Applying machine learning to create a robust intrusion detection system

Susmita Rai - 908928

Abstract

Intrusion detection systems are fundamental for defending against cyber attacks. This project will aim to create a robust intrusion detection system that can detect multiple different attacks with a high true positives and low false postives using machine learning.

Contents

1 Project definition

1.1 Introduction

With the rapid expansion of the Internet, it has become an essential part of our lives with over half the population connected [1]. However, this results in an increasingly complex and fragile network. Many systems are left vulnerable, waiting to be exploited. By 2021, it is predicted that cost of cyber-attacks will reach $6 trillion [2]. The importance of a good and secure security system is far too crucial.

"Offensive cyber capabilities are developing more rapidly than our ability to deal with hostile incidents" [3]. Attacks are becoming smarter, polymorphic viruses and obscured malwares are passing through current systems. Over a third of organizations believe that the threats they are facing cannot be blocked by their anti-virus [2]. Due to our rapid growth, we have left many openings for an attack, one of them is through the network. Our need for constantly being connected is causing a major gap in security. In 2017, 8 different network attacks dominated the market [4].

1. Browser attacks - malicious users target vulnerable websites to infect, infecting new genuine users.
2. Brute force attack - attempting to guess your way through to the system.
3. Denial of service (DoS) or Distributed Denial of Service (DDoS) – flooding a service by creating many requests in order to slow or crash the system.
4. Worm attacks – self propagating program that spreads through local system through exploitable vulnerabilities.
5. Malware attacks – programs that can take many forms, however their purpose is always malicious.
6. Web attacks – exploiting vulnerabilities found in the website such as SQL injection.
7. Scan attacks – indirect attack to gain knowledge of any vulnerabilities that exist such as an open port.

8. Other attacks – attacks that were out of scope, such as physically attempting to steal device.

Fortunately, methods such as Intrusion detection system (IDS) exist to deter most of these attacks. IDS constantly scan the network for any anomalous activity in the network. Some are even capable of stopping the attack completely rather than just alerting the user.

However, IDS face many issues such as explaining what an anomaly is in the first place. Robustness and accuracy also come into question. How often does an IDS system report false negatives or how many different types of malwares can they detect?

By using machine learning may be possible to overcome these problems because of its ability to learn patterns and understand different classifications. As malware evolve, the system can also be retrained to learn new patterns of attack. Because of this, I aim to create a robust system that is capable of detecting malwares using using machine learning.

1.2 Project aims

The project aims to create a system that can detect malware on a network by incorporating machine learning. The system should be able to classify a wide range of malwares accurately whilst having a low false positive rate.

There are several different ways accomplish this. Below is list of steps I will take to approach this problem with several extensions that I would like to do, given time constraints and my own ability.

1. Develop a system capable of classifying different malwares.

   This would be the foundation for all solutions to other problems. It would attempt to create an IDS system using machine learning techniques to detect anomalies, in this case malwares. Also, it aids to not only understand the dataset but the behaviour of malwares as well.

2. Create synthetic malware attack patterns

   This depends heavily on the results of step 1. If the system can detect malwares, then it can also learn the pattern of what makes a malware. This would allow creation of synthetic malware attack patterns. It is also possible to retrain the classifier in step 1 to be “smarter” against

fake synthetic attacks which hopefully reduces the chances of a false positive, creating a smarter IDS system.

2.1. Replicate environment and create my own attacks

Another possibility I would like to do would be to create actual malware attacks, not just synthetic. Replicating the environment, the dataset was collected in, the classifier could be tested on a live network rather than just on a pre-collected dataset.

3. Test robustness of available IDS with synthetic data.

The final step would be to test how well IDS systems work. If synthetic malware data is realistic enough, it would be able to fool IDS systems into thinking an attack has happened. It is then easy to test an IDS system's reaction to such attacks. If the system flags such an attack, it shows that there is room for improvement.

3.1. Create genetic adversarial neural networks (GANNs) to create even more realistic synthetic malware data.

With the usage of GANNs, more realistic synthetic malware data can be generated which in turn tests IDS systems. An almost infinite training loop could be created to train IDS systems to learn the correct patterns of malware which should make the system more robust.

2 Project management

2.1 Methodology

For this project, I have chosen to use Scrum agile methodology because of its iterative nature. Scrum framework follows one simple rule, that requirements can change, or otherwise known as requirements volatility [5]. Due to its flexibility, Scrum is ideally designed for a team of ten or fewer [6] and since I will be working alone, some of its fundamentals will be altered

Usually a daily Scrum takes place at the start of the day where the development team discuss the events of yesterday, however since I will be working alone, no daily Scrum will take place. Instead, a meeting with all other undergraduates under the same supervisor will take place every fortnight. This way everyone can discuss the progress of their respective projects.

Whereas for sprint meetings, I will create a meeting with my supervisor every fortnight, which will be the length of one sprint. Instead of having a different meeting for planning, review and retrospective, it will all be merged into one. By doing this, I will still be able to receive some feedback and talk about issues that are/may cause problems.

I will also be responsible for creating a product and sprint backlog. Tasks will be broken down into separate categories, such as "todo" or "in-progress". Each task will also be assigned a priority ranging from "must haves" to "would be nice". I will be using GitHub project board [7] or some alternative that is similar in order to create the task board. Having an online task board allows easy access wherever I am.

This methodology also ties in well with the structure for machine learning projects. Before a model can be created, data will have to be processed. In this step, data must be prepared and cleaned such that it can be fed into a model. This is then followed by model training and finally model testing and evaluation. Figure 1 displays the life cycle of machine learning projects [8]. Depending on how the data is processed, the results will differ for the model. The algorithms used for the model also highly influences the outcome. This ends up being an iterative process since at each sprint, the goal would be to create the best model possible.

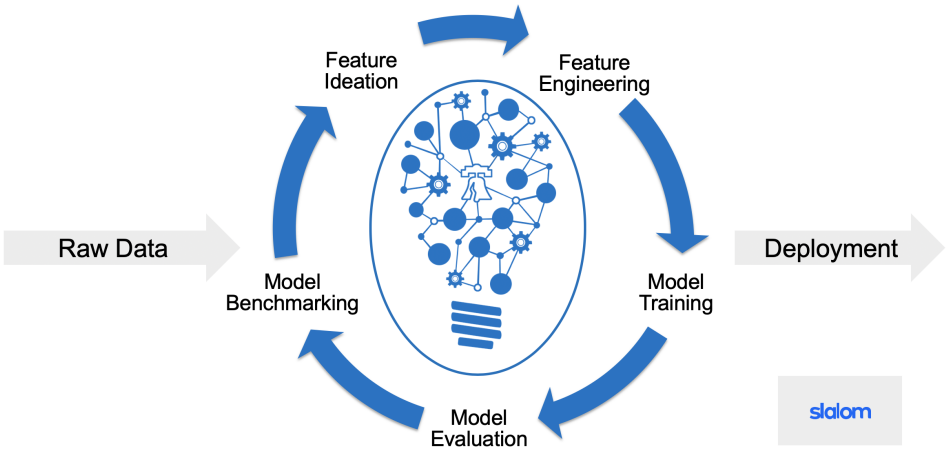

Figure 1: Machine learning life cycle

2.2 Risk analysis

1. Time management

   Due to my lack of productivity, unpredicted events or underestimation of the workload, the project may not be completed in time. However, because of my chosen software life cycle, I will be able to create a plan of tasks with their priority and effort levels recorded at the start of every sprint which should help guide the workflow. Also, the roadmap laid out by the Gantt chart contains a timeline which I will follow.

2. Hardware access

   This project deals with big data which requires extensive hardware. Lack of hardware can hinder progress as methods will have to be optimized which will require more time. For example, trying to just load the data will have to be split into various batches as there won't be enough RAM. Training models can also take significantly longer as a lot of processing power will be needed. Hopefully, this should not be an issue as the university has many different devices suited for this project.

3. Lack of domain knowledge

   This project deals with two separate issues, machine learning and security. For this project to succeed, it requires high domain knowledge in both fields which I may not have. However, by planning properly,

tasks can be broken down into simpler components. Also, guidance is available from experts in the field.

3.1. Dataset

Every machine learning project requires a dataset and since it is far too time consuming to produce my own [9], I will have to select an existing one. I may end up choosing a wrong dataset that is not fit for this project. Before I make my decision, I will ask for feedback from experts in the field.

3.2. Generalisable

A lot of IDS systems created using machine learning techniques seem to not be generalisable to other networks or system [9]. To avoid this, the model should not overfit the dataset whilst ensuring it isn't underfitting. Creating my own hack and applying it to the model could also make it more generalisable.

3.3. Malware – features of importance

Understanding what features are important will be crucial therefore my lack of knowledge can be a risk. However there are many guides on malwares which I will have to read. If necessary, ask for expert advice as well.

3.4. Machine learning - applying right methods

Machine learning has a wide variety of different algorithms to choose from and I may end up choosing the wrong ones. Also, I could preprocess the data incorrectly which could lead to the model learning biased or incorrect information. This will require a lot of background reading.

2.3 Gantt chart

The Gantt chart is split into different sections however since the dates have not been finalised, a lot of end-dates have been estimated.

It is also important to note that dataset selection (which I will expand under section 3.3.1) comes before the intial document. This is because the entire project depends on the dataset therefore, it had to be chosen before the project started.

The implementation section is split into two parts. There is an overall view of what I will be doing and it is also split into sprints to show specifically

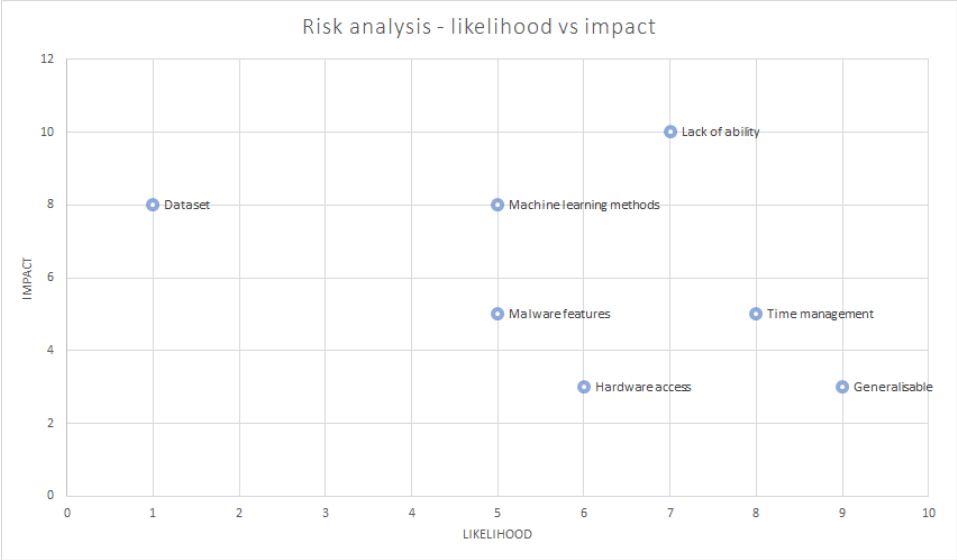

Figure 2: Likelihood vs Impact

what I intend to do each sprint. The purple tasks are extensions that I would like to do if there is any time. There is also a block of sprints allocated incase I misjudged any timings.

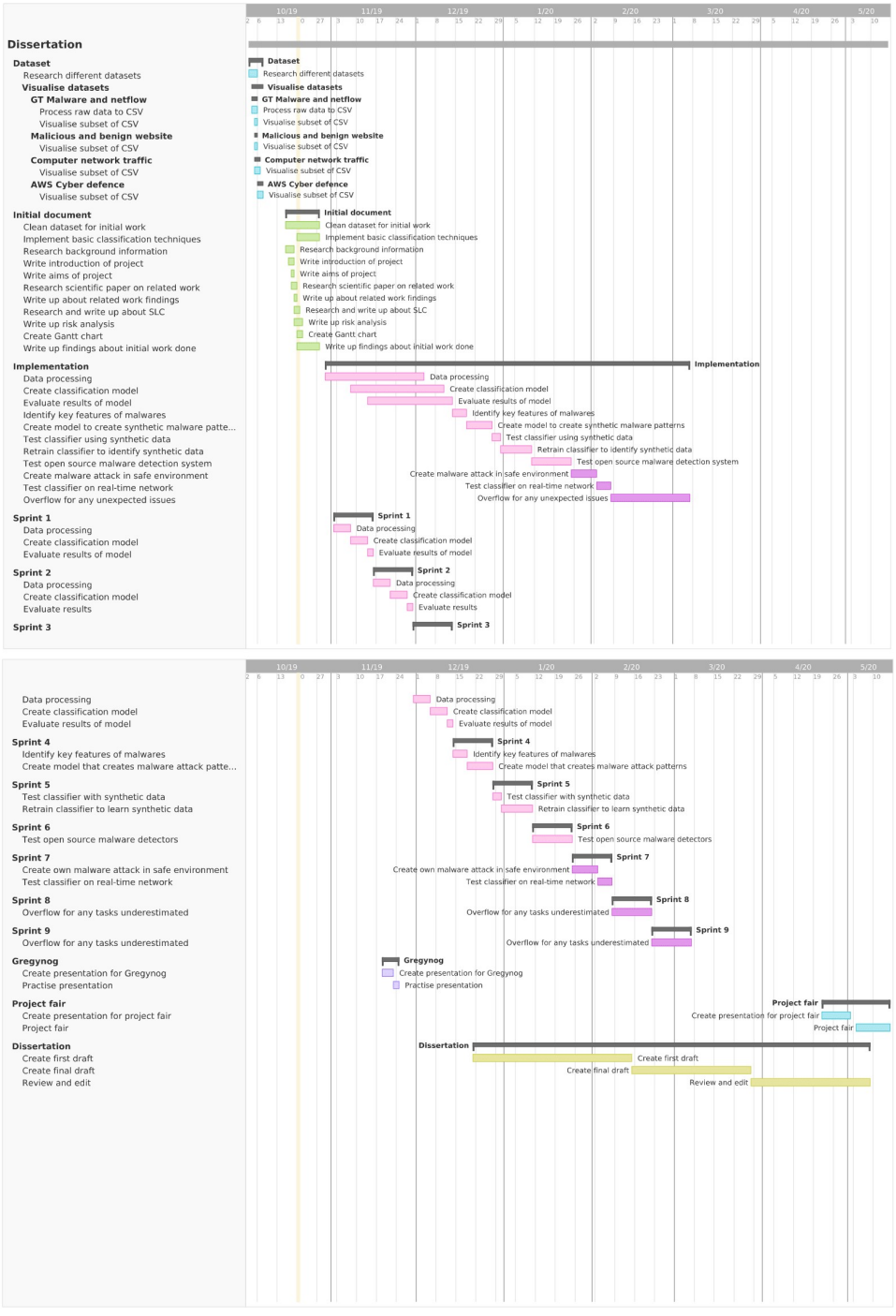

Figure 3: Gantt chart

3 Exploring the field

3.1 Related work

The hunt for a high true positive with low false positive IDS system has been sought for a while. Variety of different algorithms have been implemented in order to find one true solution.

Most attacks have a certain pattern that they follow resulting in two main approaches [10]. Anomaly detection and signature-based detection. Given a normal day-to-day activity, if a deviation occurs then that is recorded as an anomaly. That is the rule that anomaly detectors follow. Whereas for signature-based detection, it's based on the fact that an attack has a known pattern. If a known pattern is detected, there must have been an attack.

However, they both fall short. If a day-to-day activity is malicious, anomaly detectors will fail to flag it. Whereas for misuse detectors, the rise of polymorphic and obfuscated malwares is rendering it useless [10].

Nevertheless, many academics have implemented a range of different algorithms that attempts to detect malware.

One paper introduces a novel method for signature-based intrusion detection [11]. Creating an updating database that stores most frequent signatures to detect. The results show that the rate of false positives lowered, and malware detection speed increased. However, the database requires the administrator to choose whether signature is dangerous or not. Furthermore, other paper criticizes signature-based detection for its lack of ability to detect innovative malwares [12].

By utilizing data mining techniques such as Bayesian networks, to select features of importance, they were able to detect various attacks such as DOS with 100% accuracy. Their accuracy did falter for other attacks such as User2Root at 84%. [10].

Further research has been done on applying algorithms that are capable of learning and understanding features on a massive scope. Genetic algorithms mixed with traditional algorithms resulted in lower false positives and false negatives [12].

By using several different supervised techniques such as support vector machines, one paper managed to obtain high classification results. However no single algorithm was best at classifying all different attacks, each one had their own strengths. But, relatively they all showed promising results [13].

Unfortunately, even with many different implements, current IDS systems present different results for the same situation [14]. Judging the results can also be difficult. IDS systems that have high true positive and low false positive rate could lead to a false sense of security, especially if the attacks are over a large portion of traffic [14].

This shows that there's progress still to make when implementing an IDS system that has high positive rate with low false positives. One that is also able to handle the constant face-paced changes of evolving malwares.

3.2 Technology review

3.2.1 Language - Python 3

To be able to complete this project, I will be using Python 3 as it provides many native packages and resources that I can use.

For machine learning, there is a variety of packages that has all the core functionalities already covered.

1. Tensorflow

   Tensorflow is a package that can perform high-end numerical computations [15] and can create neural networks. Manually implementing a neural network is another project on its own therefore I will use Tensorflow as it abstracts away all that work.

2. Keras

   Keras is another machine learning framework that adds another layer of abstraction on top of Tensorflow. It also supports other machine learning packages such as Theano however for this project, I will be using it for Tensorflow.

   Keras makes it easier to implement models for neural network rather than using Tensorflow directly. Also, implementing custom functions such as loss, is easy to do with Keras.

3. Scikit-Learn

   Another machine learning package that contains a multitude of functions such as clustering, regression and classification algorithms [15]. Instead of just neural networks, there are many other algorithms such as random forest, which Scikit supports.

4. NumPy

   This provides a data structure known as NumPy that handles multidimensional array objects. It includes a vast range of methods on arrays that have quick execution time. This makes it easier to handle big data.

5. Pandas

   Pandas is another package for handling big data. Data can be loaded directly into Python from various formats such as CSV easily with Pandas. It also contains various functions to handle complex data operations [15].

6. Matplotlib

   Every machine learning project starts out with visualizing the dataset. Matplotlib is a great tool that allows you to plot various charts easily. However, there are other visualization packages such as Seaborn which provides more functionalities. Although, they are all built on top of Matplotlib. When necessary I will use other visualization packages however Matplotlib provides the core functionalities.

   Whereas if I were to implement my own exploit, Python again provides a lot of packages that are suitable for network communication.

Whereas if I were to implement my own exploit, Python again provides a lot of packages that are suitable for network communication.

1. Scapy

   Scapy is a package that allows manipulation of network packets. It provides the utility capable from tools such as Wireshark [16] and more. Aside from the norm such as scanning packets, Scapy also allows the ability to create attacks such as ARP poisoning. It is a powerful tool with a simple user interface.

2. Socket

   This is a low-level networking interface provided by Python. Socket provides the full functionality for network communication however there are many other packages such as Scapy that is easier to use. But most of these packages, including Scapy have Socket as their underlying tool.

3. Virtualenv

   This is a native Python package where the user can create secluded Python environments. Any experiments I perform will only exist in that virtual environment, ensuring no clashes of packages. Furthermore, there are many Python versions and many packages, if a package that is necessary is not supported by my version of Python, virtualenv makes it easy to switch.

3.2.2 IDE - PyCharm & Jupyter Notebook

There are also various integrated development environments (IDE) that work with Python. But for this project I will be mostly using PyCharm by JetBrains. PyCharm comes with an integrated visual debugger, a built-in terminal and version control support for Git. It also has a scientific version which is made for projects that use scientific packages such as NumPy.

Whilst I will do most of my development in PyCharm, I will also use Jupyter notebook. This is a web application that allows you to code in your browser. It is great for creating a presentation or writing small snippets of useful code down.

3.3 Initial analysis of dataset

3.3.1 Dataset selection

Data drives machine learning. Immense amounts of data have been collected, ready to be analysed however it is lacking for cyber security, especially for network analysis. Creating a network dataset that is unbiased and realistic proves to be a difficult challenge [17]. Unfortunately, the initial step would be finding a suitable dataset. Despite the issues with collecting network data, there are still many public resources available.

I have identified four different datasets that could be suitable for this task. Since they are all large, its impossible to understand what information they have without visualizing the data. I chose a subset of the data for each dataset and only few attributes to visualize. There are techniques available to learn what features are important however since this was just preliminary work, I did not delve deep. Also, since I was using only a subset, the analysis could have been misleading and/or important data were missing.

1. GT Malware netflow

   This is an ongoing project by Georgia Tech (GT) [18] where they are collecting a daily network feed in an isolated environment. Specific programs are executed for short period of time and recorded. However, the programs executed are not specified, only the network activity is just logged using nfcapd.

   Before I could visualize this dataset, all files had to be converted from nfcapd to CSV. By using the same tool that produces nfcapd files, NFDump, I generated the CSV files. However strangely, the CSV produced had extra attributes which did not exist in the original nfcapd file. All values for the extra attributes were set to zero, which I am therefore assuming were just default attributes that were not recorded by GT.

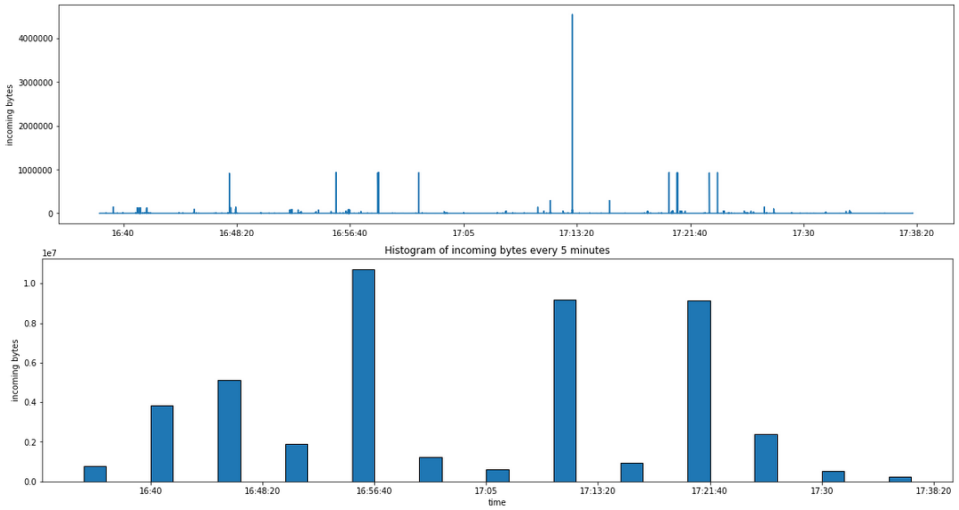

Figure 4: GT Malware netflow visualisation

For this dataset, I decided to visualize the number of incoming bytes against a certain time period which resulted in figure 4. It shows clearly that at 17H30M20S there was a spike of incoming data. There are also many periods where there is very little network activity happening. However, since the chart is so biased against the maximum incoming bytes, if the incoming bytes were very frequent but small, it would not show as clearly. Also, there could have been duplicate time entries but only the maximum value would have been displayed. Therefore, I also created a histogram of incoming bytes with bins of

5-minutes time entries.

The histogram showed that there were a lot of activity happening in time period before and after the spike which were just hidden.

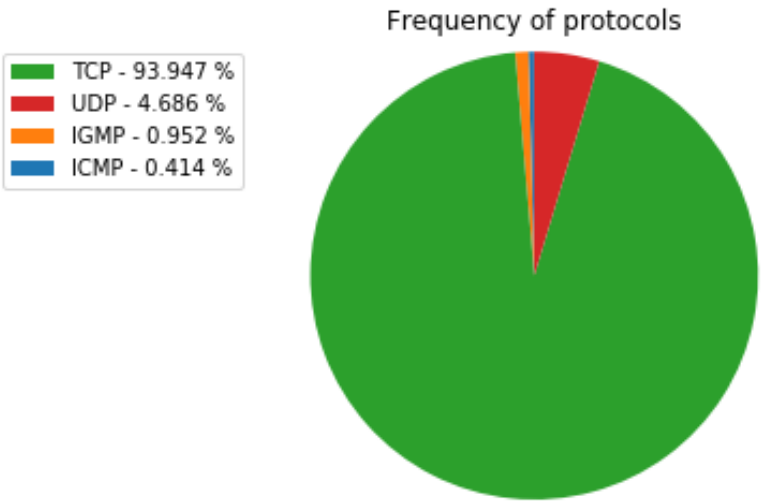

Figure 5: Frequency of protocols

Finally, I drew a pie chart (figure 5) to show what type of protocol were being used the most. This was to see what kind of network data was being passed around. Some protocols are also more vulnerable than others.

2. Computer network traffic

   This dataset is relatively small compared to others consisting of only 500k lines and 4 features [19]. In this dataset, 10 local workstations were monitored over 3 months period and half of these became compromised and were part of a botnet.

   Since there weren't many attributes to visualize, I went with the easiest one – "flow" which is the number of connections for that given day.

   Figure 6 clearly shows one machine number 4 had been compromised in September. However, that was for only one machine, there are still others to find. The Y axis again was very biased towards the max value, I decided to set a limit to the Y axis to make it easier to see the traffic for other machines

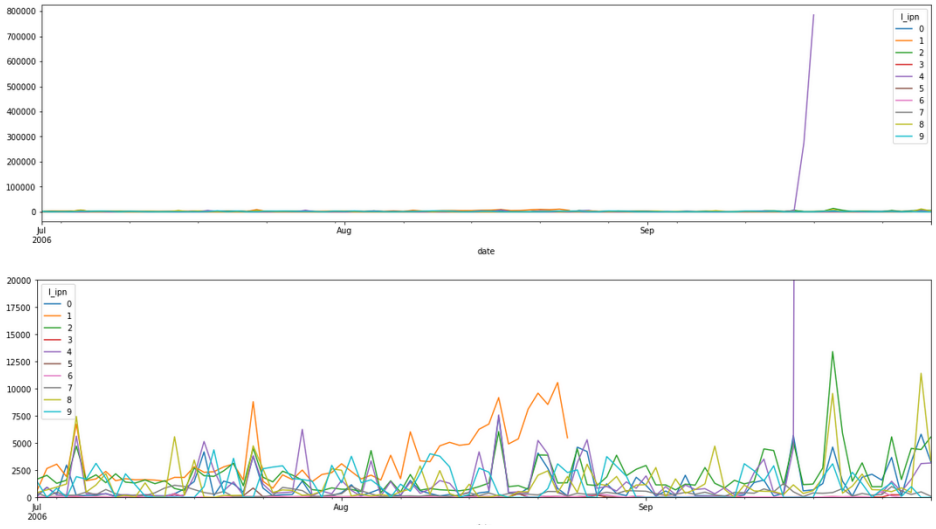

Figure 6: Flow vs time

3. Malicious and benign websites

   This dataset is different to the rest since it does not deal with network traffic. I was mostly just curious as what this dataset contained.

   This dataset contains a list of attributes about a website along with a label that says whether it is malicious or not [20].

   Since the dataset is labelled, there are different techniques to visualize the dataset. I drew a normalized boxplot (figure 7) to start off with to see if there were any outliers. Since the dataset contained both categorical and numerical data, I just plotted the numerical information. There are methods to convert categorical to numerical such as one hot encoding however I did not do it for this dataset. This graph clearly shows that there are various outliers that do not fit the rest, some are very prominent compared to others.

   I also drew a correlation graph (figure 8) since it can easily show what attributes are responsible for indicating that a website is malicious or not. However, its normally a set of features that indicate whether a website is malicious or not, not just one. Unforunately the correlation graph is unable to show this.

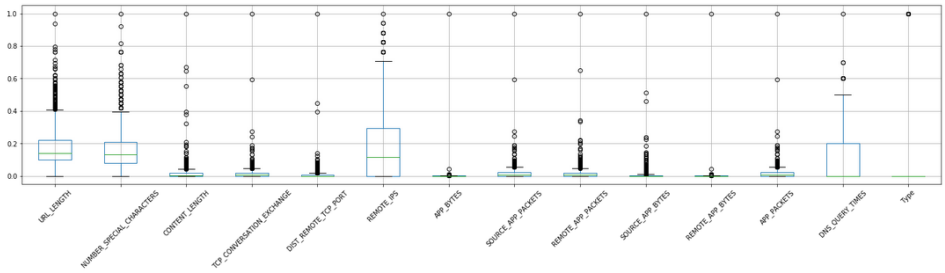

Figure 7: Boxplot of numerical features

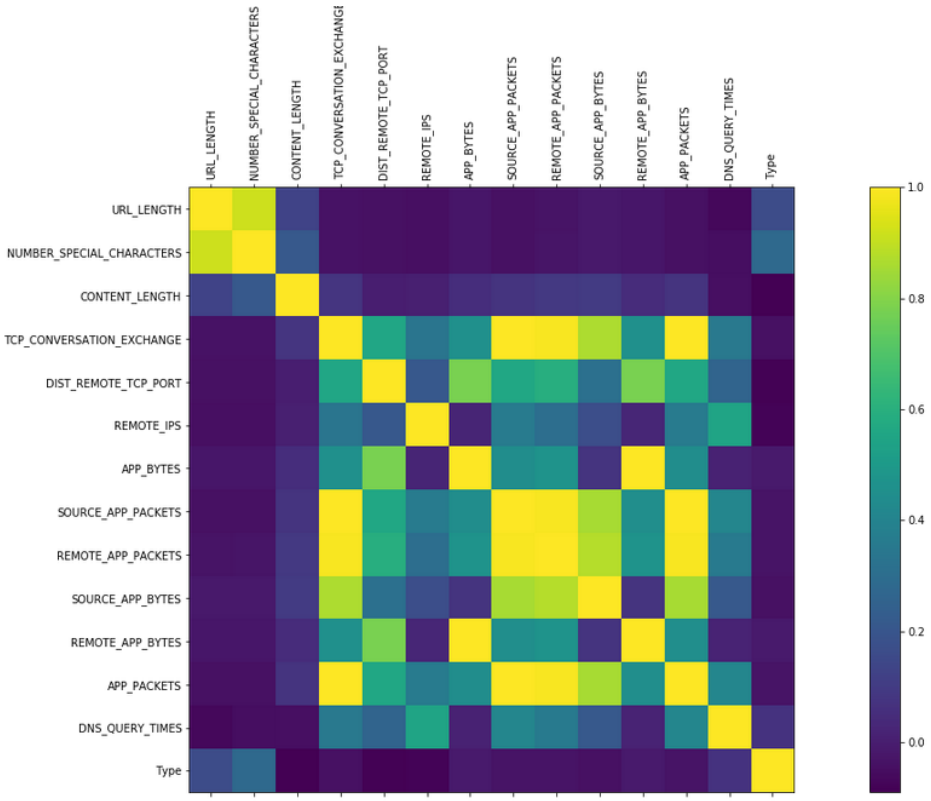

Figure 8: Correlation matrix

4. AWS Cyber defense

   For my final dataset I choose AWS Cyber defense. This dataset contains network traffic of 420 PCs and 30 servers which were victims to 7 different attacks such as DDoS [21]. All network traffic is labelled

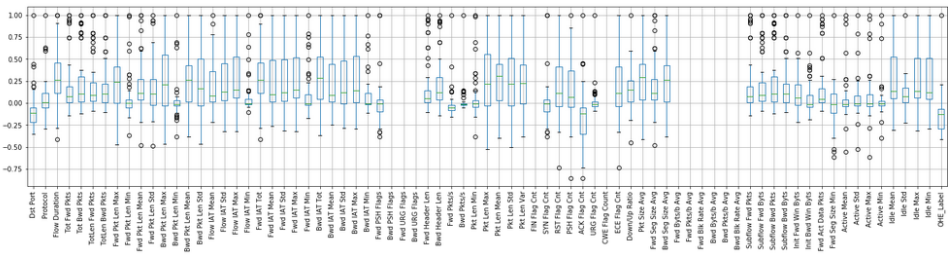

Figure 9: Boxplot of numerical correlation features

with its respective attack. This is also the biggest dataset; its raw size comes up to 220gb or 7gb of CSV.

Similarly, to benign and malicious websites I drew a boxplot (figure 9 followed by a correlation (figure 10) to visualize it quickly instead of choosing certain features. However, this time the label was categorical therefore I had to convert it to numerical value by encoding it. Also, instead of plotting just raw data, I drew the boxplot using the correlation data.

There were many other datasets to choose from, a notable one is KDDCUP99 [22] which was the first "good" network dataset. However, the dataset is quite old therefore I decided not to use it.

After discussing with experts in the field, I decided to use AWS cyber defense dataset. First, the data is labelled which means instead of attempting to cluster data to find anomalies, the data is already clustered to say whether an attack happened or not. Furthermore, there are seven different attacks in this dataset, covering most of the most common 8 network attacks, which will help me create a more robust system.

However, this dataset is quite tricky in the sense that it is bigger than rest, it will require good management of big data. Also, more classifications can make it harder. The model may perform well in one classification but fail for the rest, fine tuning parameters without affecting the good results will be difficult.

3.3.2 Basic classifiers

There is currently a lot of hype around deep learning however there are still basic classifiers that can be trained as initial findings to understand the

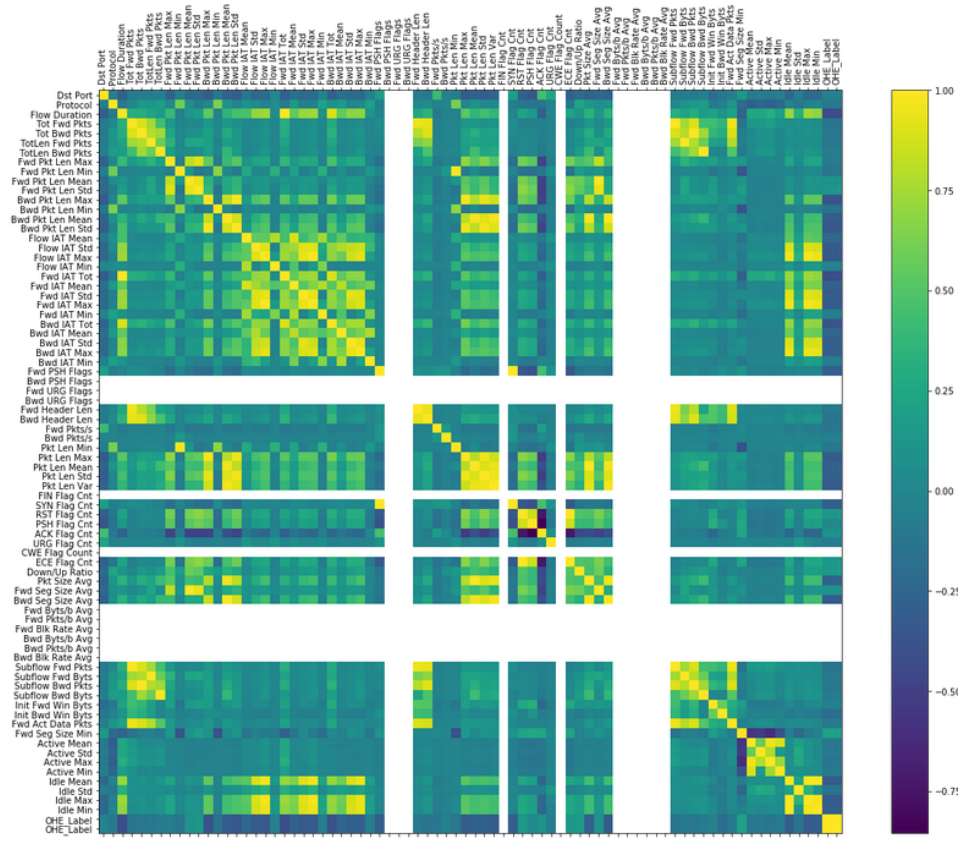

Figure 10: Correlation matrix

structure of the data. They have many advantages over deep learning such as being easy to implement and reduced training time.

But before training any classifiers, the data will have to be visualised and cleaned. Since I am working with a huge dataset containing multiple CSVs, I will start by visualising only one CSV. If I were to visualise all of them at once, important features that are only applicable to one CSV may be lost and the processing time will be longer.

I chose a random CSV of 300mb containing all information of an attack where the victim devices became part of a botnet. The data is all labelled, containing 1million rows and 80 features where most of it was benign data.

The first step would be to load the data into my script however since it is quite large, normal methods do not work. Reading the CSV in requires it

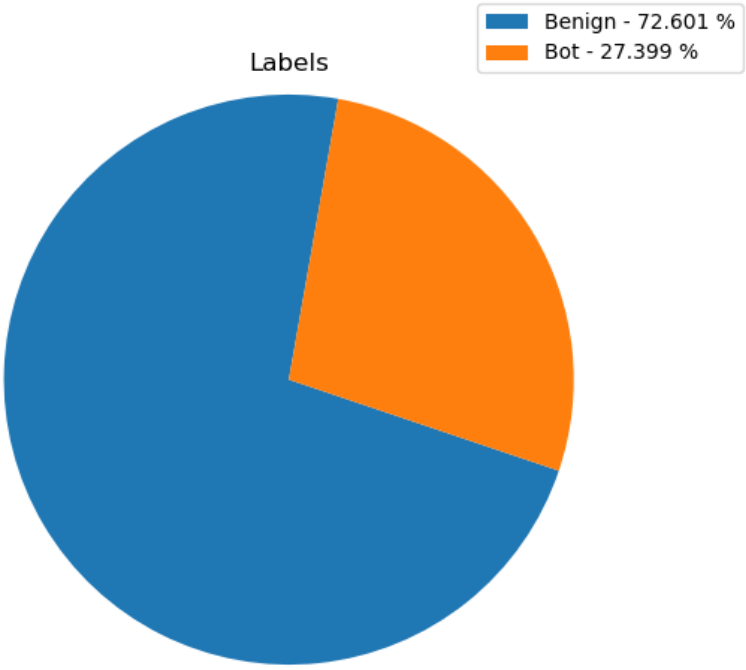

Figure 11: Ratio of bot vs benign labels

to be "chunked" where N number of rows will be read in at a time. This also requires an extensive amount of memory. This step is quite trivial however still important when processing large datasets.

Next the data needs to be visualised to see if it contains any abnormalities and there were quite a few. The first being that in random parts of the CSV, the header would repeat instead of the actual data. This caused a lot of issues as all features were no longer their correct datatype and everything became string. Fixing it was easy as when the data was being read in, filter out rows that contained header values again.

One of the biggest issues when handling big data would be occurrences of missing data. Many classifiers cannot work with such information missing as well, therefore I visualised all NaNs that existed in the data. It resulted that there was only one feature that contained NaNs.

However, there were also feature values where it was set to "Infinity" (in string) rather than an actual value. Again, classifiers will find this difficult

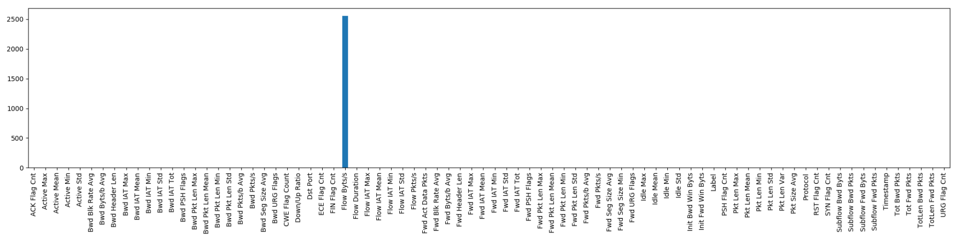

Figure 12: Sum of NaNs in feature columns

to handle.

There are various techniques when it comes to handling missing data, such as replacing NaNs and incorrect values with the mean or median and so on. However, each time this is done, the impact also needs to be considered. Changing the value to e.g. the mean will make it lose its original value and it might have been of importance. The easiest solution to handling missing data would be to just remove it. Due to lack of time, I only trained classifiers where all missing data were removed however it does need to be visualised to understand why there were in the first place.

Flow IAT M	Flow IAT S	Flow Pkts/	Fwd Act D	Fwd Blk R	Fwd Byts/	Fwd Head	Fwd IAT M	Fwd IAT M	Fwd IAT M	Fwd IAT St	Fwd IAT T	Fwd PSH F	Fwd Pkt Le	Fwd Pkt Le	Fwd Pkt Le	Fwd Pkt Le	Fwd Pkts/	Fwd Pkts/	Fwd Seg S	Fwd Seg S	Fwd URG F	Idle Max	Idle Mean	Idle Min	Idle Std	Init Bwd V	Init Fwd V	Label
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	176	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	180	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	405	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	31	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	126	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	180	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	176	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	118	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	237	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	131	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	31	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	31	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	31	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	181	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	180	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	181	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	176	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	5187	Benign
0	0	Infinity	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	-1	31	Benign

Figure 13: Snippet of CSV rows containing NaNs

Upon inspecting the feature values, it seems that when there was a NaN most of the values were set to zero where only few features are changing such as destination port. This suggests to me that they simply collected invalid data for a brief time. Also, all NaNs contained the label “Benign”. This further supports my idea, however if such an activity did occur there is a chance that the classifier I train with this data removed could label it as “Bot” since it is irregular. Although these occurrences were very rare, 2k rows contained NaNs out of a million or 0.2% of the data.

With missing data handled, it is then trivial to train classifiers.

Random forest classifier

A random forest classifier is a learning method where it contains many regression trees [23]. Each tree is based on a random subset of inputs which reduces the correlation between the trees therefore improving the performance of the random forest.

Using a cleaned dataset where it contains no missing values, it took 115 seconds to train using 100 trees. The accuracy rate was 99%, which is pretty impressive for how quickly it trained.

Linear support vector machine

Support vector machines are based on statistical learning theory where they implement methods to reduce training error to improve its performance [24].

This algorithm managed to train the data quicker than random forest at only 15 seconds with an accuracy rate of 98%. However it did require more processing than random forest as SVM do not scale well. Data had to be normalised between 0-1 which drastically improved the performance (from over an hr to under a few seconds). However this does have implications as the data is normalised, the model may have already learnt patterns from this, rather than from the raw data.

Dataset findings

Both classifiers returned incredible results for very little training time. However this was just preliminary work to understand what was in the data. More work will have to be done to understand all the CSVs and their features and understand how they correspond to their label. Training a neural network will also require considerable effort to optimise for all classifications.

4 Conclusion

From my research, it shows that there is still a need for a good intrusion detection system that has a high true positive rate and a low false positive rate and with the capabilities to adapt to new evolving malwares.

To create this intrusion detection system, I have chosen a dataset containing multiple different attacks on a network. Using this dataset, I will create a classifier that is can detect these attacks with high accuracy. My preliminary work using random forest and linear support vector machine to create a classifier that can detect if a device is part of a botnet shows

promising results.

There are still many different algorithms to implement to produce the best results including deep learning. This will require strenuous processing and visualization of the data.

Finally, if the model can understand different classifications well, then I will produce synthetic data and retrain the model such that it will be able to understand the difference between a real and fake attack. This should then reduce the number of false positives.

References

[1] Global internet usage in 2019. https://wearesocial.com/blog/2019/01/digital-2019-global-internet-use-accelerates. Accessed: 2019-10-18.

[2] Cybersecurity statistics of 2019. https://www.varonis.com/blog/cybersecurity-statistics/. Accessed: 2019-10-18.

[3] World Economic Forum. *The Global Risks Report 2018*, volume 13. 2018.

[4] Top 8 Network Attacks by Type in 2017. https://www.calyptix.com/top-threats/top-8-network-attacks-type-2017/. Accessed: 2019-10-18.

[5] Joel Henry and Sallie Henry. Quantitative assessment of the software maintenance process and requirements volatility. In *Proceedings of the 1993 ACM Conference on Computer Science*, CSC '93. ACM, 1993.

[6] Ken Schwaber. *Agile Project Management With Scrum*. Microsoft Press, 2004.

[7] About project boards. https://help.github.com/en/articles/about-project-boards. Accessed: 2019-10-20.

[8] How and why to use agile for machine learning. https://medium.com/qash/how-and-why-to-use-agile-for-machine-learning-384b030e67b6. Accessed: 2019-10-20.

[9] S. K. Wagh and S. R. Kolhe. Effective intrusion detection system using semi-supervised learning. In *2014 International Conference on Data Mining and Intelligent Computing (ICDMIC)*, 2014.

[10] Srilatha Chebrolu, Ajith Abraham, and Johnson P. Thomas. Feature deduction and ensemble design of intrusion detection systems. *Computers & Security*, 24(4), 2005.

[11] A. H. Almutairi and N. T. Abdelmajeed. Innovative signature based intrusion detection system: Parallel processing and minimized database. In *2017 International Conference on the Frontiers and Advances in Data Science (FADS)*, 2017.

[12] T. Mehmood and H. B. M. Rais. Machine learning algorithms in context of intrusion detection. In *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, 2016.

[13] T. Mehmood and H. B. M. Rais. Machine learning algorithms in context of intrusion detection. In *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, Aug 2016.

[14] E. Guillen, D. Padilla, and Y. Colorado. Weaknesses and strengths analysis over network-based intrusion detection and prevention systems. In *2009 IEEE Latin-American Conference on Communications*, 2009.

[15] Top 8 python libraries for machine learning & artificial intelligence. `https://hackernoon.com/top-8-python-libraries-for-machine-learning-and-artificial-intelligence-y08id3031`. Accessed: 2019-10-21.

[16] 5 python libraries every pentester should be using. `https://www.nopsec.com/5-python-libraries-every-pentester-should-be-using/`. Accessed: 2019-10-21.

[17] J. O. Nehinbe. A critical evaluation of datasets for investigating idss and ipss researches. In *2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS)*, 2011.

[18] Gt malware netflow. `https://www.impactcybertrust.org/dataset_view?idDataset=1143`. Accessed: 2019-10-21.

[19] Computer network traffic. `https://www.kaggle.com/crawford/computer-network-traffic`. Accessed: 2019-10-21.

[20] Malicious and benign websites. `https://www.kaggle.com/xwolf12/malicious-and-benign-websites`. Accessed: 2019-10-21.

[21] A realistic cyber defense dataset (cse-cic-ids2018). `https://registry.opendata.aws/cse-cic-ids2018/`. Accessed: 2019-10-21.

[22] Kdd cup 1999 data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. Accessed: 2019-10-21.

[23] S. P. Sotiroudis, S. K. Goudos, and K. Siakavara. Neural networks and random forests: A comparison regarding prediction of propagation path loss for nb-iot networks. In *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, May 2019.

[24] Dong-Xiao Niu, Qiang Wanq, and Jin-Chao Li. Short term load forecasting model using support vector machine based on artificial neural network. In *2005 International Conference on Machine Learning and Cybernetics*, volume 7, Aug 2005.