Intrusion Detection: Machine Learning Techniques for Software Defined Networks

Jacob Rodriguez

A Thesis Submitted to the Graduate Faculty of

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Master of Science in Cybersecurity

School of Computing

August 2023

Thesis Approval Form

GRAND VALLEY STATE UNIVERSITY

The signatories of the committee below indicate that they have read and approved the thesis of Jacob Rodriguez in partial fulfillment of the requirements for the degree of Master of Science in Cybersecurity.

8/4/2023
Andrew Kalafut, Thesis committee chair — Date

Xinli Wang
___ _____08/04/2023_
Xinli Wang, Committee member — Date

_____08/04/2023_
Byron Devries, Committee member — Date

Accepted and approved on behalf of the Padnos College of Engineering and Computing

Dean of the College

August 7, 2023

Date

Accepted and approved on behalf of the Graduate Faculty

Dean of The Graduate School

8/24/2023

Date

Table of Contents

Abstract

In recent years, software defined networking (SDN) has gained popularity as a novel approach towards network management and architecture. Compared to traditional network architectures, this software-based approach offers greater flexibility, programmability, and automation. However, despite the advantages of this system, there still remains the possibility that it could be compromised. As we continue to explore new approaches to network management, we must also develop new ways of protecting those systems from threats. Throughout this paper, I will describe and test a network intrusion detection system (NIDS), and how it can be implemented within a software defined network. This system will utilize machine learning techniques to discern between normal and malicious network traffic. The datasets that will be used for training and testing these machine learning methods include the UNR-IDD dataset, and the NSL-KDD dataset. The UNR-IDD dataset was created by researchers at the University of Nevada, Reno, and is intended to provide a wide range of samples and scenarios for machine learning-based intrusion detection systems. The NSL-KDD dataset is a newer version of the KDD '99 dataset, and is used as an effective benchmark for helping researchers compare various intrusion detection methods. Feature selection techniques will be performed during the testing phase to ensure the best features are used when performing analysis. In doing so, we'll be able to extract the best results possible from the experiments to determine the accuracy and effectiveness of the IDS.

Chapter 1: Introduction

Computer security is an essential part of ensuring the smooth operation of today's computer systems. It is also essential for protecting sensitive information, maintaining privacy, and preventing cyberattacks. With various forms of cybercrime on the rise, failure to implement proper security measures could lead to significant financial losses, organizational damage, or even legal consequences. Due to the complexity and variety of modern computer systems, there exists a number of ways to implement security measures both efficiently and effectively. One method of protecting computer systems from threats is intrusion detection.

Intrusion detection is the process of monitoring computer networks or systems in order to detect unauthorized access, or other malicious activity. It involves the use of software tools and techniques to identify potential security breaches, and alert security personnel to take appropriate action. The goal of intrusion detection is to identify attacks before they can cause significant damage. By detecting and responding to potential security breaches in a timely manner, organizations can minimize the impact of cyberattacks as well as protect their assets. There are many ways in which intrusion detection can be performed, but the one that this paper will focus on is a system that utilizes machine learning techniques.

Machine learning is a type of artificial intelligence which uses models and algorithms to allow computers to learn from data and make predictions or decisions based on that data, all without being explicitly programmed. These programs enable the machines to "learn" through experience in order to improve their performance on selective tasks. By training and testing machine learning models based on a particular problem or dataset, they can be used in a wide variety of applications, such as image recognition, recommendation systems, spam classification,

and even intrusion detection. However, creating a machine learning-based intrusion detection system requires knowledge of both cybersecurity as well as machine learning techniques.

The goal of this research project is to test and evaluate various machine learning-based techniques to create an efficient and reliable intrusion detection system that can be implemented into and used in conjunction with SDN. To accomplish this, I will perform my own experiments to determine what machine learning techniques are most effective at distinguishing between normal and malicious internet traffic. This system will also be compared with the structure and performance of similar approaches to the same issue. I've chosen this subject because, as a cybersecurity student, I've always found machine learning to be quite fascinating. I was introduced and became familiar with the main concepts of machine learning during a prior internship, and have also rather enjoyed classes I've taken in which machine learning techniques were often utilized. Furthermore, I've chosen to go with a machine learning-based system as opposed to a traditional signature-based one because it offers more benefits. With the emergence of newer and more modern methods of cybercrime, it is becoming more of a necessity that we also find new and more effective methods of threat detection. According to [1], traditional signature-based techniques are only able to detect known attacks. They are not as adept at detecting the presence of new/unseen attacks [18] - whereas a machine learning-based IDS has a greater ability to detect unknown attacks, as well as adapt to changing network conditions. Because of this, ID systems which employ machine learning techniques present a more novel solution to threat detection - one that is growing in both importance and popularity.

For this project, I'll be experimenting with a variety of machine learning techniques and algorithms, as well as taking inspiration from other researchers who have tackled the same problem, and come up with their own solutions. Through extensive experimentation, I plan on

finding a set of algorithms and techniques that will produce the best results possible. In other words, when it comes to distinguishing between normal and malicious traffic, the goal will be to do so with as much accuracy as possible.

The remainder of this paper will be laid out as follows: similar and related works of this project will be included in Background Information and Related Work, which includes other proposed solutions for developing intrusion detection systems which also employ machine learning techniques. Also covered in this section will be an overview of SDN architecture, as it pertains to the implementation of this system within an SDN environment. In the next chapter, Methods & Procedures, the methods for conducting my own experiments will be detailed, applying machine learning algorithms and techniques in order to find a solution that is both accurate and reliable. The results of these experiments will be presented in the following chapter titled Results. In the section titled Discussion, a review of the results will be further explained as well as the implications for the IDS as a viable solution. In the final chapter, a summary of the work done so far will be discussed, as well as the potential for future work involving the use of different (or additional) methods. A list of works cited throughout the paper will also be presented at the end.

All cited sources and references listed in this paper come from scholarly articles and published conferences. The datasets used, UNR-IDD [2] and NSL-KDD [4], both come from Kaggle.com. The former was created by Ph.D. Candidate Tapadhir Das, who works in the Department of Computer Science and Engineering at the University of Nevada, Reno. He, along with other researchers, worked to create a dataset that addressed a few of the flaws found in many of today's current datasets, such as the inadequate modeling of tail classes, dependency on flow level statistics, and the issue of missing or incomplete records [3]. By primarily using

network port statistics (which refer to observed port metrics recorded in switches/router ports), it can provide a better analysis of network flows from the port level - where decisions are made - as opposed to the flow level [3]. By prioritizing the utilization of network port statistics, potential intrusions can be identified quicker, and new insights can be provided to the questions regarding intrusion detection. The NSL-KDD dataset is a newer version of the KDD '99 dataset, and is used as an effective benchmark for helping researchers compare various intrusion detection methods. These datasets provide us with realistic data, in addition to a diverse set of attack types which will test both the system's accuracy and reliability.

Chapter 2: Background Information and Related Work

There are a number of works that have been done to test the efficiency and reliability of machine learning-based methods for intrusion detection. One of these works is [14], where a survey is taken of various machine learning methods and approaches used to develop intrusion detection systems within an SDN environment. Their research details both machine learning and deep learning approaches, as well as SDN architecture and its applications. Another of these works is [5], where researchers evaluate various machine learning algorithms to determine which of them will give the best results for their IDS. This IDS is then deployed over SDN, and employs the NSL-KDD dataset to train data to be able to distinguish and identify various types of attacks, using XGBoost as its driving algorithm. In [1], various machine learning techniques are evaluated to determine which of them is appropriate for flow-based intrusion detection. Through their results, they concluded that tree-based machine learning techniques held better classification rates, and required lower execution times than other, more complex algorithms [1]. [6] incorporates tree-based methods into a two-level IDS. Their proposed IDS uses an anomaly detection module in conjunction with deep packet inspection to determine whether a packet is normal or malicious [6].

The methods and techniques used to create and evaluate this IDS may use some of these works as a reference, such as feature selection methods, and various algorithms used during the testing phase. The hope is that by using methods that have been tested and proven effective for machine learning-based intrusion detection, this system can be both highly accurate and reliable. Though, in order to better understand the potential of this system within an SDN environment, let’s take a closer look at what SDN is and how it works.

SDN Overview

Software defined networking (SDN) is an approach to network architecture that enables the network to be centrally controlled, or "programmed", using software applications [7]. It is also an efficient and dynamic way of configuring the network in order to improve performance and monitoring, making it more akin to cloud computing than a traditionally managed network. Essentially, SDN provides a framework which separates a network's control plane from its data plane – a solution that proves to resolve several issues. In a typical network infrastructure lies the data plane and the control plane. The data plane is concerned with local traffic, determining how datagrams arrive in, and are forwarded out of routers according to decisions made by the control plane [8]. Whereas the control plane handles network-wide logic, determining how datagrams are routed along the end-to-end path from the source host to the destination host. The typical approaches to implementing the control plane are through traditional routing algorithms (implemented in the routers), or through software defined networking.

In SDN, network resources are managed by a logically centralized controller [9], which can compute and install forwarding tables in the routers. This controller has a global view of the network and its devices, and it can monitor and collect network configuration data in real time [9]. This feature provides a solution to a few of the issues a network faces when it comes to traditional networking architecture. One of these issues is that each router must implement its own data and control plane. This means that if a change were to occur in one router's control plane, such as updated configurations or rules, then every other router in the network would have to be made aware of that change. This is something that would not be easily accomplished, especially for larger scale networks. Requiring each router to maintain information about every other router in the network isn't feasible and would pose enormous memory constraints. Another

issue that presents itself is that router configurations are getting more complex every day. This means that now more than ever, having to focus on compatibility management for each router is a huge hassle, and only gets worse with larger scale networks. Therefore, by having the logic for the control plane taken out of each router and contained within a centralized system, SDN provides a way for the network to be easily scalable without having to worry about issues like compatibility management, memory constraints or information sharing. It's reasons like these that modern networks are moving away from a distributed control plane in favor of a more centralized one.

SDN Architecture

To better understand how applications such as an IDS can be implemented within software defined networks, we'll take a closer look at SDN architecture. Within typical SDN architecture are three main layers: the application layer, control layer, and infrastructure layer [10]. The application layer contains software applications that run on either physical or virtual hosts. The purpose of these applications can vary according to the needs of the network, such as having a flow optimizer, load balancer, or providing services like access control or intrusion detection [10]. The control layer is where the network controllers reside and is responsible for implementing the logic and rules passed from the application layer to the infrastructure layer [10]. This layer acts as a sort of network operating system for SDN, as it can have topology and flow table managers to maintain the network's state information, and dynamically allocate network resources, respectively [10]. This layer is also responsible for managing traffic data and regulating actions by either establishing or denying network flows [5]. Lastly, the infrastructure layer is made up of hardware and software components, and is tasked with simply forwarding data packets. The hardware components of this layer include devices such as programmable switches and routers, while the software components such as OpenFlow switches are able to interface with the hardware [5].

Let's now discern just how this system can be implemented within SDN architecture. As stated previously, the centralized controller has a global view of the network, and is able to collect and monitor network traffic. This is an extremely useful feature, as it can promote and enable numerous applications of machine learning algorithms. Having the IDS placed within SDN's application layer is just one way to take advantage of this feature. By implementing the IDS in this way, any traffic that flows through the central server/controller is scanned and

analyzed, determining whether it can be classified as normal or malicious. These results are then compared against the rules of the network and its configurations, which may be housed inside a rules management system also contained within the application layer. These rules would determine the course of action to be taken on the scanned data - to either accept or reject/block it. This information is then sent to the control layer, where controllers can instruct the switches (in the infrastructure layer) on how to handle the data. A controller that could ideally be used in this system would be POX, as it's compatible with Linux, MacOS, and Windows platforms [11]. POX is also a controller that's helpful in aiding and discovering paths and architectures [12], making it useful for applications involving machine learning techniques. If the IDS does in fact detect a malicious presence, an alert can be sent to the appropriate administrators, or to a network security management team. OpenFlow switches and/or various other software tools within the infrastructure layer would be responsible for conveying updates and information to the rest of the devices on the network. Should any malicious traffic be detected, then according to the rules of the network (or rules management system), the central controller will update the configurations of each router along the network, telling them to block any associated traffic. This is where the architecture of SDN can be made to showcase its practicality.

In a traditionally managed network, an IDS may be able to stop an attacker from gaining access to a central server initially, but all of the routers in that network would have to be updated one by one in order to be properly reconfigured. The time that it takes to make the necessary changes may vary, but in that time, it's possible that an attacker could persist through devices on the network which still remain vulnerable. Until every router in the network has been made aware of the threat, each of them could be considered a potential attack vector. This issue is one that is easily solved through SDN's architecture, which allows the central controller to update the

rules and configurations of each router in real time. As with any security mechanism, it's crucial that swift action is taken so that organizations can minimize disruptions in their business processes, limit their financial losses, and implement necessary safeguards to prevent similar attacks from occurring in the future. Now let's take a closer look at exactly how this IDS will work.

Chapter 3: Methods & Procedures

In this section, a variety of machine learning algorithms will be tested using two chosen sets of data. These datasets were both created for the purpose of assisting researchers by providing them with valuable data in order to test and compare various intrusion detection methods and solutions. The datasets that will be used for training and testing these machine learning methods include the UNR-IDD and NSL-KDD datasets. The UNR-IDD dataset was created by researchers at the University of Nevada, Reno, and is intended to provide a wide range of samples and scenarios for machine learning-based intrusion detection systems. The NSL-KDD dataset is a newer version of the KDD '99 dataset, and is used as an effective benchmark for helping researchers compare various intrusion detection methods. As previously mentioned, the UNR-IDD dataset is primarily composed of network port statistics, while the NSL-KDD dataset contains at least 20 different attack types, offering a wide range of data to draw from, as well as providing a level of variety that can prove useful for a machine learning model. Together, these datasets provide great material to work with and draw from in order to create a robust model that can accurately predict and classify traffic from among vast amounts of data.

The platform that was chosen to use to create the IDS is Google Colab, as I'm familiar with it and have used it previously to conduct similar assessments and projects related to machine learning. Another thing to note is that before the datasets could be used in testing, it was crucial that a bit of preprocessing needed to be performed on them. Both of the datasets contained features with categorical variables, which would pose an issue during testing and analysis. Therefore, any categorical features (columns) were translated into numerical values that could be understood and used by the system. With this, the dataset is able to maintain its integrity

without sacrificing additional data or features that could possibly cause interference during testing.

Feature Selection

The use of feature selection was also used after a dataset was uploaded and imported into a pandas dataframe (the "pandas" package was imported to be able to better manipulate the data). After the data has been separated into training and testing sets, it is then run through a feature selector which finds and determines the best features to use for analysis. This process is useful in finding certain features that are weighed very little in terms of the effect they have on the final results, such as columns that contain just a single value. Removing these features would then be a way of cleaning up the data and refining it for later use. It's also a way of extracting the features within the data that will yield the greatest results. For this case, SelectKBest was used as the selector, as it allows for the specification of the number of desired features (in this case, k=20). The selector works by calculating a "score" for each of the variables (features) based on univariate statistical analysis [13]. The top 20 features with the highest score are then determined to be the best, or most valuable features in the dataset. This process was conducted on each of the datasets separately, and with each time, the dataset needed to be reuploaded after making the necessary changes. At the end of the feature selection process, the UNR-IDD dataset contains 21 features, while the NSL-KDD dataset contains 20. Once the preparatory work for each of the datasets was complete, the testing phase could begin.

Classification Algorithms

During this phase, the testing of various machine learning algorithms was conducted. The model was tested to see how well it performs, utilizing 6 different algorithms to determine which one will yield the best scores. These classifiers include two ensemble algorithms, two that are tree-based, and two that are linear. The ensemble algorithms, which include RandomForestClassifier and AdaBoostClassifier, were chosen because ensemble-based methods can be used for classification, regression, or anomaly detection. This makes them popular choices for those who wish to utilize them for machine learning analysis. Furthermore, [14] and [15] both describe using the RandomForest algorithm as a classifier for their datasets ([15] also uses AdaBoost as one of its classification algorithms). Additionally, AdaBoost is mentioned in [16] as being used in a system which resulted in a lower false alarm rate, higher detection rates, and was computationally faster than other published results. Tree-based algorithms are also very useful when it comes to machine learning analysis, as they are very adept at performing classification and regression tasks. Those which comprise the tree-based algorithms are DecisionTreeClassifier and ExtraTreeClassifier. DecisionTree is a well known machine learning algorithm that is frequently used for both classification and regression tasks, as described in both [14] and [15]. It is also an algorithm that can be used for feature selection, which makes it incredibly versatile as well.

Depending on their parameters, some linear classifiers can fall into multiple categories, which is part of why they were also included in this group. Perceptron and RidgeClassifier are the linear models that were chosen for this set of algorithms. In the event that either could outperform any of the others during the final phase of testing, it would have been interesting to see how changing parameters would affect their results (it wasn't expected that they would yield

higher results than the others, this is evident in the Results section). After all, a variety of classifiers was chosen deliberately for the purpose of seeing how they compare against one another, as well as determining which among them produces the best results. If it were the case that the linear classifiers outperform the others, it would be something that could warrant further testing and investigation. Nevertheless, the goal of this testing phase will be to determine which algorithm from among the group produces the best results. It should also be noted that when determining this, the scores from both datasets are taken into account.

Testing the Model

The testing phase was conducted in four steps. For the first step, the UNR-IDD dataset is uploaded and run against each algorithm one at a time, while the results are captured and recorded. This process is then repeated for the NSL-KDD dataset. The results for this first step showcase each classifier's accuracy, which will henceforth be referred to as the ratio of correctly classified records with that of all records in the dataset [16]. Simply put, accuracy estimates how many records within the dataset are correctly identified, regardless of their label (normal or attack). This metric can provide a good baseline for the affinity each algorithm has when it comes to attack classification, as well as a clear picture of which algorithms are more adept at distinguishing potential attacks from normal traffic/data. Upon completing the first step, however, the number of working algorithms was reduced to half by removing the lower scoring classifiers from each category. This way, the next step will begin its phase with 3 algorithms instead of 6 (one ensemble algorithm, one that's tree-based, and one that's linear). By narrowing down the options as testing progresses, it can be observed which methods will be most useful and best equipped for machine learning-based intrusion detection. However, accuracy alone is not enough to verify that each record is truly being classified correctly.

In the first step, the model's accuracy was recorded using the various algorithms listed previously. For the second step, the model's precision will be evaluated using the remaining classifiers. Precision is described as the measure of true positives divided by the number of true positives plus the number of false positives ($\frac{True\ Positive}{True\ Positive + False\ Positive}$) [16]. For further clarity, true positives refer to the number of records that are correctly classified as "Normal", or "Benign", while false positives refer to the number of normal records which are misclassified as

an attack. In other words, precision estimates the ratio of attacks that were correctly identified with that of all identified attacks in the dataset [16]. It is not enough to merely observe the model's accuracy, as the possibility still remains that it could simply be misclassifying data. One of the main things that can hinder the integrity of an IDS is when it classifies normal data as potentially malicious. This is something that nearly all kinds of IDS's encounter and have to deal with. In order to see that the model is in fact correctly classifying the data, its precision must be tested as well. As shown by the equation above, a low number of false positives strongly indicates that the data is in fact being correctly classified. Displaying such a high precision rate suggests that a machine learning model may not only be accurate, but reliable as well. Now, the procedure for this second step will be much like that of the first. The UNR-IDD dataset is uploaded and run against each algorithm one at a time, this time calculating precision instead of accuracy. The results are captured and recorded, and the process is then repeated for the NSL-KDD dataset. When this step is completed, the classifier which scores the lowest is dropped from the group, leaving just two to work with for the remainder of the testing phase. With the completion of the first two steps, the classifiers that are left are determined so far to be the most accurate and most precise out of the original starting group. However, there's still some work to be done before finally determining which among these classifiers is best suited to be the driving algorithm for the IDS.

For the third step, the system's recall is tested using the remaining two algorithms. Recall (also called the True Positive Rate) often works in concert with precision in verifying that the system is indeed correctly classifying data in accordance with its true label (either "normal" or "attack"). Similar to how precision is calculated, recall measures the number of true positives divided by the sum of true positives plus false negatives ($\frac{True\ Positive}{True\ Positive + False\ Negative}$) [16]. Here,

false negatives refer to the number of attack records that are misclassified as normal. In other words, recall estimates the ratio of correctly classified attacks to all attacks in the dataset [16]. To clarify, precision measures the *correctness* of attack classification, while recall measures the *completeness* of attack classification. Using these two evaluation metrics in conjunction can provide great insight into the performance and effectiveness of the system. By doing so, it can ensure that the system is not only correctly classifying attacks, but is also able to catch nearly all recorded attacks. A system which displays high recall insinuates that it shows an incredible aptitude for attack classification. Moving forward, the procedure for this third step will be much like that of the previous two. The UNR-IDD dataset is uploaded and run against each algorithm one at a time, this time calculating for recall. The results are captured and recorded, and the process is then repeated for the NSL-KDD dataset. With the completion of these three steps, we can then continue on with the final phase of testing.

For the final step, the accuracy of the model will again be measured. This time, however, the classifiers will be further tuned by experimenting with different parameters. Some of these parameters include n_estimators, criterion, splitter, min_samples_split, min_samples_leaf, etc. All these parameters serve different (or similar) functions, and although some might be similar in purpose, they may all have the potential to impact the results in some way. Changing the values of these parameters, or using them in combination with each other can have either a positive or negative effect. Furthermore, this effect can be rather significant, or have little to no impact at all. At times when adding or altering certain parameters makes little to no difference, it may be appropriate to simply omit them. Some of these parameters are also already incorporated in the algorithm in a default state. This means that, while there's no visible indication, certain parameters have default values that the algorithm uses to accomplish its task. One example of

this is n_estimators. Values for this particular parameter can range from 10-100, however, some algorithms may use a default value of 100. This indicates that the algorithm uses "n_estimators=100" without it being explicitly specified. If one wants to determine whether this value carries any weight in terms of the observed results, then a new value must be specified as such: "n_estimators=50". Even if a small improvement is observed, this new value would be kept in place of the default one.

There were numerous parameters to experiment with, and working through the list of available options was somewhat tedious. However, the process went smoother than imagined, as in the end, not many changes were kept. Beginning first with one parameter, changing its variables/inputs, and acting according to the observed output, the question of whether that change would be permanent was answered. If the results improved by even a small margin, then the change would be kept, and other changes would continue by altering other parameters and their values, observing the output and repeating the process. If no change was observed in the results, or if there was a decrease in performance, then that variable was either omitted, or left in its default state. Using this process, a number of variables were tested from among the two remaining algorithms, running them both against the two datasets as before. If possible, the same alteration would be made in both algorithms so their results could be compared. In the end, both algorithms performed remarkably well, but as seen from the results, the one with the best overall score was RandomForest, and the testing phase comes to a close.

Chapter 4: Results

In this section, the results of the testing phase are displayed. The records are separated according to the stage of the testing process in which they were observed. Table 1 shows the results from the first stage, where the model's accuracy was observed using a variety of different algorithms. These include RandomForestClassifier, AdaBoostClassifier, DecisionTreeClassifier, ExtraTreeClassifier, Perceptron, and RidgeClassifier.

Algorithm	Random Forest	AdaBoost	Decision Tree	ExtraTree	Perceptron	Ridge
UNR-IDD	97.97	97.18	97.93	96.94	74.81	97.93
NSL-KDD	99.38	84.79	98.89	97.96	82.95	91.71

Table 1: Algorithm Accuracy

After the first round of testing, the various results were compared against each other. From the Table 1 results, it can be observed that most of the algorithms had little disparity between the scores from when they were tested using the UNR-IDD dataset, and when they were tested using the NSL-KDD dataset. Also, there are few outliers, as most values fall within a similar range. However, with linear algorithms such as Perceptron, it was predicted that the scores it produced would fall on the lower end of the spectrum, as they have appeared to do so. RidgeClassifier, however, seems to have performed much better than Perceptron despite them both being classified as linear algorithms. This plays into the manner in which these algorithms are separated to prepare for the next step. As previously mentioned, three of the algorithms will be removed before continuing with the next round of tests. This was done by removing the lower scoring classifier from each category. In the case of the linear classifiers, Perceptron was

removed, leaving RidgeClassifier to be placed against the next round of testing along with RandomForest and DecisionTree.

Algorithm	RandomForest	DecisionTree	Ridge
UNR-IDD	98.02	97.94	97.01
NSL-KDD	99.40	98.78	91.71

Table 2: Algorithm Precision

For this second stage, the precision of the model was put to the test using the three remaining algorithms. The purpose of this stage was to determine the false positive rate of the model, to see if there was any misclassification of data. If the current model were to display low precision scores, it would also mean that it shows a high rate of false positives. This would indicate that most of the data the model classifies as normal would actually be malicious, and would be an example of a model that is not well suited for classification tasks like intrusion detection. While the Table 1 results show that the model is highly accurate, it would not matter unless it was also very precise in its classification. However, Table 2 shows that the classifiers all displayed remarkable promise, as they had high precision scores. This suggests at least a large percentage of the data is in fact being correctly categorized. For the next step, the number of classifiers will again be reduced to narrow down the options for the IDS's driving algorithm. Again, this was done by removing the lowest scoring classifier from the group, which in this case was RidgeClassifier. The remaining two algorithms will continue to be used during the next two stages of testing.

Algorithm	RandomForestClassifier	DecisionTreeClassifier
UNR-IDD	99.92	98.86

NSL-KDD	99.36	98.85

Table 3: Algorithm Recall

In the third step, the model's recall, or true positive rate, was tested. High recall rates indicate that nearly all attacks within a dataset are being identified, and as Table 3 shows, both classifiers have done exceptionally well in that regard. This, combined with the results from step two, signifies that when it comes to attack classification, the model is able to catch nearly all recorded attacks, and does so with a high degree of accuracy. In other words, according to the results of the tests done so far, the model is able to correctly identify and distinguish normal data from attacks, with little chance of there being any misclassification (false positives/negatives). Furthermore, looking at the results of Tables 2 and 3, it can be seen that the RandomForestClassifier has a slightly higher affinity for attack classification than DecisionTreeClassifier. This is also evidenced by the initial readings for their overall accuracy showcased in Table 1. At this point in the testing phase, it has been made clear that these two classifiers are the ones most adept at detecting potential threats. However, with one more step to go, there is still work to be done before finally determining the most suitable option for this IDS's driving algorithm.

Algorithm	RandomForestClassifier	DecisionTreeClassifier
UNR-IDD	98.16	97.98
NSL-KDD	99.45	99.11

Table 4: Tuned Model

For this final step, the model's accuracy is once again put to the test. The goal this time will be to fine tune the model in order to achieve even better results if at all possible. This was done by experimenting with the various parameters for each algorithm, observing the output, and

acting according to the observed output. If changing a parameter produced any kind of improvement in the model's performance, then it was kept. If it detracted from the scores, or even if there was no change at all, then it was omitted. In regards to both algorithms, however, they share many of the same parameters, so virtually any changes made in one could be made in the other. Additionally, the effect of these changes were usually found to be the same in both as well - if there was no observable change after altering a parameter in one, it was likely that the other one would show no change as well. There were a few parameters that were unique to their respective algorithms, such as "n_estimators" for RandomForest (ensemble), and "splitter" for DecisionTree (tree-based), however, neither of them seemed to produce any significant changes. After many repeated attempts, it was found that the alteration which made the largest respectively positive difference was the "criterion" parameter. This parameter, which is accessible to both algorithms, acts as a function which measures the quality of a split, and has three possible values: "gini", "entropy", and "log_loss". By default, the algorithm takes "gini" as the value for this particular parameter, however, changing it to "entropy" yielded a small improvement in the model's accuracy scores. This is shown in Table 3, where the tuned model is run against both datasets once more, displaying the results for each algorithm. Though not much seems to have changed in regards to the model between the first test and the final one, it remains true that the tuned model produced slightly better results than the original. From this, we can conclude that a tuned model which uses RandomForest as its driving algorithm is both highly accurate and exceptionally precise.

Chapter 5: Discussion

Based on the results of the testing phase, it was determined that RandomForest was the best choice to serve as the driving algorithm for this IDS. As evidenced by other related works, it appears as though this algorithm is quite adept at classifying data for the purposes of intrusion detection. Comparing the results from the tests done in [15], where various supervised machine learning algorithms are run against the NSL-KDD dataset, RandomForest scored with an accuracy of 99.7% (AdaBoost also scored with an accuracy of 89.3%). Additionally, a literature review by [12] mentions a signature-based IDS which uses the RandomForest classifier on the CICIDS2017 database. The cross-validated score that was reached yielded an accuracy score of 99.713%. These scores closely reflect those of our own experiments, indicating that RandomForest may indeed have been the most optimal choice for this particular case. Furthermore, one of the works surveyed in [14] describes their use of RandomForest as a classifier for the NSL-KDD dataset. Their findings showed that their model held high accuracy scores as well as a low false alarm rate [14], very much like our own model. From this, we can conclude that this model is a potentially viable solution to machine learning-based intrusion detection.

Even so, this system still has yet to be fully implemented and tested within an SDN environment. Earlier in this paper, it was detailed how one could effectively implement this system within SDN architecture via the centralized controller. Though the main purpose of this project was to determine how one could use machine learning techniques to create an efficient and reliable intrusion detection system that could be used in conjunction with SDN, being able to test that system within a real-world SDN environment may have just been outside the scope of the initial design. Though the system does show great promise, extended testing inside a

simulated environment may prove useful in determining what shortcomings, if any, may exist within the system as it stands currently. That, however, could be something that can be explored in a future work of this project as a means of continued experimentation and verification.

Chapter 6: Conclusion

In this paper, we discussed what SDN is, how it works, and its importance in the world of computer networking. We also introduced a system which uses machine learning techniques and algorithms to perform intrusion detection, and discussed how that system could be effectively implemented within SDN architecture. Through testing and evaluation, it was concluded that the modified RandomForest classifier was the most optimal choice for the IDS's driving algorithm. Yielding high accuracy scores along with exceptional precision, this system boasts a proficient ability to correctly classify and distinguish normal data from potentially malicious attacks. Such high precision scores also indicate a low rate of false positives. This, along with the comparison of works referenced in the Discussion, further promotes the reliability of this system.

As with most systems, however, there may always be something to improve upon. As previously mentioned, the potential for future work on this project could lie in the post-implementation of this system within a software defined network to determine and discover any pitfalls it may have as a result of exposure to real-world data/scenarios. Understanding the system's behavior under diverse network conditions may ultimately contribute to its robustness and practical applicability. Other possibilities for future works involving this project may include additional research and testing on various preprocessing and feature selection techniques. By experimenting with these methods, the performance and efficiency of the system can be further explored. In conclusion, the continued research of these methods may contribute in advancing the understanding and applicability of this system within software defined networks.

Intrusion detection systems that use machine learning techniques are becoming increasingly popular in recent years, especially in the area of software defined networking. As

new technologies are adapted and discovered, more potentially viable solutions such as these will become evermore prevalent. For now at least, it appears as though this system has the hallmarks of a solid foundation for implementing machine learning-based intrusion detection for software defined networks.

References

1. Rodríguez, M., Alesanco, Á., Mehavilla, L., García, J. (2022). "Evaluation of Machine Learning Techniques for Traffic Flow-Based Intrusion Detection," *Sensors*, *22*(23), 9326. https://doi.org/10.3390/s22239326

2. Das, Tapadhir, "UNR-IDD Dataset." *Kaggle*, https://www.kaggle.com/datasets/tapadhirdas/unridd-intrusion-detection-dataset

3. Das, Tapadhir, "UNR-IDD Dataset." *Tapadhir Das*, https://www.tapadhirdas.com/unr-idd-dataset

4. Kiran, "NSL-KDD." *Kaggle*, https://www.kaggle.com/datasets/kiranmahesh/nslkdd

5. Alzahrani, A.O.; Alenazi, M.J.F. "Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks," *Future Internet*, 2021, *13*, 111. https://doi.org/10.3390/fi13050111

6. V. Vetriselvi, P.S. Shruti, and S. Abraham, "Two-Level Intrusion Detection System in SDN Using Machine Learning," *Lecture Notes in Electrical Engineering*, ICCCE 2018, vol. 500, pp. 449-461, https://doi.org/10.1007/978-981-13-0212-1_47

7. Y. Zhang, L. Cui, W. Wang, Y. Zhang, "A Survey on Software Defined Networking with Multiple Controllers," Journal of Network and Computer Applications, vol. 103, 2018, pp. 101-118, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2017.11.015

8. T. Hu, Z. Guo, P. Yi, T. Baker and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey," in *IEEE Access*, vol. 6, pp. 15980-15996, 2018, https://doi.org/10.1109/ACCESS.2018.2814738

9. J. Xie *et al*., "A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges," in *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393-430, Firstquarter 2019, https://doi.org/10.1109/COMST.2018.2866942

10. T. Li, J. Chen, and H. Fu, “Application Scenarios Based on SDN: An Overview,” *IOPScience*, Journal of Physics: Conference Series, 2019, vol. 1187, no. 5, https://doi.org/10.1088/1742-6596/1187/5/052067

11. L. Zhu *et al.*, “SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study,” *ACM Digital Library*, ACM Computing Surveys, 2020, vol. 53, no. 6, Article no. 133, (November 2021), pp. 1-40, https://doi.org/10.1145/3421764

12. A. Bhardwaj, *et al.,* “Network Intrusion Detection in Software Defined Networking with Self-Organized Constraint-Based Intelligent Learning Framework,” *ScienceDirect*, vol. 24, ISSN 2665-9174, 2022, https://doi.org/10.1016/j.measen.2022.100580

13. T. Desyani, A. Saifudin, Y. Yulianti, “Feature Selection Based on Naive Bayes for Caesarean Section Prediction,” *IOPScience*, IOP Conference Series: Materials Science and Engineering, 2020, vol. 879, https://doi.org/10.1088/1757-899X/879/1/012091

14. N. Sultana, N. Chilamkurti, W. Peng, R. Alhadad, “Survey on SDN Based Network Intrusion Detection System Using Machine Learning Approaches,” *Peer-to-Peer Netw. Appl.* 12, 493–501 (2019). https://doi.org/10.1007/s12083-017-0630-0

15. R. R. Devi and M. Abualkibash, “Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper,” *SSRN*, International Journal of Computer Science & Information Technology, vol. 11, no. 3, June 2019, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3428211

16. R. Vinayakumar, *et al.*, "Deep Learning Approach for Intelligent Intrusion Detection System," *IEEE Xplore*, vol. 7, pp. 41525-41550, 2019, https://doi.org/10.1109/ACCESS.2019.2895334

17. M. Hasan, *et al.*, “Attack and Anomaly Detection in IOT Sensors in IOT Sites Using Machine Learning Approaches,” *ScienceDirect*, 2019, vol. 7, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2019.100059

18. I. H. Sarker, *et al*., “IntruDTree: A Machine Learning Based Cyber Security Intrusion Detection Model,” *Symmetry*, *12*(5), 754, https://doi.org/10.3390/sym12050754