network intrusion detection system must process in order to run efficiently. These types of systems became an emerging topic of research during the expansion of the internet across the world. At that time research with host based systems was prevalent. One approach that came about was the State Transition Analysis Technique (STAT) and it was applied to network intrusion detection. The NetSTAT [15] modeled network attacks as state transition diagrams, where states and transitions are characterized in a network environment. The network environment itself was modeled with graphs that have multiple edges connecting many nodes together. However, the NetSTAT tool was only developed to allow users to analyze their networks instead of a full intrusion detection system.

Another more successful approach to network intrusion detection involves the use of genetic algorithms. This model [16] was divided into two phases: precalculation and detection. During the precalculation phase the model would analyze an initial random population and as it traversed through the network training data. If the model found some part of the population to be close to the new data, the model would merge with the new data and create a new population. Otherwise, an entirely new population is created with the training data without merging with parts of the initial random population. During detection their genetic algorithm would run and attempt to optimize the fitness level in the population to detect abnormal behavior. In other words, the model was identifying which parts of the population are normal with a high fitness level, and whenever some data that has low fitness appears it was detected as abnormal behavior. With this model the researchers were able to have a detection rate of about 95% and a false positive rate of around 30%.

2.3 kNN

The k nearest neighbor algorithm is the most used algorithm within the area of anomaly detection. [9] However, when used to detect anomalies among the system call traces it failed to effectively detect attacks even when the system call trace inputs were much shorter in length. Looking at Figure 1 we can see that just using Euclidean space for each attack high/low accuracy was followed by high/low false positives. Within the scope of a problem that only has two classes this would seem to mean that kNN is no better than merely guessing.

The evaluation of the algorithm was done by comparing how well the algorithm classified each attack separately. Because of the poor performance of the kNN algorithm not much is discussed in terms of evaluation purposes. Instead the paper turns its attention towards the kMC algorithm because it is computationally less expensive than kNN and has a much better anomaly detection rate using certain distance metrics. We can deduce from the performance of kNN that this dataset is not as frequency dependent as it may seem. Let us examine the results of the popular clustering algorithm kMC.

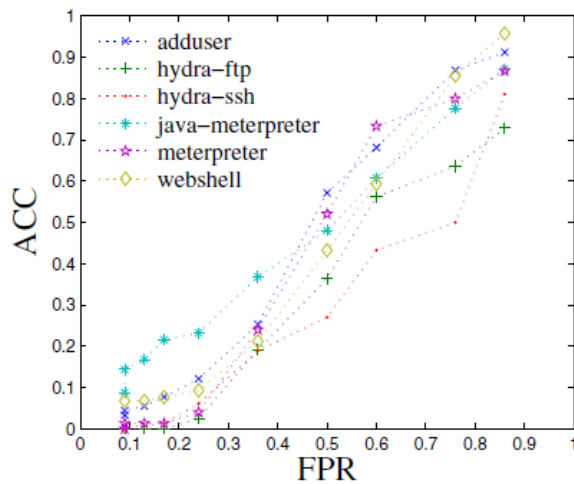


Figure 1: kNN using Euclidean Space Accuracy against False Positive Rate[9]

2.4 kMC

Classifying most of the attacks with the kMC algorithm achieved an accuracy of 60% or higher with a false positive rate of around 20%. Of all of the attack types, seen in Figure 2, the Java Meterpreter was the easiest to detect while the Hydra-FTP and Hydra-SSH can not effectively be detected using these frequency-based approaches. In terms of a distance metric the best performing metric evaluated was the cosine distance, in Figure 2 we can see the accuracy plotted against the false positive rate for the kMC algorithm.

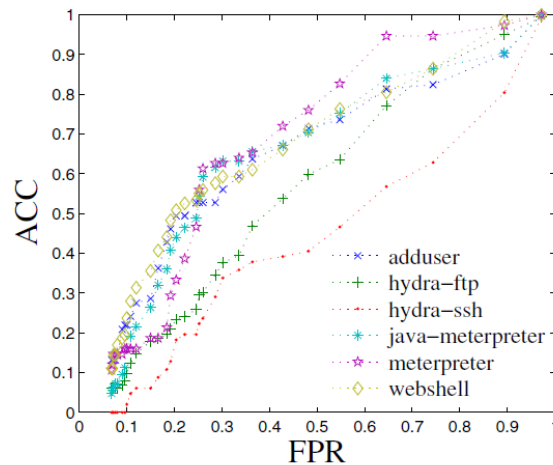


Figure 2: kMC using Cosine Space Accuracy against False Positive Rate[9]

2.5 SVM

We have two separate SVM approaches using the system call traces to classify anomalies. Thus, we will separate this section into two subsections one using short sequences the other using the dictionaries.

2.5.1 Dictionary Method

We will first examine the dictionary method of setting up the data for use inside of an SVM. Creech et. al. [8] created multiple dictionaries which contained all the possible combinations of sequences from the normal and attack dataset. Essentially, they created an algorithm which read through the dataset in multiple passes

and created “rulesets” for which sequences are normal and which are abnormal for all the sequences in the dataset. These rulesets contain all possible combinations of the attack and normal sequences as a look-up table. The algorithm created a dictionary for each system call number in the dataset. The dictionary corresponded to one system call and the entries to the dictionary were other system calls and the type of behavior that this sequence was classified under. It was from these many dictionaries that the “rules” for which particular sequences were attack or normal was generated. The “rulesets” were the sets of the rules from analyzing the dictionaries that would classify the sequence as attack or normal. This process took over a week to complete, however they were able to achieve an accuracy of around 80% with a false positive rate of 15%, seen in Figure 3. But because of the significant computations needed to create the dictionaries other methods were sought out.

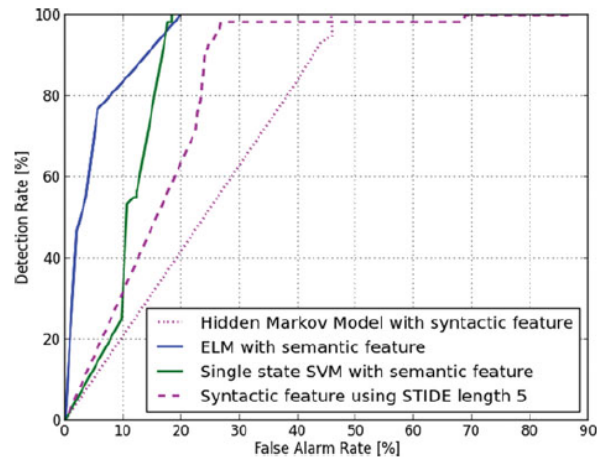


Figure 3: SVM and ELM using the Dictionary Method[8]

2.5.2 Short Sequences

The short sequence approach attempts to address the issue of computation cost and time. Short sequences are similar to frequency vectors except they only look at the most frequent sequences making them shorter on average. For example, imagine a dataset primarily containing the frequently occurring sequences “A A B”, “B B A” and other very infrequently occurring sequences. However, lets say that we have determined

that these two primary sequences makeup over say 90% of the data, we would disregard the other infrequently occurring sequences when making our frequency vectors and only examine these most frequent sequences. How short is dependent upon the frequency of the sequence occurring in the dataset.

As a result, this method can be produced much faster, and yielded results comparable to the dictionary method. An accuracy of around 70% was achieved with a false positive rate of 20%, seen in Figure 4.

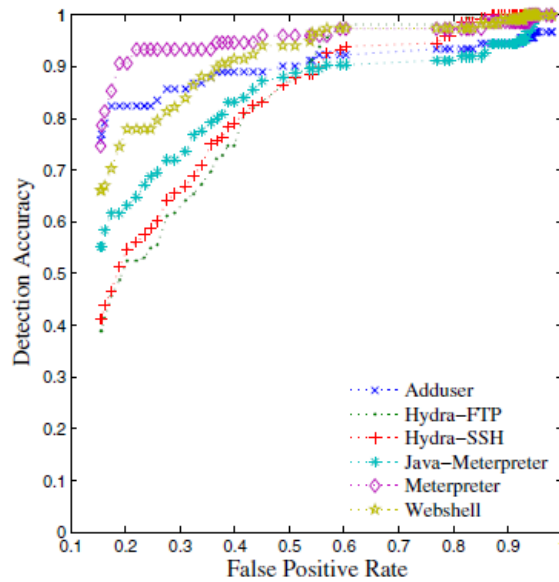


Figure 4: SVM using the Short Sequences[11]

2.6 ELM

An Extreme Learning Machine was used with the dictionary method as a form of input as described previously. What makes the result from these experiments remarkable is that it is substantially higher than most of what has been done with system call traces before. This result shows that this dataset does contain the necessary information to learn how to detect anomalies from. However, as described before, the practical use of this technique is moot because of the computational cost of creating the dictionaries. For example, by the time the dictionaries have been created the normal behavior of the system may have changed making

the dictionaries useless. The results of the ELM can be seen in Figure 3 with an accuracy of around 90% and a false positive rate of around 15%.

2.7 Previous Conclusions

First, we see how one group of researchers created a means of analyzing a network in order to attempt to detect intrusion. Although the project was never turned into a full-fledged intrusion detection system, they demonstrate the value of network-based intrusion detection. Next, we see a different approach to intrusion detection by M. Hoque et. al. [16] using network based techniques rather than system call traces. The results that are seen are considered to be sufficient detection rates in the field. However, they do suffer at the cost of a large false positive rate. Nevertheless, their work demonstrates the power that genetic algorithms can lend to the intrusion detection problem.

M. Xie et. al. [9] concluded that kNN is ineffective against detecting attacks, while kMC can detect most attacks. The paper also spoke of needing to understand the characteristics of attacks that make up system call traces in order to create more effective ways of formatting the data for detection. These conclusions are very straight forward and can be observed in Figure 1 and 2. The kNN algorithm is only as good as guessing while kMC shows some promise of being able to learn what the attacks are by clustering.

M. Xie et. al. [11] also found that short sequences are able to show separability between normal and attack sequences, while maintaining computational costs low. This technique can be implemented in a much shorter time as compared with the dictionary SVM methods described previously. However, the weighting of the short sequences still is not optimal. Here, the conclusions made seem to hold the most promising results for future work. Going forward, if a way of creating more distinct short sequences can be made then anomalies can be more easily distinguished by the SVM.

Finally, G. Creech and J. Hu [8] demonstrated the power of semantic analysis when using a system call traces. The results seen are really substantial however they practically do not hold value because of the overhead required computationally. Although, this doesn't rule out the value of a semantic approach. If a method can be found that will substantially decrease the input needed for the ELM then they can be used practically.

Number	Name
0	sys_restart_syscall
1	sys_exit
2	sys_fork
...	...
265	sys_clock_gettime
...	...
322	sys_timerfd_create
...	...
324	sys_fallocate

Table 2: Part of a System Call Table for Linux 2.6.38

In the next section we will discuss the methods that we used and how we went about designing the classification experiments.

3 Methods and Design

Now that we have an understanding of the problem and what has been attempted so far we can begin to outline the work we have done.

Our work is based on the work of M. Xie et. al. [11] Recall that M. Xie et. al. found short sequences of the larger system call sequences show separability between which are normal versus attack. We are then going to analyze the data using some idea of sub-sequences. N grams have been used as models to understand sequence data [12] as well as in predicting a sequence of events [13]. An n gram is a short arbitrary length, denoted by the n, subsequence of a larger sequence. In this context, we took subsequences of the system call traces of some length. We have created a system that generated these n grams and we have analyzed them with certain classification algorithms.

Next, we will briefly explain the dataset we are using throughout our experiments and model building.

Name	Method
Brute force Password	FTP by Hydra
Brute force Password	SSH by Hydra
Add new superuser	Client side malicious executable
Java Based Meterpreter	TikiWiki vulnerability
Linux Meterpreter Payload	Client side malicious executable
C100 Webshell	PHP Remote file Inclusion Vulnerability

Table 3: Attacks on the System

Group	Number of Traces
Normal Data	833
Validation Data	4373
Hydra-FTP	162
Hydra-SSH	148
Adduser	91
Java-Meterpreter	125
Meterpreter	75
Webshell	118

Table 4: ADFA-LD Structure

3.1 Processing the Data

3.1.1 Overview of the ADFA-LD

This dataset was generated by Creech et. al. to be used when designing and evaluating anomaly based intrusion detection systems using system calls. Our approach involves the use of this dataset, but before we begin to examine the dataset we will give a short introduction into the type of data that it contains: system call sequences. As presented previously, a host-based intrusion detection system examines local information on the system. System calls are a fundamental interface between a program on a machine and the operating system. The effective use of system calls within host-based anomaly detection was first proposed by Forrest et. al. [2] The major two advantages found were:

- (i) System calls are more dangerous than a typical user process because a system call will have greater access to the system's resources
- (ii) An operating system will have a finite list of operations, which creates predictable system call se-

quences under normal system operations

This last point is especially useful within the scope of anomaly detection, because it allows us to generate a predictive model which we can use to check for abnormal deviations on the system, and because its a finite list, in a reasonable amount of time. The ADFA-LD contains these system call “traces” of a server under normal operation and while it is being attacked. We can analyze these traces and look for ways to be able to classify what an attack versus normal operation is.

The ADFA-LD is a collection of system call traces. A single trace is a sequence of system call numbers, see Table 2, over some arbitrary time period. Each trace is discontinuous from the other traces and may contain more or fewer system calls. The traces were collected using the Linux audit daemon Auditd [3] on Ubuntu Linux 11.04. To allow for certain attacks to take place, Apache Version 2.2.17 running PHP version 5.3.5 were installed and enabled. The operating system was full patched and FTP, SSH, and MySQL 14.14 were running on their default port settings. TikiWiki 8.1 was installed because it was known to have a vulnerability to allow for a certain attack on the system. [6] This system configuration represents a modern Linux server with some small vulnerabilities build in place.

The attacks the system were placed under were carefully chosen to use methods that a modern-day hacker or penetration tester would have access to. In Table 3, we can see the breakdown for the different types of attacks in the dataset as well as the methods used to implement them. The small vulnerabilities were purposefully engineered into the system to be able to collect the data appropriately. Overly exploitable servers are rare and the use of such a server would be without purpose for the scope of the problems this dataset aims to address.

The first two attacks represent a brute force password guessing attempt on the open FTP and SSH services. This attack is often a last resort of most attackers due to its large potential to be caught out on my intrusion detection systems. For the next attack, an executable was made that would create a new user with root privileges. The executable was uploaded onto the server and was triggered remotely using social engineering. The next two attacks involve Meterpreter, this program is a command shell with enhanced functionality. The TikiWiki vulnerability allows to unknowingly upload a copy of the Java Meterpreter. Once the program was uploaded it would initiate a reverse TCP connection to attack the system. Once the

command shell was installed it could have greater access to changing the configuration of the system, escalating privileges or attempting to access the shadow password file. The Linux Meterpreter was uploaded using social engineering and behaves similarly to the Java Meterpreter. The difference between the two is only the implementation of the program itself, which could lead to different system call traces. Lastly, the C100 Webshell is a sophisticated piece of PHP which allows shell access to the attack through a web browser.

This set of attacks represents what an average attacker may use in an attempt to take advantage of a system and compromise it. The set ranges from low level, high profile brute force attempts to vulnerabilities within services running on the machine. The use of these attacks will yield traces that use modern hacking methods. [7]

Now that we have an understanding of the attacks being used within the dataset let us go into some more detail on its contents. The ADFA-LD is publicly available and contains virtually no mark-up or formatting of any kind. The reason for this is so that future researchers can manipulate the data from its purest form: the system call traces. Again, a system call trace is simply a sequence of system call numbers dictating some sequence of events happening on the machine. The data was designed for anomaly-based intrusion detection systems, not signature-based recognition. [8] It is organized into three separate groups each containing the pure system calls. The groups are: normal, attack (which contains separate attack traces groups) and validation data, see Table 4.

The traces were collected while the system was either being attacked or under normal operations. Both the normal and validation groups contain traces for normal operations of the system, while the attack set naturally contains attack traces. The data can be used to answer the simple question: is there an attack occurring on the system? This can be answered by classifying the traces as either normal or abnormal behavior for a given system. Since there are only 324 possible system calls within a single trace, due to the operating system, we can view them as being the attributes of the raw dataset. Additionally, because the data is used to classify anomalies we have two classes: anomaly or not. As a result of being used for anomaly detection, the effectiveness of classifying the anomalies is entirely dependent upon manipulating the system call traces appropriately.

3.1.2 N gram Approach

The data, as described before, is divided into three major categories: normal, attack, and validation. First we preprocessed the data so that each major category is broken down into ten separate bins. By doing this preprocessing, we have thirty data files to work with, ten for each category. We performed this preprocessing so that we can run a ten-fold cross validation on the data. The attack data is broken down into its individual attack types as outlined in Table 3 and seen in the dataset structure in Table 4. We are concerning ourselves with only learning to divide the data into two classes: attack or normal.

Once we completed this process we then started to create subsequences for each of the bins. In order to do this we first took some sequence from the dataset. Next, we broke down this sequence into a series of subsequences that we used as the series of observations of the sequence from the dataset. These observations are what we used to learn and predict the behavior of a sequence. Remember, we want to determine if the data represents attack or normal behavior. The size of the subsequence is denoted by the value n . When we broke down the sequence into its subsequences we wanted to make all of the subsequences the same size. For example, with four grams, we would take all of our sequences from the dataset and break them up into subsequences of length four. At the end of the generation of the n grams we had the thirty data files, described previously, to work with. Each of these files was a list of the separated sequences transformed into n grams. Using ten-fold cross validation, we trained a classifier on part of the data and tested the classifier on another. Because n grams separate the sequences out into shorter subsequences we needed a way to classify the sequence as a whole. We then analyzed the subsequences of a sequence. The sequence were called normal if 90% of its subsequences are normal; otherwise, it was classified as an attack. Afterwards, we compared to the true class, the ground truth, of that specific sequence. In the end, we had a classification of the n grams.

3.2 Reducing the Attributes

After gathered the results from just classifying the data we began to eliminate dimensionality of the problem. There are 324 possible attributes in any given sequence. The issue is that it is difficult to be able to quantify what constitutes an attack versus a normal sequence. The approach we used to reduce the num-

ber of attributes in a sequence is to collapse down a subset of the attributes into one attribute. Removing attributes from the dataset appears to be counter-intuitive because our goal was to learn what attacks and normal behavior look like. Taking away information would seem to be taking away information we can learn from. However, we must consider that much of what happens on a system is typically going to be normal behavior of either the user or just the system alone. One can imagine that when attacks occur on modern systems they are generally marked by a small period of operation because they want to be stealthy to avoid detection. As a result, when we take a trace of system calls much of what we learned from will be normal behavior. So in an effort to narrow down what we learned we want to reduce some of the attributes that are not occurring very often during attacks versus normal behavior. Additionally, we performed this collapsing in order to avoid overfitting the data. We wanted to make sure that we are not analyzing the data by considering all possible system call sequences. Instead, by collapsing some of them down into one system call we were able to focus the model on the important system call sequences. This process avoided overfitting because we did not learn precisely every sequence, instead we chose to ignore some of the system calls.

In order to determine what we wanted to collapse down we began by analyzing the percent frequency of the data. In other words, we wanted to find out which system call attributes are appearing the least and in what part of the data. From that point we began to choose out the ones that may benefit from being reduced and analyze the results. We have found that by collapsing the infrequently occurring parts of the data the model can focus on the more important frequently occurring data.

The next section will discuss and evaluate some of the early results that we saw during the beginning of our research with the methods outlined above.

4 Early Results

Shown in Table 5 we have the results of using our method with the k nearest neighbor and support vector machine classifiers. The implementation of our experiments was written in Python [4] using the scikit-learn package [5].

The results we saw, in the early stages experimentation, by analyzing various subsets of the data showed

Classifier	Classification Accuracy
k nearest neighbor	34%
support vector machine	64%

Table 5: Early Results

promising results. Referencing Table 1 as a guide for success we can begin to understand the early results gathered from our own experiments.

4.1 SVM

The support vector machine outperformed the other classifier as well as having been the closest to prior results using the same classifier. Support vector machines do not depend on the dimensionality of the problem. Our classification without reducing the attributes has many features under consideration during classification. However, a support vector machine is not limited computationally by this aspect. Additionally, when we have a result from a support vector machine we can feel confident in the model's future predictions. The reason for this is because support vector machines do well to not over-fit themselves when they are fitted to some data. Going forward, we aimed to use this classifier to generate even better results because of these features of the classifier.

4.2 kNN

Not much optimization of the k nearest neighbor classifier was performed. However, we performed experiments with this classifier to compare our method against prior results. We have ran a basic implementation of the classifier and have received the poor results seen in Table 5. These results are not close in relation to the literature's success with this algorithm. A k nearest neighbor classifier is used because of its simplicity in design and quick computational benefits. Generally, we can use this type of classifier as a base-line of performance and we will do precisely that. Some further optimization of this classifier is required to be able to set up a confident enough baseline of results. Having seen the excellent comparative performance of the support vector machine we have focused our future experiments using the SVM.

4.3 Early Results Discussion

We have set up a proof of concept for the method outlined previously as demonstrated by our results with the support vector machine classifier. These preliminary tests will be improved on going forward as we optimize the classifiers. We have shown that using n-grams as a means of classification does not degrade performance when compared to other results using the same classifier. The support vector machine has demonstrated superior performance as opposed to the k-nearest neighbor approach. Therefore, going forward we will use the support vector machine when analyzing the results from reducing the system calls. Recall, that we are going to reduce the system call traces based on the frequency a system call is used throughout the data set. In the next section, we will examine the results of using the support vector machine along with the reduction of the attributes described previously.

5 Reduction Results

5.1 Analyzing a small subset of the data

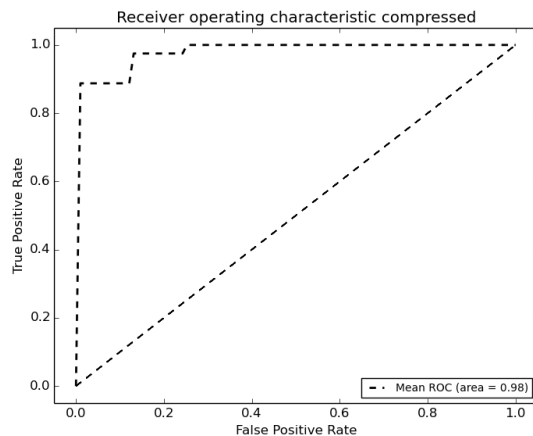


Figure 5: ROC curve for compressed traces (subset)

Within this section we outline the results of applying the reduction scheme to the data described previously. In essence, we wanted to see if by performing this reduction on the system call traces we could

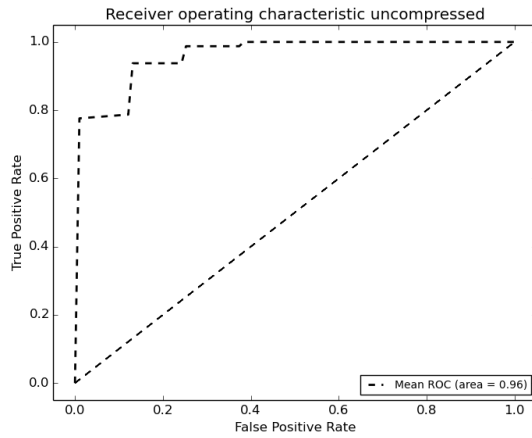


Figure 6: ROC curve for uncompressed traces (subset)

increase the performance of the support vector machine classifier. To accomplish this we analyzed receiver operating characteristic curves for both the compressed and uncompressed versions of the classifier.

Looking at Figure 5 and Figure 6 we can see a clear difference between the compressed and uncompressed classifiers. In the compressed ROC curve we have a larger area under the curve, 0.98, while the uncompressed version has the a smaller arena under the curve, 0.96. However, just by looking at these graphs is not enough to distinguish between the two classifiers. It could be the case that these two classifiers are the predicting the same results and there is just small enough of a variation between the results that yields the different ROC curves. To test for statistical significance of the models we performed a sign test on the classification results of these two models. We received a probability value of 0.0206 which is low enough to indicate that these two models are statistically different from each other. In other words, this result means that our compression hypothesis appears to be true. We can increase classification results by analyzing compressed versions of the system call traces. What remains to be seen is if this result will hold true by analyzing larger amounts of data.

5.2 Using all of the data

We analyze the results of using all of the data in combination with the reduction scheme outlined before. Remember, we wanted to identify if performing this reduction on the system call traces will increase the performance of our support vector machine classifier. As we did in earlier sections, we accomplished this analysis by looking at receiving operating characteristic curves for compressed and uncompressed versions of the classifier.

Comparing Figure 7 and Figure 8 we can immediately see that there appeared to be a difference between the two versions of the classifiers. The compressed ROC curve has a larger area under the curve, 0.99, than the uncompressed model, 0.95. As we did when we performed the analysis of the model with a small subset of the data we performed a sign test on the classifiers in order to detect statistical significance. However, the sign test did not detect a statistical difference between the two models. Additionally, the classification accuracy of the models were not different either, both the uncompressed and compressed versions reached around a 60% classification accuracy. We can not conclude whether these two models are not statistically different from each other. These results indicated that we our hypothesis can not be conclusively proven false. It appeared that with large amounts of data it was more difficult to differentiate between models. In other words, our results showed inconclusive evidence that the two models are different. However, statistically this result also told us that we can not prove that they two models are the same either.

6 Conclusion and Discussion

We have seen that as a result of reducing some of the system calls down into one attribute we can increase the performance of our support vector machine classifier up to a point. This result leads us to conclude that our initial hypothesis was partially correct. When using a small amount of data, we can increase classification by using this reduction scheme on the system call traces. Additionally, as previously noted, the n-gram approach to representing the system call trace data set does not degrade performance of the classifiers. Using the frequency based reduction approach we can see a statistically significant improvement in the classifiers when analyzing the model on a small amount of data.

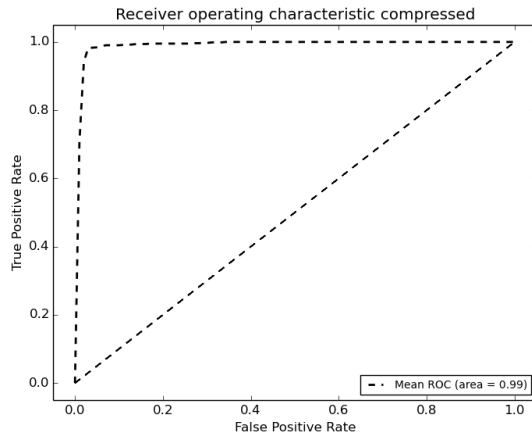


Figure 7: ROC curve for compressed traces (full)

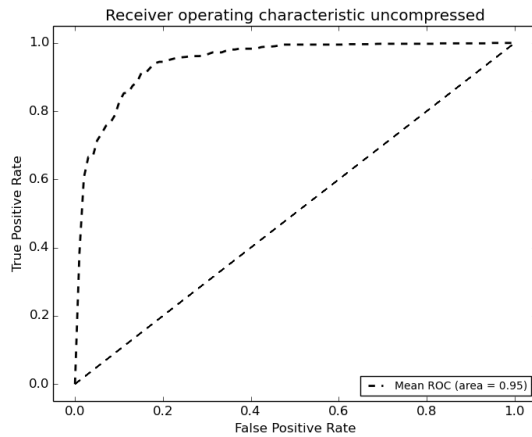


Figure 8: ROC curve for uncompressed traces (full)

The reduction scheme does not conclusively improve the performance of the classifier when the model has access to more data to learn from. A possible reason for this result may be due to how we implemented the reduction. We kept the frequency threshold during reduction the same in both experiments. However, one possible explanation is that because we increase the amount of data we needed to adjust the reduction threshold as well. Otherwise, we may not be taking into account enough of the attributes during the reduction step of preprocessing. In the future we need to evaluate the reduction algorithm in order to see if we

can come up with statistically different models.

For future work on this project we would like to implement an intrusion detection system which uses this compression technique along with the n-gram system call analysis. This project would require manipulation of an operating system's kernel code in order to be able to audit the machine properly to receive the system call traces. From there the model would be put into place as a monitoring system that will keep track of some number of the system call traces within some windowed time period. In order for the system to be able to run in real time it would need to be able to efficiently break up the windowed sequence into the n-grams and perform the classification analysis on them. After this system is implemented in a live operating system we could validate the results given here. Additionally, we would be able to understand the limits of such a system, for example how do we trick it into thinking an attack is normal behavior, and make improvements to our original classification model. ¹

7 Acknowledgments

Firstly, I would like to thank my advisor, Professor Aaron Cass, as well as the Union College Computer Science department for assisting me along my journey throughout the completion of my thesis project as well as during my time in the department. There were trying and difficult times but in the end I always felt as if I were home while working with the department. Additionally, I would like to thank my parents Bill and Anita Doyle for supporting me in my life to let me reach where I am today, as well as all my family and friends throughout my life here at Union College.

¹Access to this project and dataset is given under the MIT License agreement. <https://github.com/doylew/detectionsc>

References

- [1] J. Hu “*Host-based anomaly intrusion detection*”, in *Handbook of Information and Communication Security* P. Stavroulakis and M. Stamp, Eds. Springer Berlin Heidelberg, 2010, pp. 235-255
- [2] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T.A. Longstaff, “*A sense of self for unix processes,*” in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on.* IEEE, 1996, pp. 120-128
- [3] [Online]. Available: <http://linux.die.net/man/8/auditd>
- [4] [Online] Available: <https://www.python.org/>
- [5] [Online] Available: <http://scikit-learn.org/stable/>
- [6] G. Creech and J. Hu *Generation of a new IDS test dataset: Time to retire the KDD collection.* In *Wireless Communications and Networking Conference (WCNC), 2013, IEEE*, pages 4487-4492 2013.
- [7] M. Xie, and J. Hu *Evaluation host-based anomaly detection systems: a preliminary analysis of ADFA, LD.* The 6th International Congress on Image and Signal Processing (CISP 2013), 16-18 December 2013, Hangzhou, China pp. 1711-1716 DOI: 10.1109/CISP.2013.6743952 2013.
- [8] G. Creech and J. Hu *A Semantic Approach to Host-based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns.* *Computers, IEEE Transactions on*, PP(99):11. 2013.
- [9] M. Xie, J. Hu, X. Yu, E. Chang, *Evaluating Host-Based Anomaly Detection Systems: Application of the Frequency-Based Algorithms to ADFA-LD*, *Network and System Security*, Springer, pp. 542-549
- [10] G. Creech, *Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks*, *Doctoral Thesis, University of New South Wales*
- [11] M. Xie, J. Hu, J. Slay, *Evaluating Host-based Anomaly Detection Systems: Application of the One-class SVM Algorithm to ADFA-LD*, 2014, 11th International Conference on Fuzzy Systems and Knowledge Discovery

- [12] M. Ganapathiraju, V. Manoharan, J. Klein-Seetharaman, *BLMT*, 2004, *Applied Bioinformatics*, Volume 3, Issue 2-3, pp 193-200
- [13] Z. Su, Q. Yang, Y. Lu, H. Zhang, *WhatNext: a prediction system for Web requests using n-gram sequence models*, 2000, *Web Information System Engineering*, 2000. Proceedings of the First International Conference, pp 214-221 vol. 1
- [14] V. Jaiganesh, S. Mangayarkarasi, P. Sumathi, *Intrusion Detection Systems: A Survey and Analysis of classification Techniques*, 2013, *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, issue 4
- [15] G. Vigna, R. A. Kemmerer, *NetSTAT: A Network-based Intrusion Detection System*, 1999, *Journal of Computer Security*, pp 37-71 vol. 7
- [16] M. S. Hoque, A. Mukit, A. N. Bikas, *An Implementation of Intrusion Detection System using Genetic Algorithm*, 2012, *International Journal of Network Security and Its Applications*, vol. 4 no.2
- [17] H. Holm, *Signature Based Intrusion Detection for Zero-Day Attacks: (Not) A Closed Chapter?*, 2014, 47th Hawaii International Conference on System Science
- [18] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, *Anomaly-based network intrusion detection: Techniques, systems and challenges*, 2009, *Computers and Security*, vol. 28, no. 1-2, pp. 18-29