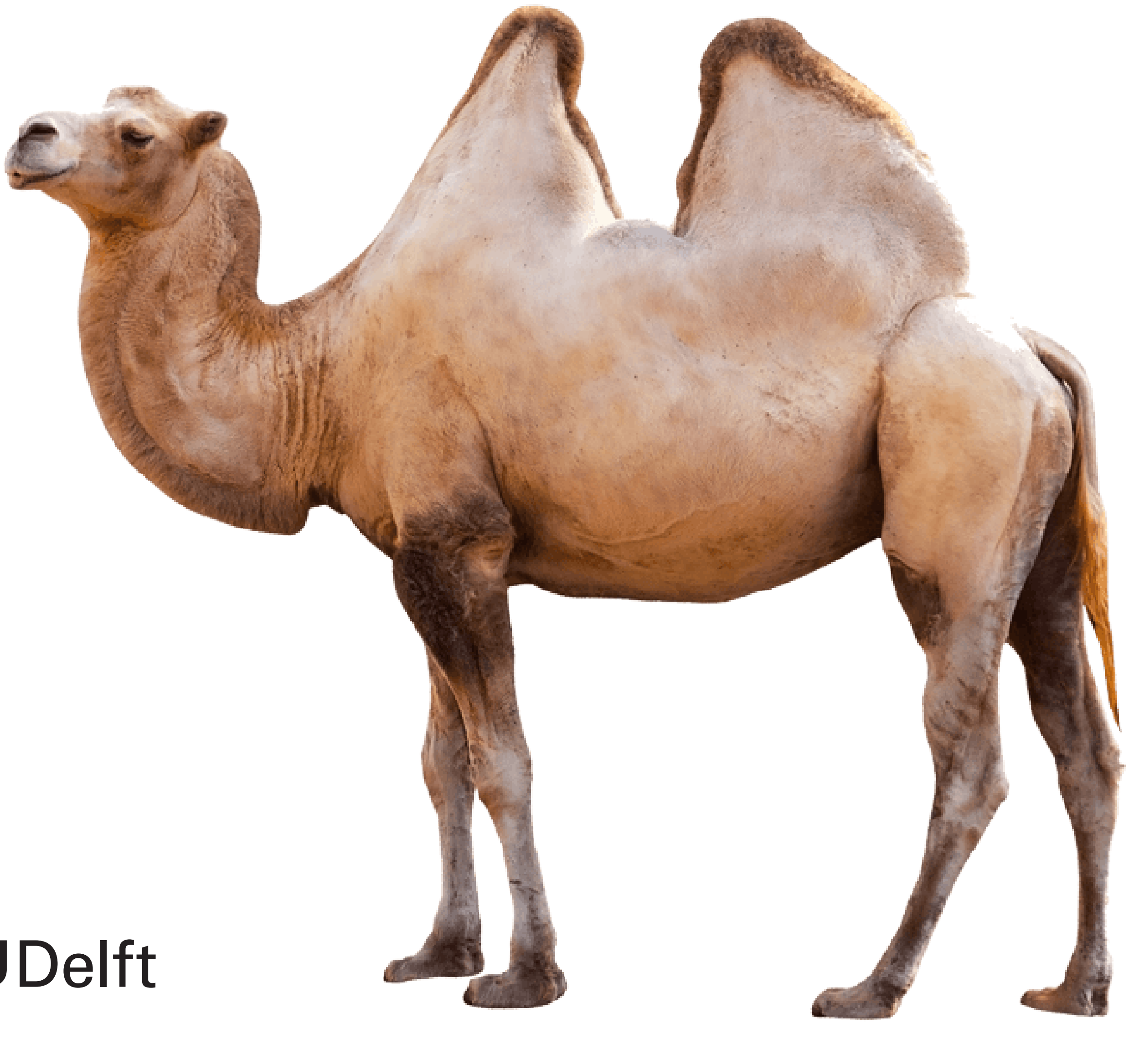
CAML-IDS

A framework for the correct assessment of machine learning-based intrusion detection systems

Mathew Vermeer



CAML-IDS
A framework for the correct assessment of machine learning-based intrusion detection systems

by

Mathew Vermeer

to obtain the degree of Master of Science in Computer Science
Data Science & Technology Track
with a specialization in Cyber Security
at the Delft University of Technology,
to be defended publicly on 15 July, 2019.

DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics & Computer Science

Mathew Vermeer: *CAML-IDS: A Framework for the Correct Assessment of Machine Learning-based Intrusion Detection Systems*, © 2019

STUDENT NUMBER:	4216989	
THESIS COMMITTEE:	Prof. dr. ir. R.L. Lagendijk,	TU Delft
	Prof. dr. M. van Eeten,	TU Delft
	Dr.-Ing T. Fiebig,	TU Delft
	Dr. S. Picek,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Abstract

The Internet is a relatively new technology that the world has become immensely dependent on. It is a tool that makes it possible to simplify our lives and better our society. But as with many things, there are people who with to exploit this tool we have for their own malicious gain. One of the mechanisms that we can use for protection against these malicious actors is the intrusion detection system. Machine learning-based intrusion detection systems (IDS) have been heavily researched for a number of years now. Much of this research, though, appears to be conducted using improper methodologies and incorrect evaluation. Such methodologies include training and testing IDSs with unrealistic data and using uninformative metrics to determine performance. In this research, we perform a case study using one such IDS. This IDS is trained and evaluated using real network traffic collected from a real-world network. Additionally, we test its performance on actual attack traffic. This research demonstrates that an IDS that is trained with unrealistic data performs nowhere near as well as is claimed by the author when trained using real network traffic. Finally, we propose CAML-IDS, a framework for the correct assessment of machine learning-based intrusion detection systems. This framework can assist future IDS research by preventing incorrect evaluation, in turn preventing the formulation of incorrect research.

Preface

For the longest time, I thought artificial intelligence and cyber security were either/or subjects. Being able to combine the two, as well as being able to make a contribution to this field is a dream come true.

Before you lies my contribution to this field: CAML-IDS. This is a framework that allows for the correct assessment of machine learning-based intrusion detection systems. It is written in order to fulfill the graduation requirements of the Data Science & Technology Track of the Computer Science program with a specialization in Cyber Security from Delft University of Technology. Most of the work was performed at the Faculty of Technology, Policy and Management.

First and foremost, I would like to thank my supervisor Tobias Fiebig, who provided me with all the support I could have possibly wanted, without the need for any spoon-feeding. I am extremely grateful for this guidance that allowed me to reach the finish line of this project. A big thank you to Michel van Eeten, who helped me out when, planning-wise, things were not going as smooth as we would have liked. Thanks to all members of my thesis committee for their valuable feedback that helped me turn a stack of papers into a coherent report. Finally, I will be forever grateful to my parents, my brother, my friends, and my girlfriend for their constant support throughout and understanding for all the absences and cancelations due to being glued to the computer screen.

It has certainly been a tough journey, but it is one I will certainly look back on with a sense of pleasure, pride, and satisfaction. It is my hope you can appreciate this final product as much as I enjoyed working on it.

Mathew Vermeer
Delft, July 2019

Acronyms

AIDS	application-based intrusion detection system.
ANN	artificial neural network.
AUC	area under the curve.
BGP	Border Gateway Protocol.
DNS	Domain Name System.
DoS	denial of service.
EER	Equal Error Rate.
HIDS	host-based intrusion detection system.
IDS	intrusion detection system.
IoT	Internet of Things.
IPv4	Internet Protocol Version 4.
IPv6	Internet Protocol Version 6.
MSE	mean-square error.
NIDS	network-based intrusion detection system.
PCA	principal component analysis.
pcap	packet capture.
PR	precision-recall.
R2L	remote-to-local.
RMSE	root-mean-square error.
ROC	receiver operating characteristic.
U2R	user-to-root.
VM	virtual machine.

Contents

1 Introduction

The advent of computing and the Internet has allowed us to construct digital systems that increase productivity, run businesses, and manage critical infrastructure. The Internet has also provided us with a way of sharing and accessing all of these digital products and services from any point in the world. Businesses and organizations have adapted it into their daily operation.

Hence, businesses and organizations have become extremely reliant on these systems and their proper functioning. All of an organization's sensitive data is stored on their systems, and the services they provide are all made possible through the proper functioning of the same systems. That means that any abnormality in the availability of these systems can have consequences for the organization.

Additionally, for certain institutions, malfunctioning digital systems can also have implications for human life and the environment itself. Take, for instance, malfunctioning sensors in a nuclear power plant [24], or a country's critical infrastructure with an openly accessible and poorly secured management panel [4].

While some failures can occur due to honest and unintentional mistakes by an administrator, designer or legitimate user, there are malicious actors that purposely and deliberately compromise and cause damage to organizations' systems. This is even made easier by the fact that many of these systems are directly connected to the Internet, including critical infrastructure systems [28].

Several tools and techniques exist that help protect networks from malicious actors. A common instance of such tools is using a blacklist-based filter. A blacklist only keeps track of illegitimate traffic, meaning that it allows all traffic, except for the types that are explicitly defined in the filter. A blacklist is easy to manage, but not entirely effective. It is not feasible to define *all* malicious traffic in the world. Even if it was, every packet would then have to be evaluated against a huge list before being allowed.

On the other hand, we have whitelist-based filters. This filter only allows traffic that has been explicitly defined, and discards any other traffic. While effective, it is this effectiveness that is also its disadvantage. The filter would need to be reconfigured every time a user needs to use some software that produces undefined traffic.

A more advanced approach is the intrusion detection system (IDS). An IDS is a system that is used to detect unwanted intrusion attempts in a network. Since every organization has a different network, there is no *one-size-fits-all* approach for IDSs. For instance, one organization would like to block all Remote Desktop Protocol (RDP) traffic, while another organization might specifically need RDP traffic in order to provide their services.

IDSs can employ either signature-based or anomaly-based detection of threats. A signature-based IDS uses patterns extracted from known threats in order detect these same attacks in the future. Even though this approach allows for easy detection of already-known threats, new types of threats will not be detected, as no pattern for them is yet known.

Anomaly-based IDSs use machine learning to create a model of legitimate behavior, which is then used to detect illegitimate, or anomalous, behavior. Its advantage over signature-based IDSs is that they allow for the detection of previously unknown threats. There is also the possibility of false positives, however: that previously unknown legitimate traffic is classified as anomalous.

The performance of anomaly-based IDSs is influenced by the choice of training data, and the manner in which the system is trained with said data. Ideally, the training data should be as close to real-world data as possible. In the case an IDS, this means that the traffic used for its training should closely match the traffic of the network that it will protect.

Anomaly-based intrusion detection is an active field of research. Many different systems have been designed in order to detect malicious traffic. And while these systems claim a certain level of performance, we have noticed flaws in the methodologies used in much of the available literature. For instance, many use outdated and overly general network data to train and evaluate their systems. Others use too small an amount of network data. Due to this, it is possible that the reported performance will not be the same if the systems were to be tested in a real-world environment.

The lack of proper training and evaluation of classifiers in the literature calls for an alternative method that produces correct and reliable results. This research will focus on developing such a methodology. This methodology will enable the correct evaluation of machine learning-based IDSs using real-world traffic data. A case study will be conducted in order to evaluate the performance and validity of the developed methodology.

1.1. Research statement

How can a machine learning-based intrusion detection system be correctly evaluated?

The objective of this project is to create a methodology that allows for the correct evaluation of an intrusion detection system.

The research question stated above can be decomposed into several sub-questions. Answering these sub-questions will allow us to answer the main research question itself.

1. **How do we properly collect training data for the machine learning-based IDS?**

   Data collection is a necessary procedure for this project, since an IDS will need to be trained. Since the IDS will be trained to protect a specific network, the collected data must be representative of the environment it is collected from. Also, sufficient data must be collected to properly train the used IDS.

2. **How do we annotate anomalous data correctly?**

   Data that is anomalous in one network might not be classified as such in a different network. It is therefore necessary to select data that will be anomalous in the specific network that is being investigated. This could be both benign but unwanted network traffic, and malicious traffic.

3. **How can anomalous network traffic be injected correctly?**

   The evaluation of the IDS must be a realistic process. The simulated evaluation scenarios must be similar to those that occur in reality. The order and frequency in which the anomalous traffic is injected into the network must be realistic, as well as the amount of anomalous traffic that is injected in a single instance.

4. **What are appropriate metrics for determining the performance of an IDS?**

   Machine learning-based classifiers have many different use cases, and there are many different metrics that can be used for determining the performance of a classifier. The choice of metrics is an important one, since different metrics provide different information about a classifier. Image classification may require a different set of metrics than anomaly detection. To make sure this research is performed adequately, we must identify the metrics that provide the most amount of information about a machine learning-based intrusion detection system.

In order to fully answer the aforementioned questions, we will conduct a case study on an anomaly-based IDS that was designed and evaluated by its developers in an artificial and unrealistic environment. We will identify common mistakes and use the best practices from state-of-the-art research to develop a framework that allows for the correct evaluation and assessment of a machine learning-based IDS. With this framework, the IDS from the case study will be evaluated, and the acquired results can be compared with the original results. The results that we obtain by using this framework will much better illustrate the actual performance of the IDS.

1.2. Research outline

This research can be split up into two parts. The first part introduces the relevant terminology and literature, as well as the anomaly-based IDS that we will use to conduct our case study. Chapter 2 gives an overview of different types of IDSs that are developed by different researchers, the techniques that are used in their design, and their performance. Common ways of IDS evaluation are discussed, as well as the recommended ways that are considered best-practice. Based on the information from this chapter, Chapter 3 describes the IDS that is chosen for the case study, and elaborates on the manner in which it was evaluated.

The second part of the research is conducting the case study itself. Firstly, we need real-world data to perform this research. Before we can use this data, though, we need an accurate description of the network from which we capture this data. This enables us to determine ground-truth of the captured data. The process of capturing and filtering network traffic is described in Chapter 4. It is also interesting to study the IDS's detection performance on actual attack traffic. The process of constructing this attack traffic is described in Chapter 5. The final evaluation of the IDS is given in Chapter 6, and these results are discussed in Chapter 7. Finally, the constructed framework will be provided in Chapter 8, as well as some concluding remarks and future research possibilities.

2 Background literature

There is more to anomaly detection and IDSs than simply choosing arbitrary machine learning techniques. This chapter will elaborate on the different steps that need to be taken when trying to develop an effective IDS. The first section in this chapter will briefly introduce the the topic of IDSs and the purpose that they serve. There are many different machine learning techniques that can be applied, each with their own advantages and drawbacks. The second section will describe a number of machine learning-based IDSs that have been developed over the years. The used methodologies will be discussed, as well as the obtained results. Evaluation is a process that needs to be performed soundly in order to produce accurate conclusions. The third section will mention and discuss commonly used metrics and methodologies for sound IDS evaluation. Lastly, no machine learning technique can be properly evaluated without appropriate datasets. The last section in this chapter will focus on the suitableness of current widely-used datasets, as well as other techniques that are used to create or collect network traffic.

2.1. Intrusion detection systems

IDSs are classified by their placement, and by the techniques that are used for detection itself. As for the placement of an IDS, there are network-based intrusion detection systems (NIDS), host-based intrusion detection systems (HIDS), and application-based intrusion detection systems (AIDS). Depending on its method of detection, an IDS can be signature-based or anomaly-based [41].

A network-based IDS is named as such, because it is placed at a strategic point within a network and analyzes all network packets it receives to detect attacks. It is generally a single, independent host whose only purpose is to capture and analyze network traffic. NIDSs can easily allow for the monitoring or large networks. Since an NIDS needs to capture and process all the traffic it receives, it may encounter difficulties in doing so if placed within a considerably busy network. Processing speed is therefore a necessary attribute for any NIDS. When an IDS is installed on an existing host, it is called a host-based IDS. An advantage that an HIDS has over a NIDS is that it has the ability to analyze the memory and logs of the machine on which it is installed. This gives it the ability to discover which process or user is responsible for any malicious activity that is detected. A drawback, though, is that it is unable to monitor the behavior of the network, as the HIDS installed on a particular host will only have access to the traffic that this host receives. Application-based IDSs are similar to HIDSs, as they are installed on specific hosts. The difference is that AIDSs monitor only the activity belonging to a single application on a host. This specialization makes AIDSs very effective at detecting malicious activity, but any malicious activity outside of the monitored application is not seen [41].

Signature-based IDSs find malicious activity by matching it with a predefined set of patterns or events that are characteristic of known attacks. Although this technique certainly is effective at detecting known attacks, it struggles at detecting novel attacks. This is because a signature for the novel attacks is not available yet. Anomaly-based IDSs work with the assumption that malicious activities behave differently than normal activities. By establishing a baseline for normal behavior, it tries to detect the malicious activities by identifying these differences. In this manner, it is often able to detect not only previously-seen attacks, but novel attacks as well. This approach is hardly perfect, and as such produce false positive results [41]. The field of anomaly-based intrusion detection is a heavily researched one, and much of this research relies on machine learning techniques. The next section will discuss in more detail a number of anomaly-based IDSs that use machine learning techniques to detect anomalies.

2.2. Machine learning-based intrusion detection systems

Wang and Stolfo [89] developed a method for anomaly detection based on the analysis of payloads that are sent within the network. They call it a payload-based anomaly detector. Given the normal payloads, their system produces a byte frequency distribution of said payloads. This distribution serves as a model for all normal payloads. The centroid of this distribution—its geometric center—is computed during the learning phase, and is used for the anomaly detection. Specifically, incoming payloads are captured and their distance to this computed centroid is calculated. The payloads with a distance above a certain threshold are classified as anomalies. Wang and Stolfo use five differently learned models to evaluate their system. The five models are trained using the following data: (i) the packet's entire payload; (ii) the packet's first 100 bytes; (iii) the packet's last 100 bytes; (iv) the entire payload of each connection; *and* (v) the first 1000 bytes of each connection. In this research, Wang and Stolfo use the DARPA 1999 IDS dataset [31] for the training and evaluation of the classifier. Additionally, a two datasets were collected from a university web server by capturing network traffic on two different days. Wang and Stolfo focus solely on TCP traffic. Therefore, attacks using a transport-layer protocol other than TCP are not detected, and all TCP traffic is filtered out of the datasets. When training and evaluating the models on the DARPA 1999 dataset and limiting the false positive rate to 1%, detection rate was around 60%. They note, however, that their models have difficulty detecting attacks when the protocol's payload structure varies more than usual. Examples of these protocols where the models have trouble detecting attacks are SMTP and Telnet. They use false positive rates and ROC curves to evaluate their system's performance on the DARPA 1999 dataset. This evaluation is not done when using the university web server dataset. This is due to the fact that they did not separate the captured traffic into benign and malicious. Instead, they trained the models on the dataset from the first day and then tested them on the same dataset, as well as the dataset from the second day. This process was repeated for the second dataset. Then, the models were trained and tested on the union of both datasets. During this evaluation, the models were able to detect buffer overflow attacks and Code Red II attacks [78]. Since the data was unlabeled, no performance metrics were calculated.

Srivastav and Challa [74] propose two similar IDS frameworks that use neural networks to detect four types of attacks: denial of service (DoS), probing, remote-to-local (R2L), and user-to-root (U2R). DoS attacks include attacks such as SYN floods and Teardrop [61], probing includes activities such as port scanning, R2L attacks are attempts at achieving unauthorized access to a host from a remote machine, and U2R is gaining unauthorized access to local superuser privileges [75]. The proposed frameworks are layered, and each layer is a chain that is made up of a data pre-processor, an encoder, and a neural network. Each element in the chain passes the data to the next. Every layer has a separate neural network that is responsible for detecting a single type of attack. If a layer does not detect the type of attack assigned to it, the network traffic is passed on to the next layer. Once an attack is detected, the process stops and the data is blocked and labeled as malicious. The only difference between the two proposed frameworks is that one has a feature-extraction module in each layer after the data pre-processor, while the other one does not. This feature extraction is achieved by means of principal component analysis (PCA) [83]. To train and evaluate their system, they use KDD Cup 1999 dataset [85], which is derived from the DARPA 1998 dataset [86]. For the training process, the training dataset is divided into tree parts: 70% of the data into the training set, 15% into the validation set, and the remaining 15% into the test set. The system is trained until the mean-square error (MSE) on the validation set is constant for six epochs. Performance was evaluated using mainly confusion matrices. The first model achieved an overall accuracy of 97.1%, with a detection rate over 90% for every type of attack. However, the false positive rate of this model was nearly 0.25. The second model achieved a slightly lower overall accuracy, namely 94.4%. In this case, the false positive rate dropped considerably to 0.08. Somewhat noteworthy is that a layer would often detect attacks that did not correspond to that particular layer. For instance, in the second model, the layer responsible for detecting R2L attacks detected about 77% of U2R attacks before the traffic could reach the responsible U2R layer.

Naoum et al.[49] have constructed an IDS using a single neural network to detect all types of attacks. This contrasts the approach by Srivastav and Challa[74] that uses multiple layers of neural networks that each only detect one single type of attack. The neural network is trained using the RPROP (resilient propagation) algorithm developed by Riedmiller and Braun [63], which is their improvement on backpropagation using purely gradient descent. This algorithm allows for a faster and more efficient learning process. The first element in the system is a pre-processor. This pre-processor converts the data into a format that is compatible with the neural network. The neural network is learned with labeled data, and, therefore, in a supervised manner. During the learning process, the weights in the neural network are updated on every iteration until the MSE reaches a predetermined number. Naoum et al. experimented with a different number of hidden neurons during the training phase and found that 26 hidden neurons provide the highest detection rate. The data that is fed into the neural network is classified into one of five categories: normal, DoS attacks, R2L attacks, U2R attacks, or probing activities. Naoum et al. use the NSL-KDD [87] dataset, which is derived from the KDD Cup 1999 dataset. Evaluation is largely based on detection rates. A confusion matrix is provided, although not discussed. All attacks have a detection rate of over 96%, except for the U2R attacks, which are detected only 54.1% of the time. For the entire test set, they achieve a detection rate of 94.7%, with a rather high false positive rate of almost 15.7%.

Shone et al.[70] utilize deep learning techniques for intrusion detection. They feed network traffic data into their proposed “non-symmetric deep autoencoder” in order to learn features from the data in an unsupervised fashion. “Non-symmetric” refers to using a system containing only an encoding segment, instead of the usual autoencoder structure that contains the “symmetric” encoder-decoder segments. This is done to reduce computational complexity and execution time. The “deep” part of their proposed system refers to the multiple hidden layers that each encode the output of the previous hidden layer. They stack two of these non-symmetric deep autoencoders on top of each other, with the output of one being the input of the next. The final output of the encoders is then fed into a random forest, which performs the actual classification. They train and test the data on both the KDD Cup 1999 dataset [85] and the NSL-KDD dataset [87]. The metrics that are used to evaluate the classifier’s performance are accuracy, precision, recall, false positive rate, and F-score [71]. Additionally, a ROC curve was used to evaluate performance when using the NSL-KDD dataset. When testing on the KDD Cup 1999 dataset, their system achieved an average accuracy of 97.85% with a false positive rate of about 0.02. However, R2L attacks are rarely detected, and U2R attacks are never detected at all. Shone et al. point to the lack of of enough attack instances to train the classifier on as the reason for this poor performance. The NSL-KDD dataset test consisted of two parts. The first evaluation tested the classifier’s ability to separate the date into five different classes: normal, DoS, R2L, U2R, and probes. The second evaluation had the classifier separate the data into 13 classes: normal, and the 12 specific types of malicious activities present in the dataset. In the 5-class classification test, the classifier achieved an average accuracy of 85.42% with a false positive rate of approximately 0.15. Again, the classifier had difficulties detecting R2L and U2R attacks. Compared to the 5-class classification, the 13-class classification provided a 3.8% improvement in overall accuracy, which Shone et al. use to support their claim that their model works more effectively with more complex datasets.

Mirsky et al.[47] use autoencoders for intrusion detection, similarly to Shone et al.[70]. However, instead of stacking two autoencoders on top of each other, Mirsky et al. feed the features extracted from the network data into an ensemble, or collection, of autoencoders simultaneously. The process starts with the arrival of a packet and the extraction of a number of features from the packet, such as IP and MAC addresses, packet timestamps, and packet sizes. The learning process has two phases: the clustering together of the features, and the learning of the autoencoders. After the features are clustered together based on their correlation, each cluster of features is fed into a different autoencoder ensemble. The individual autoencoders measure the abnormality of each subspace (feature) of the packet. All of these measurements are then used as the input for the final layer of the system, which uses all of the abnormality measurements to calculate a final abnormality score in order to determine whether the packet is legitimate or not. They extracted their network traffic from two different networks. One was a private surveillance camera network, and the other one a small test network containing a few PCs and a collection of IoT devices. In order to evaluate the effectiveness of their system, Mirsky et al. performed a series of attacks on their networks and captured the traffic while performing the attack. These attacks included OS scans, SYN floods [7], and infecting a host with the Mirai botnet malware [1]. Mirsky et al. perform the attacks on the network one at a time. They then fed this attack traffic into their anomaly detection system. The evaluation of their system is based on true positive rates, area under the curve (AUC) values, and equal error rates. All the attack traffic tests they performed on their system, they also performed on a number of different anomaly detection systems. They rate the performance of their system based on how well the rest of these different anomaly detection systems fared. From the results we can see that their system significantly outperforms most other anomaly detection systems when evaluating fuzzing attacks, SSDP floods, and Mirai botnet infections.

2.3. Evaluation of intrusion detection systems

Network intrusions are anomalies, and anomaly detection is a classification problem. All instances are classified either as normal or anomalous. While being a classification problem, however, the usual practice of using simply overall accuracy does not provide sufficient information about the performance of an anomaly detection system, according to He and Garcia [19]. This is because of the inherent imbalance between normal and anomalous instances. In other words, there are far more normal instances than anomalous instances in datasets, which is why they are called anomalies. Take a dataset with 0.1% anomalies. A trivial classifier that classifies all instances as normal will still achieve an accuracy of 99.9%, all without detecting a single anomaly. He and Garcia [19] state that more informative metrics are needed to properly evaluate these types of systems. These metrics include receiver operating characteristic (ROC) curves, precision-recall scores and curves, and cost curves. The ROC curve plots a classifier's false positive rate against its true positive rate. It is used as a measure for a classifier's ability to separate classes [83]. ROC curves can be reduced to a single value by computing the area under the ROC curve (AUC). This AUC value is equivalent to the probability that the classifier will rank a random positive instance higher than than a random negative instance [15]. This means that the higher the AUC, the better the average performance of the classifier. The AUC value can be computed without an ROC curve for a single point using the formula [71]

$$AUC = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right),$$

where *TP* is the number of true positive instances, *FN* the number of false negative instances, *TN* is the number of true negative instances, and *FP* the number of false positive instances. DeLong et al. [12] state, however, that an ROC curve that has few thresholds will significantly underestimate the true AUC. An ROC curve with a single threshold is therefore a worst-case scenario. Precision and recall [83] are computed as follows:

$$\text{precision} = \frac{TP}{TP+FP} \text{ and recall} = \frac{TP}{TP+FN}.$$

These metrics are more informative than accuracy, because they penalize incorrect classifications [19]. A precision-recall curve is obtained by plotting precision and recall against each other. While ROC curves certainly are informative, both He and Garcia [19] and Saito and Rehmsmeier [65] state that precision-recall curves are more useful than ROC curves when evaluating binary classifiers on imbalanced datasets. ROC curves are overly optimistic for imbalanced data, He and Garcia [19] state.

Sokolova and Lapalme [71] state that another common metric for binary classifier evaluation is the F_1 score. This metric is the harmonic mean of precision and recall, and it is calculated with the following formula:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \text{ or } F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}.$$

Its values range from 0 to 1, where 1 is its best value.

Sommer and Paxson [73] also discuss IDS evaluation. They focus less on metrics, and more on methodologies for sound evaluation. First of all, they strongly suggest researchers manually examine the classification results. This is often difficult due to the black-box nature of many types of classifiers, such as artificial neural networks (ANNs) [90] and random forests [55]. Even so, Sommer and Paxson state that researchers need to understand the reason behind certain classification results. This can help researchers identify flaws in their systems. Manual inspection can help researchers identify what aspects of the data are responsible for its corresponding classification. For example, researchers can ensure their system is basing anomaly detection on multiple features, instead of just flagging packets based on their packet size. If researchers find that their system's performance is the result of faulty methodology, they can rethink their approach before publishing inaccurate research or implementing unreliable systems. Furthermore, they state that "the single most important step for sound evaluation concerns obtaining appropriate data to work with." This "appropriate data" is ground-truth data obtained from a real-world environment. Dataset collection will be further discussed in Section 2.4. Ground-truth in this sense means that we can say with certainty that data that is labeled as "normal" truly is normal, and "bad" data truly is bad. Sommer and Paxson go on to recommend that the system be trained with data different than the data used for the final evaluation. The best evaluation of all, Sommer and Paxson state, is testing an anomaly detection system in the real world. The system being deemed useful by real network administrators gives much support to the work that was performed by the researchers.

2.4. Dataset collection and generation

There is a particular pair of datasets that is widely used for machine learning-based intrusion detection research [5, 13, 48, 74, 89], namely the DARPA 1998/1999 Intrusion Detection Evaluation Datasets [31] by Lippmann et al. [33], and its derivative KDD Cup 1999 dataset [85]. It is a dataset generated by DARPA, as the name suggests, that consists of seven weeks of network traffic. Different versions exists, and these versions are identified by the year they were created. DARPA claim that this generated dataset is similar to real network traffic found inside actual Air Force bases. This network traffic contains benign activity of hundreds of users on thousands of UNIX hosts. Additionally, this dataset contains malicious traffic produced by an assortment of different attacks that were launched against hosts in the network. Originally there had been little intrusion detection research using a single dataset shared by different research groups. Privacy concerns about the captured network traffic always complicate the publication of such datasets. Also, a wide range of attacks must be successfully performed against different hosts with different systems. For the most part, datasets back then had neither a wide range of attacks nor a large number of hosts. The network in the dataset is split into two segments: the internal segment (i.e. inside the base), and the external segment (i.e. outside the base). In the internal segment, there are three machines that serve as the victim machines, and a gateway that serves as a link with the rest of Air Force base PCs and workstations. Most of the hosts in the internal segment run UNIX-based operating systems. The 1999 edition of the DARPA dataset also contains Windows NT hosts. A Cisco router connects the inside network with the outside. The external segment is supposed to represent the Internet. It contains two different gateways: one that provides access to outside web servers, and another that provides outside workstations access to the network. This second gateway is where the attacks are launched from. Finally, the traffic sniffer is located in front of the Cisco router, on the outside. This placement of the sniffer means that internal network traffic is not captured. Only the communication between the inside and outside networks is captured. See Figure 2.1 for an illustration of the network. This private network was used to generate network traffic that is “similar” to traffic found in real Air Force bases. The hosts within the internal network segment ran software that generated a large quantity of network traffic. This allowed a single host to act as if it were instead a collection of hosts, each with its own IP address. User actions such as sending email, using FTP services, or accessing other hosts via Telnet, were automated. These automated user actions were performed using the same software that was used in real Air Force base networks. Examples include *sendmail*, *ftp*, and *Telnet* [33]. Additional protocols that can be found in the dataset include HTTP, IRC, DNS, and SNMP, among others. The timing of the usage of the different services, as well as the proportions of all the traffic corresponding to each individual service, was based on actual Air Force base network usage, claim Lippmann et al [34]. Usage of synthetic data instead of actual network traffic is due to security and privacy concerns. For the 1998 version of the DARPA dataset, the network traffic included 38 different types of attacks. These attacks range from reconnaissance activities, like Nmap scans and port sweeps, to more malicious ones, such as Smurf [26] and Ping of Death attacks [9]. Much of the attack traffic was present in the training data. However, many of the attack types were only placed in the test data. This was done to test the effectiveness of IDSs on not only known attacks, but on novel attacks as well. Some of the attacks were performed on software vulnerabilities that were discovered specifically for the project. The 38 types of attacks are classified into the following four groups: Denial of Service, Remote-to-Local, User-to-Root, and probes. Of these four groups, the tested IDSs were able to “generalize well” to probes and User-to-Root attacks, but were not able to detect unseen Denial of Service and Remote-to-Local attacks accurately.

Many researchers have been using the DARPA datasets and its derivatives for their own research projects. They do this in spite of the thorough criticisms the datasets have received. McHugh’s 2000 paper [40] is one such example. On the datasets, he mentions the claim by Lippmann et al. that the network traffic contained in the datasets are “similar” to real Air Force base traffic, and criticizes the lack of evidence supporting that claim, statistical or otherwise. Also, McHugh states that Lippman et al. only speak of the frequency and usage time of software utilities. The similarity of the content of the network traffic created by the utilities is never discussed. It is only mentioned that the network traffic is constructed using public domain and randomly generated data sources. Furthermore, no analysis is provided as to the false positive rates of the synthesized background data. According to McHugh [40], this type of analysis is important, because in order for the synthesized traffic to be a valid substitute for real network traffic, the false positive behavior of the tested IDSs should not differ significantly when classifying real and synthetic network traffic. Another point raised by McHugh is that Lippmann et al. do not discuss the data rate within their synthetic network, nor any variation in this data rate over time. A quick examination of some of the DARPA dataset revealed an unrealistically low data rate, especially for a network containing such a large number of hosts. Due to this low data rate, the false alarm rates would differ significantly if the IDS were tested on a noisier and more realistic network. As for the attack data, McHugh criticizes the distribution of the attack traffic among the background traffic. Most notably, every type of attack was performed roughly the same amount of times. This creates an unrealistic scenario, according to McHugh, since surveillance or probing activities are by far the most prevalent cause of malicious traffic.

Mahoney and Chan [36] are others who criticize the DARPA datasets. They do this using a more detailed approach than McHugh [40]. They focus on specific aspects of the synthetic background traffic and simulated attack traffic. Firstly, they show that the network traffic collected from the artificial Air Force base contains simulation artifacts that can be utilized in order to detect attacks. To demonstrate this, they created an IDS of very poor quality. This IDS bases its anomaly detection on a single byte of a packet. If this specific byte was not seen during the training phase (and 60 seconds have passed without detecting an anomaly), then the packet is classified as malicious. It is clear that this IDS would be useless in a real-world scenario. In this particular case, however, this IDS detects 45% of all attacks, with 43 false positives. This performance is comparable with the best-performing systems in the original DARPA dataset evaluation. Mahoney and Chan collected network traffic from a university departmental server and mixed this captured traffic into the DARPA dataset. After training and testing on this newly created dataset, the IDS was barely able to detect any attack at all. Additionally, the false positive rate increased significantly. They also compared the synthetic background data to real network traffic they collected from a university server. This comparison yielded several findings. For instance, many of the DARPA packets have no IP, TCP, UDP, or ICMP checksums. Also, they found a lack of variety in many of the DARPA packet fields. The packets' TTL fields were set only to nine values out of the possible 256 values. The real network traffic, on the other hand, had 177 different values. Furthermore, compared to the DARPA dataset, the real network traffic had much more variety in the types of HTTP, SMTP, and SSH requests. Using these values as part of an anomaly detection rule would result in a high false positive rate, since the values are much less predictable in real-world traffic. This is especially evident given the low-quality simulation of several attacks. Attacks such as *dosnuke*, *neptune*, and *queso* [33], among others, use only two different TTL values. ICMP packets in the *smurf* [26] attack contain checksum errors. Detecting these attacks using these attributes might increase the performance of the IDS when operating on the DARPA datasets, but such an IDS would be worthless in the real world. Despite the many shortcomings of the DARPA datasets, Mahoney and Chan considered it to be the most sophisticated dataset that was publicly available. To alleviate the dataset's inherent problems, they propose mixing real traffic into the simulated traffic. Though in this case, it would be important that the real traffic be transformed in such a way that would make the real and simulated traffic indistinguishable to an IDS.

Tavallaee et al. [80] performed an analysis on the KDD Cup 1999 dataset. In this analysis, they found two significant issues that plague the dataset. These issues affect the performance of the anomaly detection systems, which in turn results in an inadequate evaluation of said systems by researchers. The first issue is the vast amount of redundant records in the KDD Cup 1999 dataset. Training a classifier on this dataset cause it to be biased towards these redundant (frequently-occurring) records, which makes the classifier much less likely to learn from records that occur less frequently. Furthermore, these duplicate records in the test set will cause the classifier to present a much higher detection rate for these specific records than for less frequent records. Tavallaee et al. solved this issue by removing all duplicate records and keeping only a single instance of each record. They also found that simply using accuracy, detection, and false positive rates for evaluation when using the KDD Cup 1999 dataset is an unsuitable approach for evaluating an anomaly detection system. They discovered this by learning 21 different classifiers on random subsets of the dataset, and then having these classifiers label all the records in the training and test sets. All 21 classifiers were able to correctly label 98% of the training set and 87% of the test set. To address this second issue, the NSL-KDD dataset is made more "challenging". Tavallaee et al. do this by selecting more “difficult” records and less “easy” records from the KDD Cup 1999 dataset. This record “difficulty” is determined by the number of classifiers that were able to correctly label the record. An easy record would be one that is correctly labeled by all 21 classifiers, while a difficult record would be one that is correctly labeled by a single one. However, since the NSL-KDD dataset’s origins can be traced back to the DARPA 1998 dataset, Tavallaee et al. admit that it still contains some of the problems that were described by McHugh [40].

Others opt not to generate synthetic network traffic, but instead to directly capture traffic from a small test network. Labib and Vemuri [29] test their developed IDS with data collected from their small private network. This network is made up of two subnets. The first subnet contains a single attacker and “tens of other hosts”. The second subnet contains only two hosts, both of which are attackers. They then proceed to train and test their IDS using the collected data. This collected data, however, is far from realistic. The network is not connected to the Internet, nor is there any background traffic present in the used dataset. Under minimum load, a host on this network receives an average of one packet per second. Furthermore, when extracting the source and destination IP addresses from the packets, only the two trailing bytes of the IPs are selected to form the dataset. The rationale behind this was that, since their private network is of Class C, the leading two bytes of the IP addresses will never change, and these two “redundant” bytes would negatively influence the performance of their classifier. Mirsky et al. [47] also chose to construct their own sample networks and using the traffic produced in those networks as training and test data. They built two different networks: one containing eight networked surveillance cameras, and another containing nine IoT devices and three PCs. These networks, too, are rather unrealistic, and will produce very uniform traffic. The first network, for example, will only ever produce “surveillance camera-like” traffic. Any deviation from this “surveillance camera-like” traffic, be it malicious or not, will be more likely labeled as an anomaly. Likewise, the latter network will be saturated with IoT traffic. This will likely have similar implications as the surveillance camera network. This points to a somewhat lacking evaluation, since the network traffic used for training is not realistic. As for the attack traffic, Mirsky et al. perform a series of attacks on the networks and capture that traffic at a specific point in the networks. Attacks include SYN floods, fuzzing, and Mirai infection, among others.

Collecting data from a small private network is much easier than doing the same thing in a bigger, noisier network. Testing hypotheses using this data might even produce promising results. However, results obtained from small environments are seldom the same when the same experiments are repeated in a larger environment [73].

Due to security and privacy concerns, releasing collected network traffic is not a straightforward process. Sensitive data might be present in the network traces, for instance. Having publicly available realistic data is something that is of interest to many researchers in this field. To avoid these security and privacy issues, people that have captured some amount of network traffic choose to anonymize it before releasing it to the public. However, this anonymization is not necessarily an effective measure [11]. Anonymization of network data significantly reduces the data’s usefulness for research [30, 56]. According to Killourhy and Maxion [27], this is because by anonymizing traffic, one very often removes key information that IDSs use to detect anomalies (e.g. IP addresses, packet payloads).

Sommer and Paxson [73] state that for properly evaluating anomaly detection systems, obtaining “appropriate data” is the most important issue. The researcher should strive to acquire a dataset that contains real network traffic. This network traffic should also be collected from network that is as large as possible. If possible, different datasets collected from different networks should be used. They state that working with actual traffic increases the quality of research, since one can immediately observe the performance of the studied IDS in a real-world scenario.

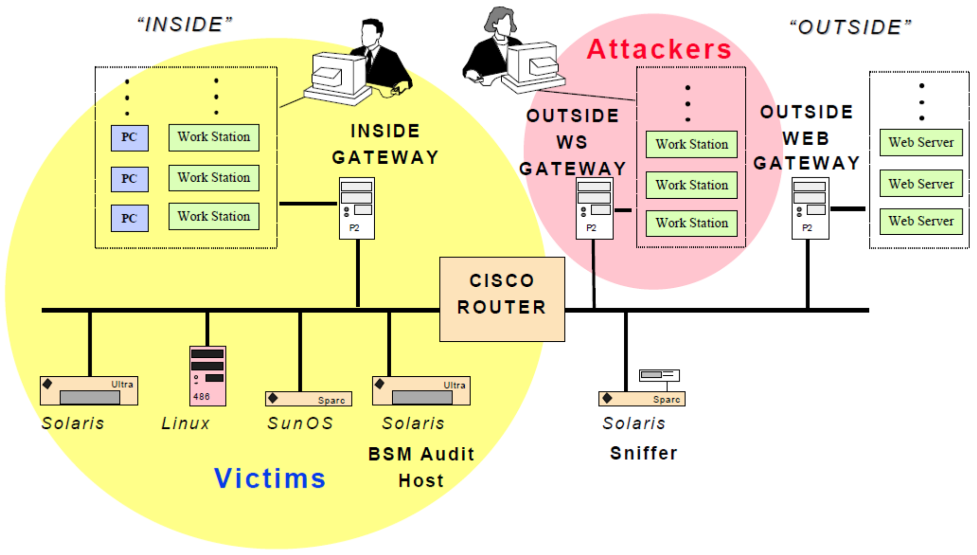


Figure 2.1: Diagram of the network in the DARPA datasets [33]

2.5. Summary

Much research has been done in the field of intrusion detection. Many different machine learning techniques have been applied to this problem, with many researchers claiming to have created highly effective IDSs. These "highly effective" IDSs are designed and evaluated using methodologies and datasets that have been heavily criticized by others. For instance, most of the papers discussed in Section 2.2 use some flawed dataset, as the papers by McHugh et al. [40] and Mahoney and Chan [36] have demonstrated. The rest use data from small, private networks, which is inadvisable, according to Sommer and Paxson [73].

There seems to be large disconnect between these two groups of researchers. One group criticizes current methodologies and suggests more suitable alternatives, while the other group never seems to adjust their approach to anomaly detection research in order to improve the quality of their work.

None of this research has ever been revisited in order to test how the same methodologies and technologies will perform in a real-world scenario, with real network traffic, as Sommer and Paxson suggest [73]. This is unfortunate, since much of this research may contain erroneous results and conclusions. Future research is then built on top of these possibly erroneous conclusions, as many researchers will not doubt the validity of published research articles. It is therefore in the interest of the entire field of anomaly detection to verify the findings of such published research.

3 Case study

From the background literature presented in Chapter 2, we see many different approaches and methods being developed to resolve a common task, namely automatic intrusion detection. However, many of these methods use data that is flawed [36, 40] for not only the training of the classifiers, but the evaluation as well. The fact that these inadequate methodologies and data are used for the realization of academic research raises questions about the validity of said research.

It is therefore interesting to see whether these approaches are able to hold their own when correctly tested in a real-world environment. By using real-world network traffic data, we will be able to appropriately evaluate the developed approaches, as well as the validity of the authors' conclusions.

3.1. IDS selection

The first thing that is necessary for this research is an IDS that we can study. Section 2.2 lists a number of IDSs that were developed over the years. We will select one of these IDSs for this project. This choice is made with two factors in mind: the age of the system, and its ease of use.

We prefer the IDS to be as recent as possible. Using recent research ensures that the techniques used are up to date with current developments in the field of anomaly detection. The two most recent works discussed in Chapter 2 are the systems proposed by Shone et al. [70] and Mirsky et al. [47]. Both published their research early 2018. Coincidentally, both use autoencoder-based approaches for the design of their IDS.

The final selection will therefore be based on each system's ease of use. Due to the time constraints set upon this project, we very much favor a system that is easy to set up and easy to use. The system by Mirsky et al. [47] clearly has the advantage over the system by Shone et al. [70] Shone et al. have not made their system publicly available, so trying to obtain it would be unnecessarily time-consuming. On the other hand, the IDS by Mirsky et al. is open-source, and they have made it publicly available on GitHub [44]. The public availability of the source code makes it a straightforward process to install the system on our machines. Furthermore, this system also takes its input data in raw pcap format, meaning that newly captured network traces can easily be fed into the system without the need for any data transformation or feature extraction. Thus, the IDS chosen for this research is the Kitsune IDS by Mirsky et al. [47].

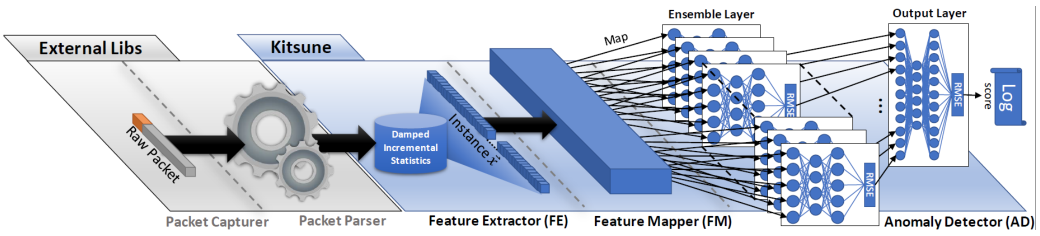


Figure 3.1: Architecture of Kitsune [47]

3.2. Kitsune

The Kitsune system is not simply a classifier, but includes an entire framework of functionality. The system uses autoencoders to learn the "normal" behavior of the network in order to, later on, detect deviations from this "normal" behavior.

An autoencoder is a type of artificial neural network that is trained in an unsupervised manner to create a representation of its input [32]. Explained simply, an autoencoder encodes the data it receives as input as an object of lower dimensionality, after which it tries to reconstruct the higher-dimensional object as accurately as possible.

Its functionality comprises the following elements:

1. Feature Extractor;
2. Feature Mapper; *and*
3. Anomaly Detector.

See Figure 3.1 for an illustration of Kitsune's architecture.

3.2.1. Components

This section will discuss the aforementioned components of Kitsune, as well as some of the relevant technical details.

Feature Extractor

Kitsune does not directly operate on raw packet data. Instead, as its name suggests, the Feature Extractor component extracts a collection of features from every captured packet. Packets are parsed and certain information is extracted from them. Specifically, the following fields are selected:

- IP version (IPv4/IPv6);
- packet timestamp;
- frame length;
- source and destination IP;
- source and destination MAC address; *and*
- transport, internet, or link layer protocol [79].

It is from these fields that the eventual features are actually constructed from.

Mirsky et al. emphasize the importance of analyzing the temporal features of network traffic to find anomalies and possible intrusions. Instead of using packet windows to compute these statistics, Kitsune uses packet timestamps to create damped windows, using a technique that Mirsky et al. call *Damped Incremental*

Statistics. They claim using a conventional window will not scale due to memory usage, which is one of the reasons for the usage of their damped windows. For each packet that arrives, the previously mentioned fields are retrieved. These fields are then used to update the statistics in the damped window.

The goal of damped windows is to maintain the temporal statistics of a collection of packets within a given time window, but have the weight of older values decrease as time goes on. Temporal statistics are computed for five different time windows λ: 100ms, 500ms, 1.5s, 10s, and 1min ($\lambda = 5, 3, 1, 0.1, 0.01$). For every one of these time windows, Kitsune maintains a collection of incremental statistics tuples for each of the following:

- every source MAC-IP address (denoted **SrcMAC-IP** [1]);
- every source and destination IP address pair (denoted **SrcIP**);
- every connection between a source IP and destination IP (denoted **Channel**); *and*
- every connection between a source IP-port pair and destination IP-port pair (denoted **Socket**);

For all records except the SrcIP records, Kitsune extracts from packets both the packet size and timestamp. The purpose of the SrcIP records is to capture jitter behavior between different hosts, and therefore only extracts timestamps. The tuples corresponding to SrcMAC-IP and SrcIP are used to compute one-dimensional (1D) statistics. The tuples corresponding to Channel and Socket are used to compute two-dimensional (2D) statistics. This naming (1D and 2D) refers to the number of tuples involved in the statistics computations: the 1D statistics involve a single one of these tuples, while the 2D statistics are computed using two different tuples. The tuples are defined as

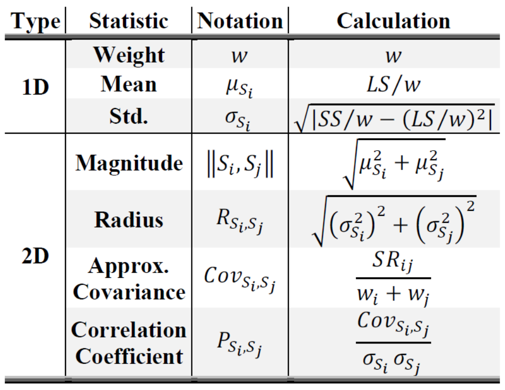
$$IS_{i,\lambda} \leftarrow (w, LS, SS, SR_{ij}, T_{last}),$$

where λ is the time window, w is the weight, LS is the linear sum of the packet sizes, SS is the sum of the squares of the packet sizes, SR_{ij} is the sum of residual products between the tuples $IS_{i,\lambda}$ and $IS_{j,\lambda}$, and T_{last} is the timestamp of the most recently received packet that corresponds to the given tuple. SR_{ij} uses data from two different tuples, and is only present in the tuples used for 2D computations: Channel and Socket.

Kitsune creates or fetches a tuple for every packet that it receives. It then uses these tuples to compute a total of 23 statistics to capture the behavior of the network for each of the given time windows. Thus, a total of 115 features are extracted. See Figure 3.2 for the full list of features extracted from the tuples.

It is important to note that the usage of 2D statistics makes Kitsune detection context-dependent. By this, we mean that previous behavior between two hosts, if there is any, is taken into account when evaluating traffic. Therefore, it matters whether traffic is evaluated within the context of other network traffic or out of context. For example, say there is a malicious host that had previously interacted with the network in a strictly benign manner. If this host suddenly launches an attack, Kitsune will evaluate these malicious packets differently than in a situation where the malicious host had not previously interacted with the network at all.

[1] notation taken from Mirsky et al.[47]



Type	Statistic	Notation	Calculation
1D	Weight	w	w
	Mean	μ_{S_i}	LS/w
	Std.	σ_{S_i}	$\sqrt{\lvert SS/w - (LS/w)^2 \rvert}$
2D	Magnitude	$\lVert S_i, S_j \rVert$	$\sqrt{\mu_{S_i}^2 + \mu_{S_j}^2}$
	Radius	R_{S_i,S_j}	$\sqrt{\left(\sigma_{S_i}^2\right)^2 + \left(\sigma_{S_j}^2\right)^2}$
	Approx. Covariance	Cov_{S_i,S_j}	$\frac{SR_{ij}}{w_i + w_j}$
	Correlation Coefficient	P_{S_i,S_j}	$\frac{Cov_{S_i,S_j}}{\sigma_{S_i}\sigma_{S_j}}$

Figure 3.2: Statistics computed by Kitsune using the incremental statistics tuples [47]

Feature Mapper

The features that are extracted from the network traffic are passed to the Feature Mapper. The Feature Mapper then processes these features in order to produce the input for the Anomaly Detector's ensemble of autoencoders.

Firstly, the features that are given as input are clustered together using correlation between features as the distance between them, which they define as

$$d_{cor}(u, v) = 1 - \frac{(u - \overline{u}) \cdot (v - \overline{v})}{\|(u - \overline{u})\|_2 \|(v - \overline{v})\|_2},$$

where $\bar{u}$ is the mean of the elements in vector u, and $u \cdot v$ is the dot product [47].

Out of the 115 features extracted, every cluster will be constructed in such a way that it contains at most m features, where m is a user-defined parameter. So, each autoencoder will have a maximum of m inputs. This parameter has an effect on the complexity of the autoencoder ensemble, and therefore also on the speed and performance of the system as a whole. Mirsky et al. state that a smaller m will "usually" make Kitsune perform better than a higher value of m, although processing speed will decrease significantly [47]. This makes sense, since a smaller m will lead to a larger amount of clusters, which which will in turn lead to an increase in complexity.

The output of this component is a list of these feature clusters.

Anomaly Detector

The Anomaly Detector is made up of two layers of autoencoders: the Ensemble Layer, and the Output Layer. The Ensemble Layer takes the list of feature clusters as input, while the Output Layer takes the output of the Ensemble Layer as input.

Each autoencoder in the Ensemble Layer will be assigned a single cluster of features to process. The features are mapped to a subspace and then reconstructed by the autoencoder. After the mapping and reconstruction phases, the root-mean-square error (RMSE) is computed between the original features that were the inputs of the autoencoders and the reconstructed features. A lower RMSE indicates an accurate reconstruction of the input features, which is the objective of the autoencoders. All the RMSE values from the Ensemble Layer are collected, and provided as input to the Output Layer.

The Output Layer is a single autoencoder that has the same amount of inputs as the Ensemble Layer has autoencoders. Every RMSE value from the Ensemble Layer is fed into the Output Layer. Initially, this final autoencoder is trained on the RMSE values that are computed with normal network traffic. When actually in operation, the Output Layer tries to reconstruct the original RMSE values. It then computes takes the original RMSEs and the reconstructed RMSEs and computes one final RMSE value, which will serve as the packet's anomaly score.

The previous was a brief overview of Kitsune's inner workings. For a more detailed description of the mathematics and algorithms behind Kitsune, see the paper by Mirsky et al. [47].

3.2.2. Experimental setup

For the purpose of testing Kitsune, Mirsky et al. set up two different networks from which to collect network traffic and inject attacks [47]. The first network was an IP camera video surveillance network. The second one was a WiFi network containing a number of Internet of Things (IoT) devices, and three PCs.

The video surveillance network consisted of two sub-network containing a switch each to which four video cameras were connected. Each sub-network had a site-to-site VPN tunnel established to an third, external, network that contained a DVR server to which the cameras streamed their data. The attacks were performed from several different locations on this network, such as directly via a switch in a camera sub-network, or via physical access to the DVR network's VPN router.

The IoT network consisted of nine IoT devices, such as security cameras, doorbells, and thermostats, among others. Additionally, the network contained three different PCs, one of which was situated on a separate subnet. The IoT devices were connected to the network via WiFi, while the PCs used an Ethernet cable to establish a connection to the network. On this network, the attacker executes attacks via a connection to the WiFi router, although it is not stated in the paper whether access was wired or wireless.

Mirsky et al. placed the Kitsune instance at specific places within the network. The effect of this is that not all traffic flowing through the network will be picked up by Kitsune. The Kitsune instance would also collect all the network traffic that it received.

Every experiment only lasted for a few hours. During these few hours, Kitsune would train itself on the unaltered, benign traffic that flowed through the network. After training on the first million packets received, it would start evaluating the traffic in order to find anomalies in the traffic. After some time evaluating benign traffic, Mirsky et al. would perform the attack. When the attack had been completed, the experiment would be brought to a halt.

3.2.3. Attack traffic

Mirsky et al. performed their attacks one at a time. In other words, Kitsune's detection performance was only ever tested on one attack per evaluation. Every attack is independent from the rest. There are nine different types of malicious activities that Mirsky et al. performed on their networks. Mirsky et al. split these activities up into four groups: reconnaissance, Man-in-the-Middle, Denial of Service, and Botnet Malware. The four groups will be discussed below.

Reconnaissance

This group contains two types of malicious activities: OS scanning and fuzzing. The OS scanning consists of an attacker using Nmap [50] to scan the network and its hosts and operating system in order to find any possible vulnerabilities.

The fuzzing attack uses SFuzz [10] to probe the surveillance camera's web servers. It sends a collection of malformed CGI commands to the web servers in hopes of triggering an error and find a vulnerability.

Man-in-the-Middle

These attacks are a special case. As mentioned in Section 3.2.2, the Kitsune instance would not receive all the network traffic that flowed through the network. Mirsky et al. state that, although the actual malicious traffic would not be seen by the Kitsune instance, statistical changes in the network's behavior would be picked up by Kitsune due to its usage of incremental statistics. They state that network behavior caused by Man-in-the-Middle attacks, such as packet jitter, can be detected this way.

There are three types of Man-in-the-Middle attacks that were performed on the networks. The first is video injection. Since the surveillance cameras maintain a constant stream of video traffic to the DVR server, it is possible for a malicious actor with access to the network to access this traffic. In this attack, Mirsky et al. inject a recorded video clip into a camera's live video stream.

The second attack is an ARP poisoning attack. They poison ARP caches in order to intercept the network traffic between a surveillance camera and the DVR.

The third is the usage of an active wiretap. It is performed by installing a network bridge on an exposed network cable. This allows the attacker to intercept all network traffic flowing across the network bridge.

Denial of Service

Mirsky et al. perform three types of DoS attacks against their network, the first of which is an SSDP flood. The surveillance cameras use SSDP to advertise themselves on the network. This can be exploited by attackers who want to overload a host. The attacker sends the cameras an SSDP request with the IP address of the DVR as a spoofed address. Once received, the cameras "reply" to the DVR with a much larger SSDP response. This overloads the DVR server.

Second is a SYN flood. Mirsky et al. overload a camera's web server by sending it a large number of SYN packets. The result of this attack is the disabling of the camera's live video stream.

Lastly is an SSL renegotiation attack. Using the RSA cryptosystem, decryption takes significantly longer than encryption. This is because the private exponent is computed as the modular multiplicative inverse of the pubic exponent, therefore making it much larger than the public exponent [88]. Due to this imbalance in processing speeds, an attacker can flood an SSL service with SSL renegotiation requests and disable it by forcing it to perform a huge amount of expensive decryptions. Mirsky et al. perform this attack on one of the cameras, which disables its live video stream.

Botnet Malware

Mirsky et al. perform this attack not on their surveillance camera network, but on their IoT network. A malicious actor connects to the network's WiFi access point. He then scans the network for available Telnet services. When one of these services is found on an IoT device, the attacker connects to it using default admin credentials and installs on it the Mirai malware. The infected IoT device then starts scanning the network using ARP requests in order to find more live hosts to infect.

3.2.4. Evaluation

As discussed in Section 3.2.1 of this chapter, the final output of Kitsune is an RMSE value that serves as an anomaly score for the currently processed packet. The greater this RMSE value, the greater the anomaly [47]. A RMSE by itself, however, is not sufficient to classify traffic as anomalous. A threshold value needs to be chosen first. Any packet that receives a final RMSE score below this threshold is considered benign. Packets receiving scores above the threshold are classified as anomalous.

Naturally, the choice of threshold value will determine the accuracy of classification, and hence the overall performance of the classifier itself. Therefore, it is important that this threshold is not chosen in an arbitrary manner. A way of choosing this threshold is to minimize the number of false positives generated by the classifier, since intrusion detection systems become practically unusable with even a minute false positive rate [3]. This is the approach that is taken by Mirsky et al. for determining their threshold value [47].

For every attack, they select two different thresholds that would give them fixed false positive rates of 0 and 0.001 when classifying the *benign* traffic. We will refer to these thresholds as $FPR = 0$ and $FPR = 0.001$, respectively. After selecting these two thresholds, the portion of network traffic containing the attack itself is classified. Applying the $FPR = 0$ and $FPR = 0.001$ thresholds to the malicious traffic, Mirsky et al. compute two statistics for each threshold (four in total): the true positive rate (TPR) and false negative rate (FNR). In addition to the TPR and FNR, Mirsky et al. use two ROC curve [83] statistics: Area Under the Curve (AUC) [17], and Equal Error Rate (EER) [69]. These statistics are computed, not only for Kistune, but for other IDSs and algorithms, such as Suricata [76], Isolation Forest [35] and pcStream [46]. Kitsune's overall performance is then determined by comparing it to the performance of the other algorithms on the same attacks.

4 Data collection and processing

Usage of inadequate data for the development of IDSs has been highlighted in Chapter 2. As stated by Sommer and Paxson [73], one should aim at acquiring a dataset that contains real network traffic from an environment that is as large as one can get their hands on. This is an important aspect of data collection, because it cannot be assumed that analysis of small environments will yield the same results when such an analysis is performed on larger and more realistic environments [72]. Performing one's research on real network traffic, as opposed to pre-collected data or data collected from an artificial network, improves the quality of the research and its results, since the evaluation of the system will demonstrate its performance in a real-world scenario [73].

4.1. Obtaining appropriate data

For this research, we will not use pre-collected datasets, such as the DARPA 1998/1999 [31] or KDD Cup 1999 datasets [85]. Neither will we be collecting network traffic from small experiment networks, as Labib and Vemuri [29], and Mirsky et al. [47] have done. Instead, we will follow the recommendations by Sommer and Paxson [73]. They state that it is important to a research project that datasets be collected from a real-world environment.

We must find and obtain access to as large a network as we can. If this network belongs to a third party, agreements need to be made about proper storage and usage of any captured network traffic. Once the agreements have been made and network access has been provided, we will start capturing network traffic. Several weeks of network traffic will be collected in order to have enough variety in the eventual network traces. This variety in the traces can include many unique IP addresses, usage of many different protocols, and variety in packet sizes, among many others. Special care needs to be taken of the possible privacy concerns that may arise. In any case that involves sensitive data, measures need to be taken in order to secure this data. If necessary, this sensitive data should be anonymized before being processed.

As stated by Sommer and Paxson [73], obtaining ground-truth data is imperative for proper IDS evaluation. Without knowledge of the network that is used, it is impossible to obtain this ground-truth. Therefore, cooperation between us and the network administrator is necessary. The network administrator will need to provide a thorough description of the network. This includes the hosts, which services are running on which ports, and how the hosts communicate between themselves and with external hosts on the Internet. When we have received the description of the network, we will begin filtering the captured network. The goal is to obtain ground-truth data by filtering network traffic so that we end up with two sets of network traffic for which we know with absolute certainty that one contains only normal traffic and the other contains only illegitimate or malicious traffic.

The filtering will be performed by creating a static rule-set from the provided network description. It is likely that this network description will be incomplete, meaning that the rule-set created from the description will be incomplete as well. Filtering the network traces using this rule-set will split the data into three categories: legitimate traffic, illegitimate traffic, and unclassified (i.e. traffic for which no rule currently exists). The next part of the filtering must be an iterative process. Of the unclassified traffic, we will select a subset and discuss this traffic with the network administrator. The network administrator will label this traffic as legitimate or illegitimate. We will then update the network description and the rule-set, and, once again, filter the network traces. This process is repeated until the unclassified traffic disappears and the rule-set is able to filter all of the captured network traffic into either the legitimate or illegitimate category. In the case that ground-truth cannot be established about a specific type of network traffic, it will be removed from the dataset. Ground-truth is essential for anomaly detection datasets, so data of which we are uncertain cannot be included in our datasets.

4.2. Network description

The data that will be used in this research is taken from a real-world network. This section will describe the layout of this network, as well as the components that it is made out of.

The services provided by this network are used by a multitude of users from different places all over the world. While itself a relatively small network compared to large organizations, many types of traffic that are also found in such large organizations, such as mail, web, and DNS traffic. The presence of these different types of network traffic makes this used network a much more realistic one than the artificial network created by Mirsky et al. for their paper [47], since that network mostly consisted out of IP cameras that output video traffic.

In addition to using IPv4, this network also accepts the newer IPv6 and can use it to communicate. All components of the network have both an IPv4 address and an IPv6, with the only exception being the private DNS server. This private DNS server is only assigned an IPv4 address.

The network used in this research contains (i) a gateway, (ii) a mail server, (iii) a web server, (iv) a public DNS server, (v) a private DNS server, (vi) a shell access server, and (vii) a web server requiring authentication.

While every component on the network is a separate and independent unit, in actuality, they are all virtual machines (VM) running on the same physical server. The gateway VM is the only component that is directly connected to the Internet. Its job in the network is to provide the remaining virtual machines with Internet access, and to allow for communication between said virtual machines. It does this by means of a virtual bridge. This virtual bridge is set up on the gateway VM, and is configured to contain all of the other virtual machines as interfaces. See Figure 4.1 for a diagram of the network.

All virtual machines on this server are running OpenBSD 6.3 [52] as operating system. OS distribution updates are downloaded from an OpenBSD mirror[1]. An SSH ser-

[1]`https://ftp.halifax.rwth-aachen.de/`

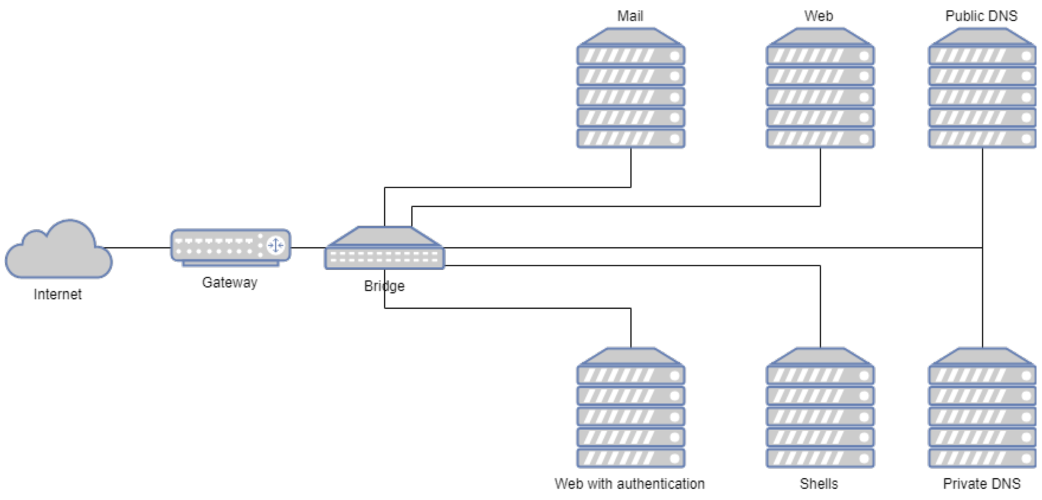


Figure 4.1: Diagram of the network

vice is running on every virtual machine but the gateway VM. Additionally, all virtual machines except the gateway and shell access server use the ACME protocol to automatically renew their *Let's Encrypt* SSL certificate. The characteristics of the different components of this network will be discussed in the sections that follow.

4.2.1. Gateway

The gateway provides Internet access to the network. No filtering of firewall is set up on this virtual machine. Therefore, any type of traffic is allowed to enter and leave the network.

4.2.2. Mail server

Being a mail server, this VM offers services such as SMTP on port 25 and IMAP on port 143. In addition to these services, also offered are its encrypted alternatives that connect using SSL: SMTPS and IMAPS on ports 465 and 993, respectively. A mail submission agent (MSA) running ESMTP runs on port 587. This MSA offers opportunistic TLS [20] using the `STARTTLS` command in order to encrypt the otherwise plaintext SMTP traffic entering and leaving this port. Although these ports are open and visible to the Internet, emails will only be successfully received if recipient is member of the local domain. The mail server also offers a web interface that allows users to access their email through their web browser. This web interface can be accessed on port 80 or port 443, although users attempting access through port 80 will be redirected to port 443.

4.2.3. Web server

The web server hosts a website that is publicly accessible on the Internet. As with the mail server, people trying to access this website through port 80 will be automatically redirected to port 443.

4.2.4. Public DNS server

This virtual machine offers a DNS service on port 53. DNS requests are accepted in both TCP and UDP format, and replies are sent back in the same manner. A web interface is available for management purposes, which is accessible through ports 80 and 443. Every attempted connection to port 80 is also redirected to port 443.

4.2.5. Private DNS server

The private DNS server provides the same services and functionality that the public DNS server does. The only difference is that DNS requests to this server originate only from within the local network. Outside access to this server is prohibited.

4.2.6. Shell access server

The only service this server itself provides is SSH on port 22. This port is accessible from the Internet, although only authorized users are allowed to utilize it. The users themselves then have the ability to run or access services that produce other network traffic, such as ICQ, XMPP, and IRC.

4.2.7. Web server with authentication

This virtual machine is also a web server. In contrast to the public web server, however, this server uses HTTP Basic authentication in order to grant users access to the web page. All HTTP requests to port 80 are redirected to port 443. This is necessary for confidentiality, since HTTP Basic authentication does not encrypt user credentials before sending them to the web server.

4.3. Capturing network traffic

For the purpose of collecting the network traffic necessary for this research, we made use of *tcpdump*, which is a packet capture and analysis tool [81]. A tcpdump instance was set up on the gateway VM to capture all the traffic that passes the virtual network bridge. This includes incoming and outgoing Internet traffic, as well as internal communication between the virtual machines themselves within the network.

When writing network traffic to a file, tcpdump writes raw packets straight to disk [39]. These packet captures are saved to a *pcap* file, which contains the captured sequence of raw packets. This allows the tcpdump instance to run as fast as possible in order to minimize the amount of dropped packets [39].

This tcpdump instance ran uninterrupted for a period of seven weeks, or 49 days. During this time, a total of 33.7 GB of traffic was captured, containing approximately 240 000 unique IP addresses. Per day, the average packet capture file is around 690 MB in size and contains 12 170 unique IP addresses.

4.3.1. Sanitation

This is a real-world network with actual users. Given the possibility that the captured network traces contain the users' sensitive data, we need to take their privacy concerns into consideration. This means that the captured data must be sanitized in order to remove all of the potentially sensitive information.

This network contains two servers that require network traffic sanitation. These are the mail server and the shell access server.

The mail server makes use of SMTP on port 25 for sending emails. Plain SMTP does not make use of encryption. Hence, capturing this traffic will also capture the contents of the users' emails as plaintext. In order to prevent privacy issues, any packet with port 25 as either source or destination port is stripped of its TCP payload. The result of this sanitation is that all SMTP packets are transformed into empty TCP packets, meaning TCP packets with no payload.

The shell access server allows users to set up their own messaging services or connect to external messaging services. These messaging services include ICQ, XMPP, and IRC. Network traffic created by these messaging services are not necessarily encrypted [62]. IRC, for instance, is a plaintext protocol. Anyone with access to the network can also access the information that is sent via this protocol. Several clients that allow for encrypted IRC traffic are available [43, 66]. However, encryption is neither mandatory nor enabled by default, so we must assume that captured IRC traffic will be in plaintext. While XMPP had no encryption capabilities originally, such functionality has been developed. The developers of XMPP have been pushing for mandatory encryption on XMPP, and in May of 2014 they signed an agreement with a large number of XMPP operators to ensure they cooperate [57]. There are not all of the XMPP users globally, though, and the users of this server, or the external XMPP servers they use, might not necessarily adhere to these rules. Even though ICQ offers end-to-end encryption for all voice and video calls [38], messaging uses no encryption whatsoever [37]. Based on this information, the network traffic that corresponds to these services must be sanitized.

See Table 4.1 for an overview of the sanitized hosts and ports.

Table 4.1: All hosts and ports that were sanitized.

HOST	PORT	PROTOCOL
Mail server	25	SMTP
Shells server	5190	ICQ
	5223	XMPP
	6667	IRC
	6697	IRC
	9999	

Sanitation was achieved by creating a Python script that utilizes the packet manipulation library *Scapy* [68]. Every packet that is captured is processed by this script. The script checks whether a packet's source or destination and service matches one of the aforementioned services that handle sensitive information. If it does, it removes the TCP payload from the packet and saves the rest of the packet to disk.

4.4. Filtering network traffic

After capturing and sanitizing the captured traffic, the next step is to start the filtering process. The filtering is performed in order to separate the malicious traffic from the benign. This allows us to obtain the ground-truth data that is necessary to properly train an IDS.

The filtering process is not only useful for research purposes, but potentially beneficial to the network administrator as well. First of all, knowing exactly what type of traffic is present in a network is useful to a network administrator. It is possible that the network administrator might have overlooked certain configuration errors. Alternatively, unknown network traffic could indicate a possible network intrusion or a compromised machine.

In the paper by Mirsky et al. [47], Kitsune is described as an "NIDS which can learn to detect attacks on the local network, without supervision." Unsupervised learning, as opposed to supervised learning, does not require class labels in order to build a classification model [18]. In this specific case of Kitsune, it is implied that Kitsune need not know whether the traffic used for training is benign or malicious, and that it will be able to differentiate between normal and abnormal traffic on its own. In the penultimate section of the article, however, the authors clarify: "When first installed, Kitsune assumes that all traffic is benign while in *train-mode*." Due to this, any malicious actors already present in the network will be able to circumvent Kitsune's attempts at detection. Technically, Kitsune is an unsupervised system due to its usage of autoencoders [32] for its learning and classification process, which do not need explicit labeling of data. In practicality, however, training on *just* normal data suggests that any candidate data that will be used for training must first be split up into benign and malicious traffic before being fed into the system. This is especially the case in the current *Bring Your Own Device* [6] era. People bring their own devices to the workplace and use them to access the company network. In turn, these devices bring with them a large collection of software that produces lots of new network traffic that would otherwise never be seen in the company network. Much of the traffic within a company network will undoubtedly be benign, but one cannot say with certainty that *all* of the network traffic is benign without performing some sort of analysis on said network traffic. Consequently, any real-world network traffic must be analyzed and filtered before being used as training data.

The objective of this research is to test Kitsune's classification performance on a real-world network that is noisier and contains a larger variety of types of network traffic. Additionally, for anomaly detection research, it is imperative to obtain reliable ground-truth data [73]. Therefore, the traffic that is captured cannot simply be used immediately to train a classifier. The traffic collected from this network must be filtered in order to train Kitsune on ground-truth and completely benign traffic. Any presence of malicious or unwanted traffic in the training data will influence the classifier's performance.

A Python script was created to perform the filtering process. The script utilizes tcpdump's filter rule functionality. This allows the user to extract a specific portion of network traffic from either a real-time network traffic stream, or a pcap file containing previously collected traffic. This script carries out a number of tasks in order to extract ground-truth benign traffic.

Certain services offered by the network require authentication before usage is possible. Therefore, we cannot assume that all network traffic corresponding to this service is benign. See Table 4.2 for the full list of these services. Access to these services is logged, and these logs enable us to differentiate between benign access and unwanted or malicious connection attempts. The IP addresses that attempt to access the services are extracted from the logs and separated into valid and invalid IP addresses. This separation is based on whether the host has successfully completed authentication on the corresponding service. Once extracted and separated, only the hosts with valid IP addresses have their traffic classified as benign. It is important to note, however, that the validity of hosts is not global. That is to say, if a host is considered benign for a certain service on a single host, it is not assumed that this host is benign on all the other hosts. For instance, a host with a valid connection to the SSH service on the mail server does not mean that any connection attempts to the web server's SSH port will be considered benign. Furthermore, there exists the possibility that a host has their IP address classified as both valid and invalid, e.g., a user successfully logging in to their account after a number of failed attempt. It is difficult to make a distinction between a legitimate user forgetting or mistyping their password and a malicious user attempting to break into an account. Since obtaining ground-truth benign data is necessary, and determining the ground-truth of this specific case is difficult, such hosts, and their corresponding traffic, are considered malicious.

A drawback of using service logs for this type of filtering, is that logging applications do not log *all* activity on its port. For instance, OpenSSH, which provides an SSH service on all hosts, logs the “beginning, authentication, and termination of each connection” [53]. It does not, however, log port scans or incomplete connections. Any traffic that does not produce a complete connection will not be logged, and will, therefore, remain unclassified. There is no infallible method of determining the nature of such network traffic, and since the goal is to obtain ground-truth data, we cannot make any unfounded assumptions about said traffic. For this reason, this remaining network traffic is discarded from the dataset altogether.

After service-specific filtering has been completed, the general behavior of the network is considered. The network description from Section 4.2 that is provided by the administrator of the network must be translated into tcpdump’s filter rule format. These rules provide a model for the network traffic that is expected to be present in the network. The traffic remaining from the previous step is filtered using these filter rules. All of the traffic that fits into this model is assumed to be benign traffic.

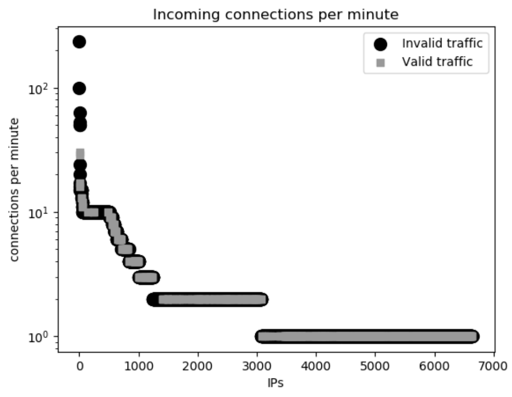
Table 4.2: Services that require authentication per server

SERVER	PORT	SERVICE
Mail	22	SSH
	80/443	HTTPS
	143	IMAP
	465	SMTPS
	587	ESMTP
	993	IMAPS
Web	22	SSH
Public DNS	22	SSH
	80/443	HTTPS
Shells	22	SSH
Web with auth	22	SSH
	80/443	HTTPS

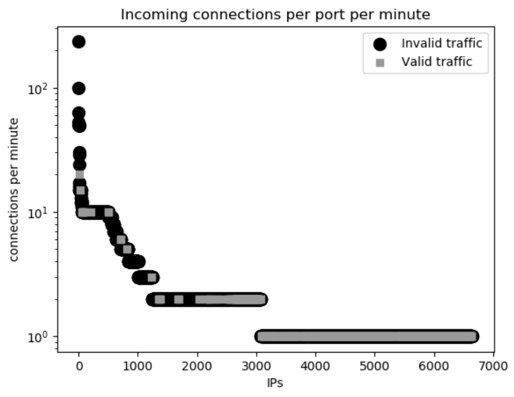
Much of the captured traffic, though, does not fit into the model. Not all of the traffic can be explained by the constructed filter rules. This remainder must be filtered further until all of the unknown traffic is accounted for. The following is a list of the types of traffic that can be found in the remaining traffic:

- port scans;
- ICMP traffic;
- exploitation attempts;
- additional legitimate traffic; *and*
- additional unknown traffic

The subsections below will describe the traffic from this list. We will also elaborate on the filtering processes for every type of traffic.



(a) Incoming connections per minute for every IP



(b) Incoming connections to a single port per minute for every IP

Figure 4.2: Attempted heuristics for port scan detection

TCP port scans

The majority of the remaining traffic was composed of several types of port scanning activities. These can be categorized into two main groups: TCP scans and UDP scans. The objective is to detect these port scans in order to separate the them from the rest of the traffic.

TCP connections are initiated with SYN packets [79]. Several statistics can be created by counting the number of SYN packets sent out by hosts. So, initially, port scan detection was attempted using two different statistics that were extracted from the SYN packet information. The first of the two is the number of incoming connections per minute per IP address. When performed aggressively, port scans attempt to initiate a large number of connections in a relatively short span of time. The objective was to depict that phenomenon with this particular statistic. The expectation was that hosts scanning the network would have a much higher number of connections per minute than legitimate hosts. Figure 4.2a illustrates this statistic as a scatter plot. As can be seen in the figure, however, there is no clear correlation between connection validity and the number of connections per minute. There is no way of separating valid and invalid connections based on this statistic alone.

The second statistic is the number of incoming connections to a single port. Figure 4.2b illustrates this statistic. It describes hosts scanning a single specific port across the network. This type of scanning is performed when looking for specific services or specific vulnerabilities. While certainly a far cleaner figure than Figure 4.2a, Figure 4.2b is not perfect either. Although most of the valid connections have a low amount of connections per minute, there are several valid connections that share the same amount of connections per minute as lots of the invalid connections. Just as with the previous statistic, it is impossible to divide the traffic into valid and invalid using this statistic.

One could argue that the thresholds could be set in such a way that the majority of the invalid traffic could be captured using these statistics. The problem with this argument is that it does not take into account the need for ground-truth. And data obtained using these statistics would therefore be useless for our purpose.

A form of TCP scans is the SYN scan. A side effect of a SYN scan is the creation of RST packets, either by the host performing the scan, or by the host that is being scanned [14]. The scanner creates a SYN packet addressed to a specific host and port, and sends it to the target host. In the case that the target port is open, the host will respond to the scanner with a SYN/ACK packet. The SYN/ACK packet tells the scanner that the port is open. After receiving the SYN/ACK packet, the scanner responds with a RST packet to close the connection. If the target port is closed, however, the scanned host does not respond to the scanner with a SYN/ACK packet, but responds with a RST packet instead. In both cases, a RST packet is generated by either of the two hosts. Also, since RST packets must be generated when a port receives packets for which an open connection does not exist [59], we can assume that no legitimate connections will produce RST packets. Hence, the presence of this RST packet can be used to detect SYN scans and the scanning hosts themselves. The filtering process is then as follows: 1) For every RST packet, collect the IP address of the remote host. 2) Extract all of the network traffic corresponding to the collected IP addresses. 3) Classify the extracted traffic as malicious.

Some port scanners, though, do not close the connections they open when scanning. Just as the previous case, the scanners send a SYN packet to a certain port on a server, who then responds to the scanner with a SYN/ACK packet. In the style of a SYN flood attack [7], the scanner does not to respond to the server with a RST packet, but instead opts to end the connection. This leaves the connection on the server's side in a half-open state [59]. The server assumes the sent SYN/ACK packet did not successfully arrive at its destination and retransmits the packet a number of times before finally closing the connection. Retransmissions of the SYN/ACK packets can be identified in the network traffic, and can be used to detect this type of port scans. The retransmitted SYN/ACK packets are extracted, and the IP addresses responsible for this traffic are collected. The traffic generated by these hosts is then extracted and classified as malicious.

The network traffic captures also contained a number of lone SYN packets originating from remote hosts and addressed to local servers. No matching SYN/ACK packets appear in the pcap file. Considering there is no firewall or packet filter active on neither the gateway nor the rest of the virtual machines, the SYN packets could have simply been dropped silently by the virtual machines. Alternatively, the tcpdump instance could have failed to capture these packets. Either way, lone SYN packets that do not attempt to set up a full connection can be indicative of port scanning, and, therefore, this traffic should be filtered as well. IP addresses are extracted from the SYN packets, and all traffic corresponding to those IP addresses is classified as malicious. As mentioned earlier, all TCP connections are initiated with SYN packets. By simply filtering out SYN packets, all SYN packets belonging to complete connections will be removed too, be they malicious or not. It is for this reason that it is important for this step of filtering to be performed last. This ensures that all other traffic has been accounted for, and that lone SYN packets are the only thing left to worry about.

UDP port scans

Seeing as how using connection statistics as in Figure 4.2 did not help with identifying TCP scans, that method was not attempted when trying to identify UDP port scans in the network traffic. Since UDP is a connectionless protocol, there is no need to use handshakes and establish a connection in order to exchange data. Given that there is no firewall or filter in place, hosts can just send out any arbitrary UDP datagram to a port listening for UDP traffic, and the port will receive it [79].

In this network, the only servers that accept UDP traffic are the public DNS server and the private DNS server. As these two servers are the only users of UDP traffic, the only legitimate instances of UDP traffic in the network are the following:

– Internal DNS traffic to and from port 53 on the private DNS server.
– Incoming DNS traffic to port 53 on the public DNS server.
– Outgoing DNS traffic from the public DNS server to port 53 on a remote server.

Nevertheless, it is wise to avoid using a single blanket filter that captures all but the aforementioned traffic. Doing so makes it impossible to find possible network configuration errors or intrusions that the network administrator is oblivious of. Hence, it is interesting to examine the specific instances of UDP scans, because flaws in the network can possibly be identified.

Outside hosts probe many different ports in hope of finding one that responds to the scan with a UDP packet of their own, thus informing the scanner of an open port. Many of these attempts are basic scans, but a great deal of them are targeted scans. Specific ports for which their corresponding services are known are singled out and receive UDP packets specially crafted for those particular services. Examples of these include LDAP (Lightweight Directory Access Protocol) on port 389, SIP (Session Initiation Protocol) on ports 5060 and 5061, and NetBIOS on port 137.

During the examination of the UDP scanning, we discovered an NTP (Network Time Protocol) service running on a server where no such service should be running. Upon further investigation by the administrator, it was determined that the NTP service was set up some time in the past for legitimate reasons. The service was originally supposed to be running temporarily, but nobody remembered to shut it down afterwards. It could be argued that this traffic should belong with the rest of the benign traffic, since the service served a legitimate purpose. However, this argument is outweighed by the fact that—by own admission of the network administrator—this service is not part of the network's normal operation. Just because this NTP service turned out to be benign in this particular case, there can be no certainty that no malicious actor will set up an such an NTP service in the future. Since the classifier should be trained with ground-truth benign data, placing this traffic with the rest of the benign data will bias the classifier and be of detriment to the results of classification. But because the NTP service is not malicious either, it cannot be placed with the rest of the malicious traffic. For this reason, all traffic corresponding to this NTP service was removed completely from the dataset.

The rest of the ports receiving UDP packets were examined, and no other irregular incidents were discovered. A filter rule was created for every one of these ports, so that the UDP scans get placed with the malicious traffic.

As none of the other probed services are running on any of the virtual machines, no UDP packets are sent back. Although no server responds to the probes with

UDP packets, closed ports respond to unexpected UDP packets with an ICMP Type 3 Code 3 packet (Destination unreachable: Destination port unreachable) [58]. ICMP traffic will be discussed in a later section.

Miscellaneous targeted port scans

In the captured traffic, there was one type of port scanning traffic that was neither TCP nor UDP. An outside host was scanning the network for services using SCTP (Stream Control Transmission Protocol). The network does not run any service that uses SCTP, so this is classified as malicious.

ICMP traffic

There are many types of ICMP packets present in the captured traffic. Since this network operates with both IPv4 and IPv6, both its respective ICMP versions can be found in the dataset. Different types of these packets serve different purposes. Some are part of normal network management traffic, while others can be indicative of malicious activities. Some ICMP traffic can also be used for purely malicious objectives.

When deciding what ICMP traffic is malicious, we checked out the IP addresses in the packet. The IP addresses were then compared to the lists of valid and invalid IPs per host and service. If an ICMP packet corresponds to an invalid IP for a certain service, the packet is placed with the malicious traffic. Conversely, if the packet corresponds to a valid IP and service, the packet is deemed benign.

As mentioned in the **UDP port scans** section, UDP scans can generate ICMP packets of the "Destination unreachable" type. The ICMP traffic of this type that is present in the network traces is the result of UDP scanning. Therefore, these packets are part of the malicious traffic and, as such, are classified as malicious.

Ping traffic can be either benign or malicious, depending on the frequency of the ping request and the size of the packet. Both the Windows and Linux versions of the *ping* utility sends out one ICMP request per second by default. The network administrator has no need for ICMP requests being received by the network at a high frequency. Thus, we set the limit for benign ICMP requests to four per second for a single host. Any host that sends ICMP requests at this frequency or higher will have their corresponding ICMP deemed malicious. The rest is placed with the benign traffic.

An example of a purely malicious instance of ICMP traffic is the Ping of Death [9]. This is an IPv4 ICMP packet that is larger than the maximum packet length of an IPv4 packet, which is 65,535 bytes. To extract Ping of Death packets, we created a tcpdump filter rule that examines ICMP packets and their IP header. It the total length of the packet exceeds 65,535 bytes, the packet is a malicious one. All remaining ping traffic is benign.

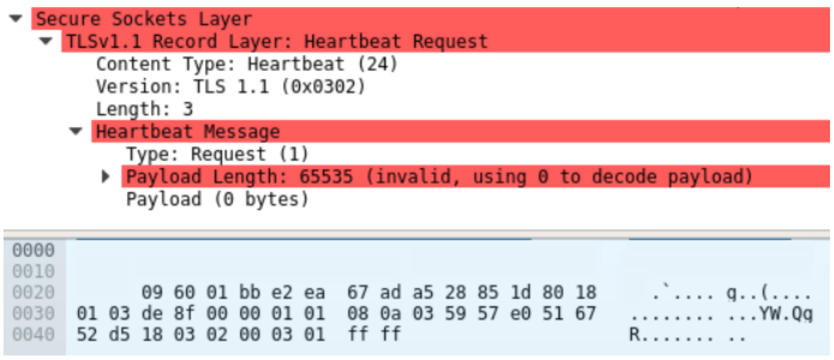
Furthermore, we also encountered ICMP packets of type *Time Exceeded* that originate from outside the network. These are ICMP packets that are sent back to a packet's source by a router when the time-to-live field reaches zero [58]. The packets announce that certain DNS packets originating from the public DNS server did not reach their destination before the expiration of its time-to-live value. There is nothing abnormal about either the DNS packet's destination IP or the contents of the payload. Therefore, a BGP routing loop is the most likely explanation as to the cause ICMP Time Exceeded packets. This routing loop is possibly caused by a link failure somewhere between our public DNS server and the destination of the packet. When the link failure is detected, the affected router *R* sends a withdrawal to routers with which it has a connection. It also sends the packet to one of the other routers so another route to the destination may be found. While new paths have not been propagated throughout the network, the other routers will only know of paths to the destination that use router *R*. Since router *R* has already sent a withdrawal to the other routers, the paths are invalid, and the routers will end up forwarding the packet to each other until the time-to-live value expires. See the paper by John et al. [23] for a more detailed explanation.

The network traces also contain the ICMPv6 packets used for the Neighbor Discovery Protocol. This protocol is used with IPv6 and performs actions similar to ARP in IPv4. Since the network uses IPv6, this traffic is necessary for the proper functioning of the network, and is therefore benign.

The rest of the ICMP and ICMPv6 packets that are present in the captured network traffic are part of normal network management traffic. They are placed with the rest of the benign traffic.

Exploitation attempts

During filtering, we found that outside hosts were attempting to exploit the Heartbleed bug [77] on a number of the local servers. See Figure 4.3 for one of the captured Heartbleed packets. The Ethernet and IP headers have been redacted from the figure.



```
▾ Secure Sockets Layer
  ▾ TLSv1.1 Record Layer: Heartbeat Request
      Content Type: Heartbeat (24)
      Version: TLS 1.1 (0x0302)
      Length: 3
    ▾ Heartbeat Message
        Type: Request (1)
      ▸ Payload Length: 65535 (invalid, using 0 to decode payload)
        Payload (0 bytes)

0000
0010
0020       09 60 01 bb e2 ea  67 ad a5 28 85 1d 80 18     .`.... g..(....
0030 01 03 de 8f 00 00 01 01  08 0a 03 59 57 e0 51 67   ........ ...YW.Qg
0040 52 d5 18 03 02 00 03 01  ff ff                     R....... ..
```

Figure 4.3: Wireshark capture of a Heartbleed packet

To detect Heartbleed packets, we must first make sure that the packet is a TLS Heartbeat Request packet. This is done by checking if the first byte of the TCP payload equals `0x18`. Then, we check if the packet's TLS version is one that is vulnerable to the exploit by checking if the second byte of the payload equals `0x03` and the third byte less than `0x04`. These bytes represent the major and minor version, respectively. Finally, we calculate the actual length of the payload and verify whether it matches the claimed length of the payload. Figure 4.3 shows that the attacker entered a payload length of 65,535, which certainly does not correspond to the packet's true length.

Legitimate traffic

The network traces also contain legitimate traffic that was overlooked by the network administrator when providing the network description. Specifically, this traffic is composed of network traffic between the Shells server and the IP address 178.63.40.67 on port 443. This IP address belongs to a chat client website[2]. Since this is legitimate user traffic, this is placed with the benign traffic.

Unknown traffic

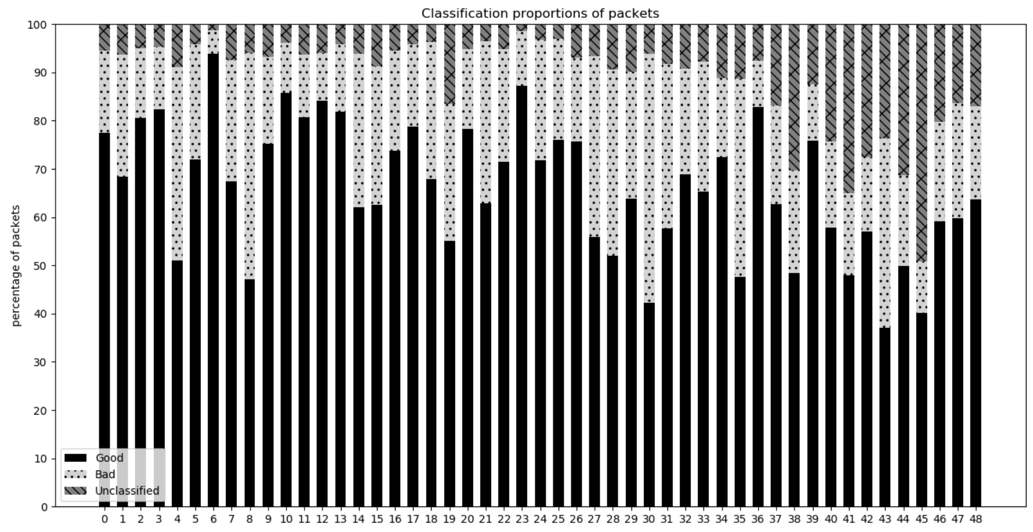
The last portion of unclassified traffic that we encountered was traffic to port 443 on the Shells server. This traffic is not supposed to be present, and only occurred on a small number of days. Interestingly, none of this traffic is HTTPS, even though it is port 443 that is receiving requests. We know this because the Shells server presents an OpenSSH banner when a connection with a host is established. After examination by the network administrator, no definite explanation could be found as to the cause of the OpenSSH service, nor for what purpose it was used. Due to the lack of information on this particular type of network traffic, we cannot establish the necessary ground-truth. All traffic of this type, namely traffic that corresponds to port 443 on the Shells server, was eliminated from the dataset.

4.5. Results of filtering process

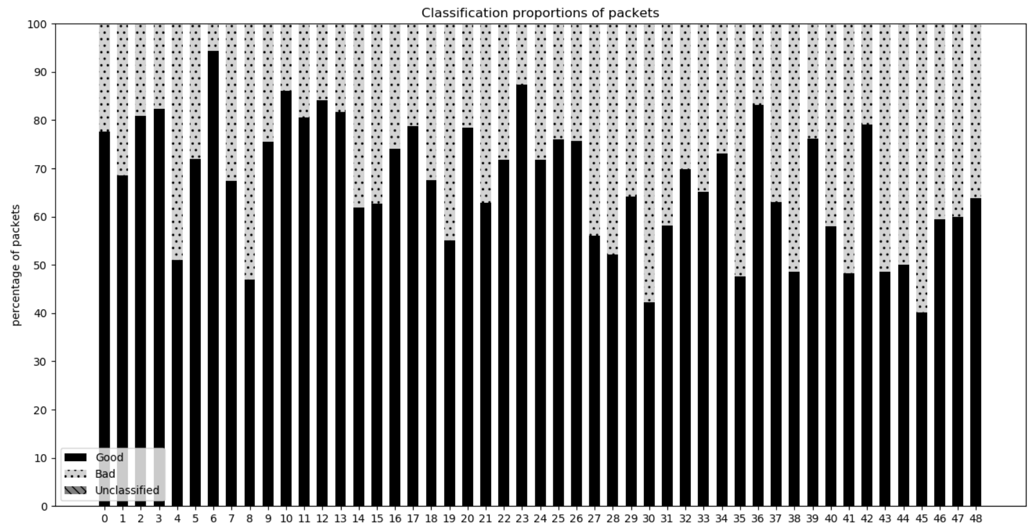
Figure 4.4 illustrates the proportion of good, bad, and unclassified traffic before and after the complete filtering process. Specifically, Figure 4.4a illustrates these proportions after filtering the traffic using the original description of the network that was provided by the network administrator. Using the filtering rules described in this chapter, we were able to entirely annotate the unclassified portion of network traffic as good and bad.

An interesting aspect about Figure 4.4a is the increase in unclassified network traffic as the days progress. While the original network description was somewhat effective at classifying most of the network traffic during the initial portion of the capture, it became increasingly difficult to do so later on. This demonstrates the unpredictability of traffic within a network, and the limited knowledge we have about our own networks.

[2]`https://weechat.org/`



(a) Before filtering



(b) After filtering

Figure 4.4: Proportions of good, bad, and unclassified traffic before and after the filtering process.

5 Attack traffic construction

The previous chapter spoke about the network used for the experiment. Its structure, contents, and the different types of traffic that the network receives every day were discussed. In addition to testing the Kitsune classifier on the everyday traffic of a real-world network, it is interesting to see how well the classifier would fare against network traffic that is undeniably malicious.

In the research by Mirsky et al. [47], they evaluate the performance of their classifier by simulating a number of attacks on their artificial networks. Nine different attacks are simulated in the network, which can be categorized into four types:

- Reconnaissance: OS scan, fuzzing.
- Man-in-the-Middle: Video injection, ARP poisoning, active wiretap.
- Denial of Service: SSDP flood, SYN flood, SSL renegotiation.
- Botnet Malware: Mirai infection.

After allowing the Kitsune instance to train itself on a stream of benign traffic, one of these attacks was performed on the network. This process was then repeated for every attack in the list.

Mirsky et al. have released the datasets of the captured traffic from their network that were used in the testing of Kitsune [45] in the form of pcap files. Directly testing our Kitsune instance on this dataset, however, will not provide us with any meaningful results. This is because our network, and all the network traffic found within it, differs significantly from the network used by Mirsky et al. The difference is not only in type of network traffic (web, mail, SSH, etc. vs. mostly video), but more subtle differences, such as IP and MAC addresses, and network latency. Any of these differences could give a classifier enough reason to label certain traffic as malicious, whether or not it actually is.

Therefore, it is necessary to transform the provided datasets to resemble our own network traffic as closely as possible. Doing so will ensure that our Kitsune instance is evaluating the attack traffic on the aspects that actually make the traffic malicious, as opposed to, for instance, IP or MAC addresses it as never seen before. On the other hand, we must not alter the datasets too much. Since the idea is to replicate the evaluation performed by Mirsky et al., the altered attack traffic must still resemble the original attack traffic. Only by ensuring this, will the results obtained from our experiments be comparable to the work by Mirsky et al. Along with the network traffic datasets themselves, Mirsky et al. also provides files containing labels for every packet in the datasets. This enables the extraction of the malicious traffic from the pcap files, which in turn allows us to focus exclusively on transforming the important (malicious) traffic, and prevents us having to deal with the benign traffic ourselves.

Figure 5.1 illustrates the true positive rates that Mirsky et al. [47] obtained when performing their experiments given the false positive thresholds of 0 and 0.001. Looking at the results, and specifically to the true positive rates at a false positive threshold of 0.001, we see that Kitsune does not seem to perform better than alternative intrusion detection systems for every type of attack. The three attacks where Mirsky et al. report significantly improved detection are when detecting Mirai infections, fuzzing attacks, and SSDP flooding. And apart from SYN flooding and OS scanning, the other intrusion detection systems either have very similar performance to Kitsune or significantly outperform it.

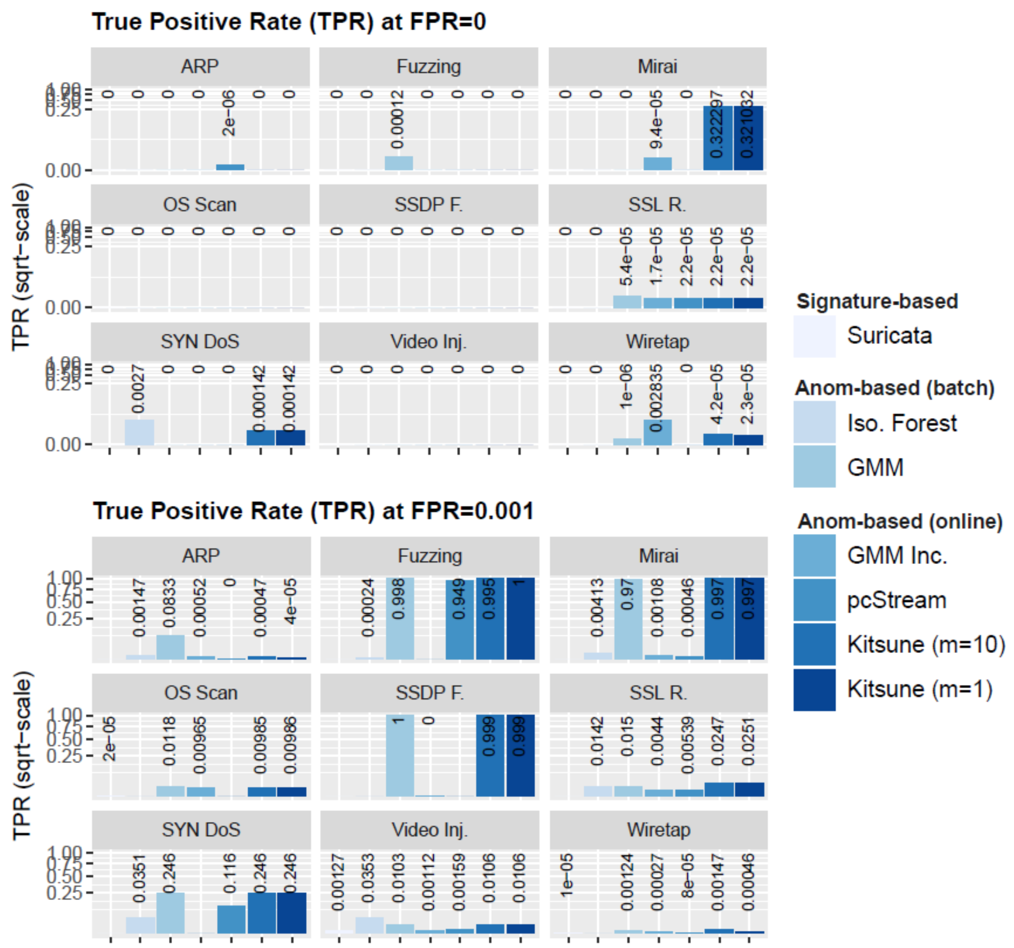


Figure 5.1: Performance statistics from Mirsky et al. [47]

So, out of the nine attacks, there are five which are interesting to reproduce in this current research. These are the Mirai infection, fuzzing, SSDP flooding, SYN flooding, and OS scanning.

Besides reproducing the same attacks as Mirsky et al., experimenting on Kitsune with novel attacks will provide added value to this research. One of the novel attacks that will be created for this purpose is a successful SSH brute-force with log in and

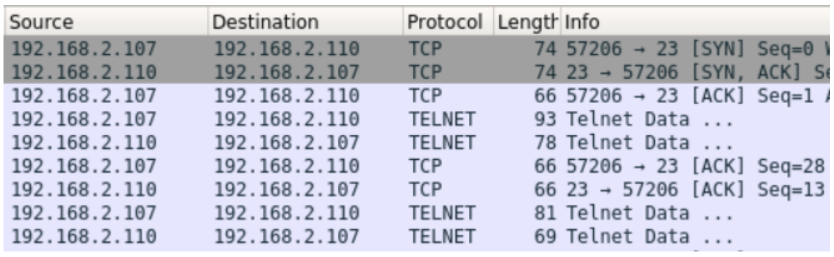

```
Source          Destination     Protocol Length Info
192.168.2.107   192.168.2.110   TCP         74 57206 → 23 [SYN] Seq=0 W
192.168.2.110   192.168.2.107   TCP         74 23 → 57206 [SYN, ACK] Se
192.168.2.107   192.168.2.110   TCP         66 57206 → 23 [ACK] Seq=1 A
192.168.2.107   192.168.2.110   TELNET      93 Telnet Data ...
192.168.2.110   192.168.2.107   TELNET      78 Telnet Data ...
192.168.2.107   192.168.2.110   TCP         66 57206 → 23 [ACK] Seq=28
192.168.2.110   192.168.2.107   TCP         66 23 → 57206 [ACK] Seq=13
192.168.2.107   192.168.2.110   TELNET      81 Telnet Data ...
192.168.2.110   192.168.2.107   TELNET      69 Telnet Data ...
```

Figure 5.2: The attacker initiating a Telnet connection with the victim.

command execution. SSDP flooding is an attack that was made possible to Mirsky et al. due to the numerous plug-and-play video surveillance cameras connected to the network [47]. The network that we use for our research has no such equipment that uses SSDP. Therefore, we must replace this attack with another that is more suitable for the type of network we have access to. Since SSDP flooding is an amplification attack, we choose the similar DNS amplification attack as a replacement. A DNS amplification attack is more suitable for our network, given the presence of a public DNS server on the network. This is the second novel attack that we will create for this research.

The sections below will elaborate on the process of transforming the attack traffic from the Mirsky et al. paper [47] into traffic closely resembling our own network's traffic. Furthermore, the creation of novel attack traffic will also be discussed.

5.1. Mirai infection

The Mirai attack was performed on the network composed of nine IoT devices and three PCs. This was the only attack that was performed on this particular network.

All devices, with the exception of a single PC, operated on a 192.168.2.0/24 network. The remaining PC sat in its own 192.168.4.0/24 network, but was able to communicate with the first network.

Firstly, the malicious traffic needed to be extracted from the dataset. The process of doing so, however, was not a straightforward one. This is because, for this particular case, the labels provided by Mirsky et al. did not indicate which specific packets were benign or malicious. Instead, the labels merely marked the point in the network trace where the attack was initiated. Every packet before this point is labeled benign, while every packet after is labeled malicious. Extracting all "malicious" traffic then leaves us the task of figuring out what is the actual attack traffic, and what is benign background traffic.

The Mirai botnet uses Telnet brute-forcing in order to spread itself to new devices [1]. Exploring the network traffic, we can indeed find Telnet traffic. In Figure 5.2, we can see a device initiating a Telnet connection with one of the IoT devices. Examining this Telnet conversation, we find that the device initiating the connection logs in successfully, and installs Mirai malware onto the victim device. We can therefore conclude that the malicious actor possesses the IP address 192.168.2.107, and the first victim 192.168.2.110.

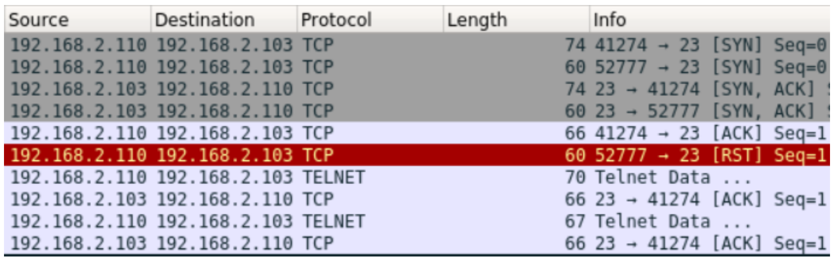
Interestingly, after infection, the IoT device starts to flood the network with ARP requests, probably to collect a list of all the devices on the network that can be probed for a vulnerable Telnet service.

Mirsky et al. [47] describe their Mirai attack as follows: "The attacker infects [an] IoT [device] with the Mirai malware by exploiting default credentials, and then scans for new vulnerable victims [on the] network." Based on that description, we need to verify whether the attacker attempts to infect any host other than the one at 192.168.2.110. A quick search informs us that no other attempts occur, and 192.168.2.110 is the only host that is personally infected by the attacker. Devices infected with Mirai will try to spread the infection to other vulnerable devices [1], so we must also check if the device at 192.168.2.110 tries to do this.

The infected IoT device indeed attempts this, as can be seen in Figure 5.3. This infected device initiates a Telnet connection with the device at 192.168.2.103, although it is not clear from the packet data whether the spread of the malware was successful.

Based on these findings, we must extract the following:

- all network traffic generated by 192.168.2.107;
- all network traffic between 192.168.2.107 and 192.168.2.110;
- all network traffic between 192.168.2.110 and 192.168.2.103;
- all ARP requests generated by 192.168.2.110; *and*
- all ARP replies addressed to 192.168.2.110.



```
Source         Destination    Protocol   Length  Info
192.168.2.110  192.168.2.103  TCP            74  41274 → 23 [SYN] Seq=0
192.168.2.110  192.168.2.103  TCP            60  52777 → 23 [SYN] Seq=0
192.168.2.103  192.168.2.110  TCP            74  23 → 41274 [SYN, ACK]
192.168.2.103  192.168.2.110  TCP            60  23 → 52777 [SYN, ACK]
192.168.2.110  192.168.2.103  TCP            66  41274 → 23 [ACK] Seq=1
192.168.2.110  192.168.2.103  TCP            60  52777 → 23 [RST] Seq=1
192.168.2.110  192.168.2.103  TELNET         70  Telnet Data ...
192.168.2.103  192.168.2.110  TCP            66  23 → 41274 [ACK] Seq=1
192.168.2.110  192.168.2.103  TELNET         67  Telnet Data ...
192.168.2.103  192.168.2.110  TCP            66  23 → 41274 [ACK] Seq=1
```

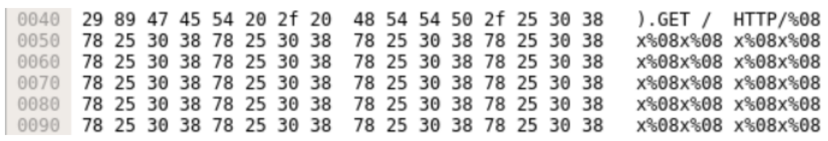
Figure 5.3: The infected IoT device initiating a Telnet connection with another IoT device.

After extraction, we obtain the true malicious network traffic from Mirsky et al.'s Mirai attack. In this remaining traffic, six different hosts: one wireless router, one attacker, and four IoT devices. To have this network traffic resemble our own network, every host had their MAC and IP address swapped with the MAC and IP of a server on our own network. See Table 5.1 for the specifics.

Table 5.1: Transformations applied to hosts on the Mirai network

DEVICE	IP	MAC	ASSIGNED VM
Wireless router	192.168.2.1	`4c:09:d4:c6:12:7b`	Gateway
Attacker device	192.168.2.107	`40:8d:5c:4b:99:14`	none (given new IP on network)
IoT 1 (initial victim)	192.168.2.110	`3c:33:00:98:ee:fd`	Shells
IoT 2	192.168.2.103	`00:01:6c:d5:63:5c`	Web
IoT 3	192.168.2.101	`40:8d:5c:4b:99:1d`	Mail
IoT 4	192.168.2.118	`b8:27:eb:e1:a9:f6`	Web with authentication

Finally, we need to shift all of the packet's timestamps to the same time period as our own collected traffic. This is done in order to prevent any adverse effect a timestamp mismatch might produce during processing by Kitsune. The original timestamps on this traffic are set to the 25th of October, 2018, and we initiated our packet capture the 21st of June, 2018. All timestamps must therefore be shifted 10,886,400 seconds into the past.



```
0040  29 89 47 45 54 20 2f 20  48 54 54 50 2f 25 30 38   ).GET /  HTTP/%08
0050  78 25 30 38 78 25 30 38  78 25 30 38 78 25 30 38   x%08x%08 x%08x%08
0060  78 25 30 38 78 25 30 38  78 25 30 38 78 25 30 38   x%08x%08 x%08x%08
0070  78 25 30 38 78 25 30 38  78 25 30 38 78 25 30 38   x%08x%08 x%08x%08
0080  78 25 30 38 78 25 30 38  78 25 30 38 78 25 30 38   x%08x%08 x%08x%08
0090  78 25 30 38 78 25 30 38  78 25 30 38 78 25 30 38   x%08x%08 x%08x%08
```

Figure 5.4: Snippet of an invalid GET request sent out by SFuzz

5.2. Fuzzing

The Mirai attack was the only attack that was performed on the IoT network. The remaining attacks, including the fuzzing attack, were carried out on the IP surveillance camera network [47].

Fuzzing is a technique in software testing in which incorrectly formed data is provided as input to some piece of software in order to discover bugs and vulnerabilities. Mirsky et al. [47] describe their fuzzing attack as "[searching] for vulnerabilities in the camera's web servers by sending random commands to their CGIs." A camera's CGI enables the users to control several aspects of the camera by sending commands to said CGI. CGIs a range of different commands, including the HTTP-like GET, POST, and HEAD methods [64]. The CGIs of the surveillance cameras used in the network, specifically, accept these commands when contained within a UDP packet.

Two obstacles stood in the way of directly porting this attack traffic to our own network. These two were the following:

1. The extracted malicious traffic contained UDP video traffic from the victim surveillance camera.
2. The fuzzing traffic itself consisted of HTTP requests wrapped in UDP packets.

Firstly, there is no UDP video traffic present anywhere in our network, so the Kitsune instance that will be trained on our network will more readily regard this traffic as abnormal. This traffic must therefore be eliminated from the dataset. Additionally, while HTTP requests certainly appear within the captured network traffic, they are all contained in TCP packets, not UDP packets. Since HTTP requests wrapped in UDP packets are not present in our dataset, we cannot directly use this traffic either. Fortunately, Mirsky et al. [47] share with the reader the tool that is used to create their fuzzing attack, namely SFuzz [10]. Instead of attempting to transform the UDP traffic into valid TCP connections, we create our own fuzzing traffic that is similar to the original by using the same tool and method described in their paper.

First, we set up a virtual machine and install SFuzz onto it. We then set up an Apache web server that will receive and reply to the requests sent by SFuzz. When the web server is up, we execute SFuzz on the port 80 web server. The fuzzing is made up of malformed and invalid GET, POST, and HEAD requests. See Figure 5.4 for an example of such an invalid request. The entire fuzzing process was captured using tcpdump.

The attack traffic contained approximately 20,000 packets. Since the attack was performed locally on a virtual machine, both the attacker and victim have 127.0.0.1 as IP address, and both have the same MAC address too. We can distinguish the direction of traffic by looking at whether port 80 is the source port or destination port. If port 80 is the destination port, it is a packet coming from the attacker and its destination is the victim machine. The source IP should then be replaced with a random IP address that is not present in the legitimate traffic. This would prevent any bias Kitsune might have against certain IP addresses to influence detection results. The destination IP address then has to be replaced with an IP address of one of the local servers. For consistency with our network, we replace this destination IP address with the IP address of the Web server. If port 80 is the source port, then the packet is a reply to the attacker coming from the web server. The source IP is then replaced with the IP address of our Web server, and the destination IP is replaced with the randomly chosen IP address. Furthermore, MAC addresses need to be fixed according to the working of our network: in case of incoming traffic (i.e. fuzzing traffic addressed to our Web server), the source MAC is replaced with a previously-unseen MAC, and the destination MAC is replaced with the MAC of our network's Gateway VM. Outgoing packets get the source MAC of our Web server, and the randomly chosen MAC address as destination MAC.

5.3. SYN flooding

A SYN flood is a denial-of-service attack in which an attacker sends out of SYN requests to a certain server. The objective is to consume all of the victim's server resources in order to prevent the server from providing its service to legitimate users.

The SYN flooding attack dataset is also one that is released by Mirsky et al. [45] The attack is performed using the hping3 tool [67]. The attacker sends a flood of SYN packets to the web server (HTTP port 80) of a single video surveillance camera, which causes its video stream to halt. The source IP address in the SYN packets are spoofed and replaced with random IP addresses. This dataset contains normal video traffic over UDP and over HTTPS. The malicious traffic itself that is present in the dataset consists of the SYN/ACK packets that the victim camera sends out in response to the incoming SYN requests, as well as retransmissions of these SYN/ACK packets when no ACK packet is received from the spoofed IP addresses. The original malicious SYN packets are not available in the dataset, because these packets did not reach the segment of the network where Mirsky et al. had set up their Kitsune instance [47].

This situation would not occur in our case, since all of the servers are connected to a central gateway. Therefore, this dataset is not suitable for our experiments, and we must create our own SYN flood dataset that is more appropriate for our type of network.

For our dataset, we decided to simulate a 100 Mbps SYN flood. A single SYN packet is 60 bytes, meaning that a 100 Mbps SYN flood would have to contain around $210,000$ SYN packets per second. Just like the dataset released by Mirsky et al., these SYN packets contain spoofed source IP addresses. In the original dataset, the SYN packets were addressed to a web server on port 80. For the newly created dataset, the SYN packets are addressed to port 80 of our own Web server. Finally, it is necessary to ensure that the SYN packet's timestamps correspond to the 21st of June, 2018. This dataset was constructed with the use of Scapy.

However, this dataset that was created has two limitations that were not possible to avoid due to them being caused by Kitsune itself. The Python version of Kitsune that is released by Mirsky et al. [44] can consume a great deal of memory when dealing with large surges of traffic. Much more than the unreleased C++ version of the IDS [47]. Because of these issues with memory consumption, we were forced to limit the number of SYN packets to a just million packets. In a 100 Mbps SYN flood, this is approximately five seconds worth of traffic. Also, every new IP address creates a new incremental statistic tuple, consuming more memory in the process. As a result, we also limited the randomization of the source IP addresses. All spoofed addresses are of the form $123.123.123.x$, where x is the only randomized portion of the IP address.

5.4. OS scanning

The OS scan dataset is also one that is released by Mirsky et al. [45] This attack is described as "[an] attacker [scanning] the network for hosts, and their operating systems, to reveal possible vulnerabilities." [47] In reality, the OS scan differs from what is described by Mirsky et al. in their paper. What actually occurs in the scan traffic is that the attacker targets a single host and scans this host by sending a single SYN request to every port (1-65535). No separate scan was conducted for UDP service discovery.

The IP address of the host performing the OS scan is 192.168.2.7. This address is used only by the attacker, and only for the purpose of scanning the target host. The IP address of the host being scanned is 192.168.2.1, meaning that the attacker and the victim are on the same subnet (assuming a */24* network). Seeing as this is the way Mirsky et al. performed their experiment, we must make sure that our version differs as little as possible. To this end, we replace the IP address of the attacker with a completely new IP address that would place the attacker inside our network (again, assuming a */24* network). We replace the victim's IP address with the IP address of the Web server. The MAC addresses are also altered: the attacker's MAC is changed to the MAC address of the Gateway VM, and the victim's MAC address is changed to that of the Web server.

5.5. Successful SSH brute-force

The successful SSH brute-force attack is another novel attack that we created for the purpose of our research. We wanted to test Kitsune's detection abilities in a scenario where an attacker uses readily available tools to successfully brute-force the password belonging to the admin user on an open SSH service, logs in using the found password, and executes some commands on the remote host. We set up a virtual machine we could perform the attack on, and all traffic was captured using tcpdump.

In order to perform a brute-force attack, an attacker sometimes uses a list containing a collection of commonly-used passwords. We constructed such a list, which contains 1141 passwords. Of the 1141, only the last password in the list is the correct one. An SSH server was set up on a virtual machine, and the *hydra* tool [82] was used to launch the brute-force attack. As the attacker, we launched hydra with 16 parallel tasks to speed up the brute-force attack. Since the correct password is the last element in the list of 1141, the brute-force attack had 1140 failed attempts before finally succeeding at attempt number 1141. After the successful brute-force and finding the correct password, the attacker then logs in to the remote SSH service using the acquired password and executes the commands "`cat /etc/passwd`" and "`cat /etc/shadow`" before exiting.

Because the attack was not performed on the actual network, but instead in a virtual machine, the pcap file must be altered before it can be used. First of all, the IP address of the attacker should be an IP address that is not present in the legitimate traffic. Again, this is to prevent any bias Kitsune might have against IP addresses it has already seen. Secondly, the victim's IP address should be replaced with the IP address of one of the local servers. It is important to keep the attack traffic consistent with the functionality offered by the servers on the network. Although all servers offer an SSH service, the Shells server is the only one that has this service as its primary function. The Shells server would send and receive much more SSH traffic than the rest of the servers. It is possible that Kitsune would see a spike in SSH traffic to the other servers as malicious—regardless of whether it actually is—simply because it is a spike that it is not used to seeing. To prevent this other bias from influencing the results, we chose to assign the IP address of the Shells server to the victim of the brute-force. Lastly, the

MAC address of the victim should also be replaced with the MAC address of the Shells server.

5.6. DNS abuse and DNS amplification

This section describes three different attacks that we created. All three are related to each other. The first two contains traffic that abuses the functionality of the public DNS server. The last attack is an incoming DNS amplification attack that resulted from the abuse of external open DNS servers.

DNS amplification attack. In a DNS amplification attack, an attacker sends DNS queries with a spoofed source IP to open DNS servers, which causes the DNS servers to flood the victim with all the DNS responses [8]. Since the size of DNS responses is larger than the query sent by the attacker, the amount of traffic that reaches the victim is amplified by the DNS servers. DNS amplification attacks are a very common type of attack, making up 18.6% of all DDoS attacks on the Internet [22]. The network used for this research contains such an open DNS server that could potentially be used to carry out DNS amplification attacks. Therefore, it is interesting to see whether Kitsune is able to detect such a scenario, and also whether it is able to detect such an incoming attack itself.

When abusing open DNS servers for DNS amplification attacks, attackers will oftentimes use ANY queries for DNS amplification attacks, as this results in the largest possible DNS response [8]. The public DNS server on our network, however, does not accept DNS queries of type ANY over UDP. It only accepts ANY queries over TCP. An alternative that also amplifies traffic significantly is the request of an MX record. For example, one incoming 88 byte MX query to the public DNS server produced a 492 byte response. The public DNS server has no restrictions on querying MX records over UDP.

We extracted the aforementioned MX query and response from the DNS traffic. This two packets form be the basis of the DNS amplification attack. Using *Scapy*, we replaced the source IP of the query and the destination IP of the response with a randomized IP address that is not present in the legitimate traffic. This IP address will represent the victim's IP address. In order to avoid the same memory issues as in the **SYN flood** case, we reduced the total number of packets to 200,000. Half of these packets are DNS queries, while the other half are DNS responses.

To examine the effect of the frequency with which the DNS queries are sent, we created to variations of this attack. The first version has the queries sent with a frequency of 100 DNS queries per millisecond, while the second version does it at a lower frequency of one DNS query per millisecond.

The DNS amplification attack itself was also constructed using the extracted DNS transaction. The difference with the previous two scenarios is that only the DNS response was used to create the attack. Since a DNS amplification attack is a flood of DNS responses, this is the only packet that is necessary. We simulated an attack where 10 external servers would flood the Web server with DNS response traffic. To this end, the source IP of the packet was replaced with one of 10 randomly chosen and previously-unseen IP addresses. The source MAC address was replaced with the MAC address of the Gateway VM. Because the Web server is the host that is being attacked, the destination IP and MAC addresses were replaced with those corresponding to the Web server. The DNS responses are received at a packet rate of 100 packets per millisecond.

6
IDS evaluation

Previous chapters described the preparation of the datasets that we used for the experiments. This chapter will begin by elaborating on the methodology we followed to perform the evaluation of the Kitsune IDS. Afterwards, we present the results obtained from the evaluation process. These results include Kitsune's performance on everyday data, as well as on actual attack traffic.

6.1. Performance evaluation

Once we have downloaded the Kitsune IDS [44] and installed it onto our machines, we must train it in the manner Mirsky et al. [47] did. Proper performance evaluation is only possible when sufficient data is captured, and all of this data is properly filtered and separated into legitimate and illegitimate traffic.

Mirsky et al. make the objective of their research to create an IDS that uses unsupervised learning [47]. They state this as an objective due to the difficulty encountered by supervised learning IDSs when trying to detect previously-unseen malicious traffic. Afterwards, they also state that Kitsune must be trained on "normal data" [47]. Therefore, we must test Kitsune in two different manners.

In the first, we will train Kitsune on *all* of the ground-truth legitimate data obtained from the first day of capturing. Because we are training on all of the legitimate data instead of just a portion, this will function as a best-case scenario for Kitsune's detection performance. After training on the legitimate data, we will merge the legitimate and illegitimate traffic into a single dataset and attempt to detect the illegitimate traffic with Kitsune. An additional experiment will be performed. We will inject attack traffic into our network traces. Then, the trained Kitsune instance will be tasked with processing the altered network trace and detect this malicious attack traffic. Specifically, we will use the benign portion of the first day of captured traffic for this experiment. We choose this portion of the network trace, since it is the same data with which we will train our Kitsune instance. To maintain consistency with their work, we will use attack traffic that is as similar as possible to the attack traffic used by by Mirsky et al. [47]. Mirsky et al. have released their attack traffic to the public [45]. We will use these datasets for own experiments. This research will also contain attacks that have not been performed by Mirsky et al. [47] A detailed explanation as to the generation of this attack traffic can be found in Chapter 5. In Chapter 3, we describe Kitsune's *m* parameter. Mirsky et al. perform their experiments in two different scenarios: (i) using a maximum autoencoder size $m = 10$, and (ii) using a maximum autoencoder size

$m = 1$ [47]. To maintain consistency between this research and that by Mirsky et al., we will also perform these experiments using both parameters.

The reason for Kitsune making use of unsupervised learning is so that it can be able to detect previously-unseen attacks as malicious traffic [47]. We must therefore perform a second type of test on Kitsune's detection performance. Instead of training Kitsune on only the legitimate traffic, we now train Kitsune on the *full* first day of captured network traffic. After training, we have Kitsune process the network traces from the rest of the captured days. The goal of this second test is to determine whether Kitsune is able to detect previously-unseen traffic if trained on realistic, noisy data.

Section 2.3 discussed many types of techniques and measures that are used for the performance evaluation of anomaly detection systems. Of all the performance metrics, Mirsky et al. focus on the following four: false positive rate, true positive rate, AUC, and equal error rate. Again, for the sake of consistency and to provide a true comparison of performance, we will also focus on these metrics during evaluation in this project.

6.2. Classifying everyday network traffic

We collected seven weeks of network traffic. Two experiments were performed using just this captured data to test both Kitsune's detection performance and claims made by Mirsky et al. [47] about their IDS. In the first experiment, we trained Kitsune on the ground-truth benign data filtered from the first day of capturing, and then processed the benign and malicious traffic from the remaining days. The second experiment had Kitsune trained on both the benign and malicious parts of the first day of captured traffic. Kitsune then processed the remaining traffic in an attempt to detect previously-unseen attack traffic. This section will discuss the results obtained from these two experiments.

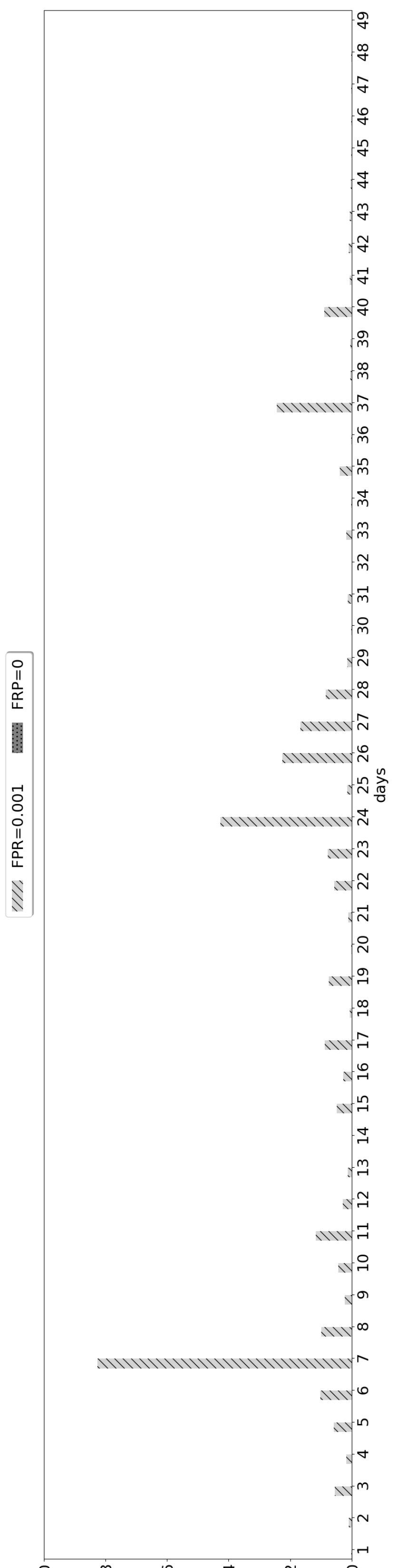
6.2.1. Detecting anomalies

After obtaining ground-truth benign and malicious traffic for all of the seven weeks of traffic, we trained Kitsune on the benign data of the first day of capturing. We first trained Kitsune using the parameter $m = 10$. The detection thresholds were determined by having our trained Kitsune instance process the same benign data with which it was trained. For $FPR = 0.001$ and $FPR = 0$, this yielded as thresholds approximately 0.1857 and 7.460, respectively. We then used the trained Kitsune instance to process the remaining 48 days of traffic and we recorded the anomaly scored produced by Kitsune. This process was repeated using the parameter $m = 1$. In this case, the detection thresholds for $FPR = 0.001$ and $FPR = 0$ were approximately 0.2021 and 6.2696, respectively. The false positive rates and false negative rates were computed using the detection thresholds. See Figures 6.1 and 6.2 for the false positive rates and false negative rates for all seven weeks of captured traffic for $m = 10$ and $m = 1$, respectively.

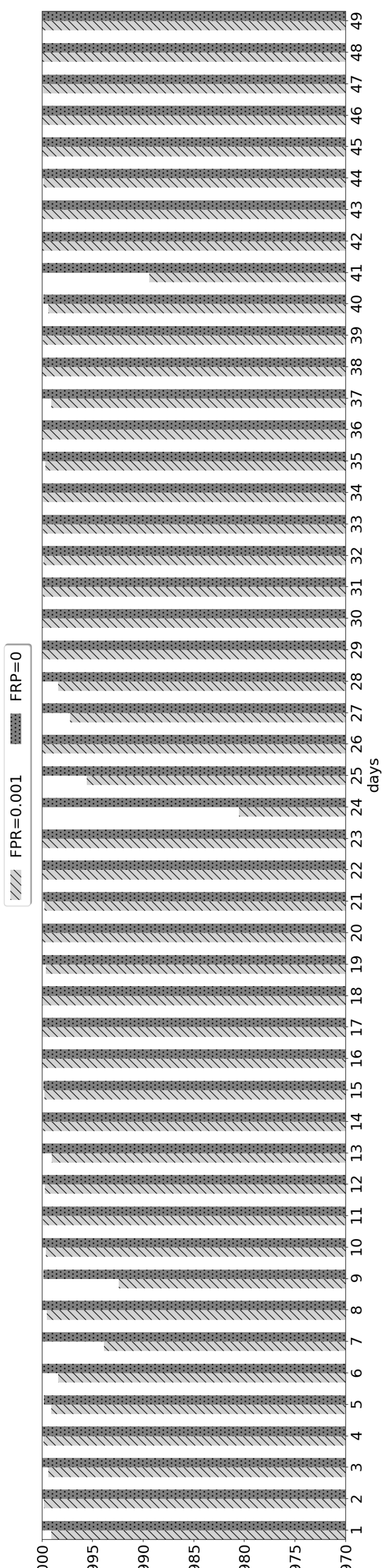
Table 6.1: Average FPRs and FNRs, using the thresholds obtained from the first day of network traffic

	$m = 10$		$m = 1$	
	$FPR = 0.001$	$FPR = 0$	$FPR = 0.001$	$FPR = 0$
actual FPR	0.064313	0.000002	0.057190	0.000002
actual FNR	0.999870	0.999998	0.999039	0.999998

As can be seen from both Figures 6.1a and 6.2a, Kitsune's FPRs are minuscule when using the $FPR = 0$ threshold. When looking at Figures 6.1b and 6.2b, however, we see that this low FPR comes at a price, namely an extremely high FNR. For every day, Kitsune does not produce an FNR lower than 0.97. This is the case for both thresholds, and for both $m = 10$ and $m = 1$. Therefore, while Kitsune rarely classifies normal traffic as malicious, detection of actual malicious traffic occurs much less frequently. Table 6.1 contains the average FPRs and FNRs from Kitsune's anomaly detection.

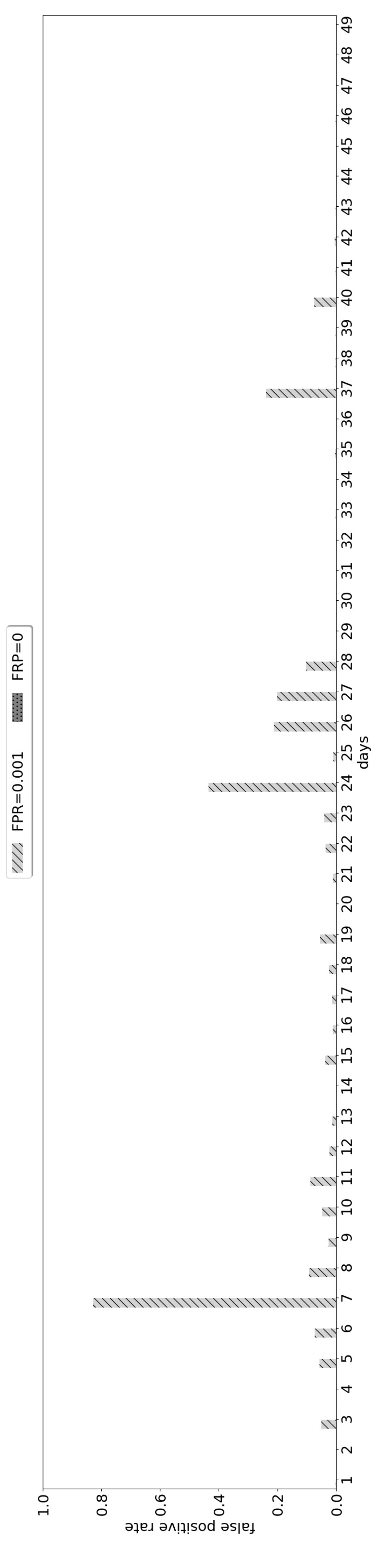


(a) False positive rates

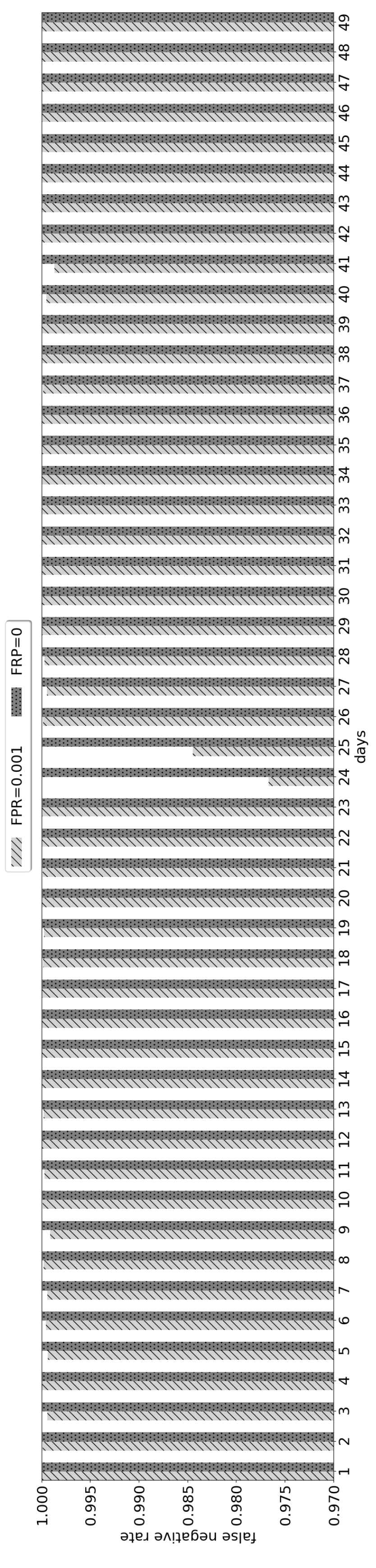


(b) False negative rates

Figure 6.1: False positive rates and false negative rates using Kitsune instance with maximum autoencoder size $m = 10$.

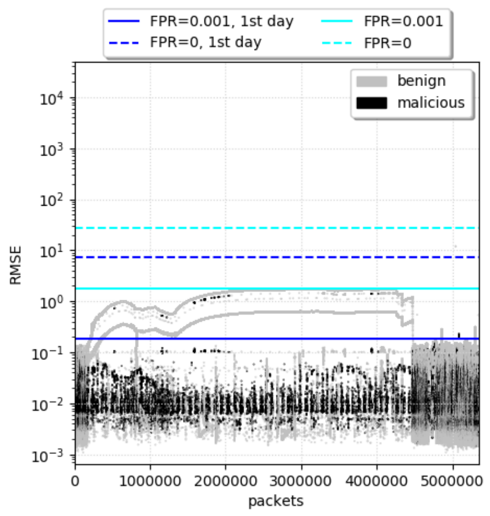


(a) False positive rates

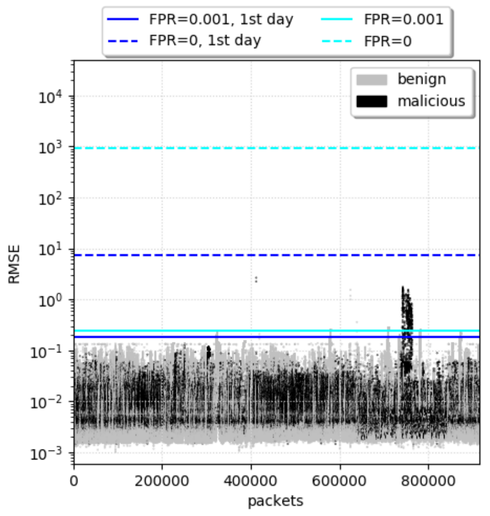


(b) False negative rates

Figure 6.2: False positive rates and false negative rates using Kitsune instance with maximum autoencoder size $m = 1$.



(a) The anomaly scores of the day 7 network trace show a significant amount of false positives.



(b) The anomaly scores of the day 41 network trace show a large burst of false positive instances.

Figure 6.3: Two scatter plots of Kitsune anomaly scores that clearly contain anomalous traffic

We can make a scatter plot of the anomaly score per packet from the processed traffic. Such a scatter plot provides us with a general view of the classifier's effectiveness and allows us to quickly evaluate its performance. Two of these scatter plots are illustrated in Figure 6.3. These two specific graphs are especially interesting. Figure 6.3a, for instance, illustrates a case where a large amount of unrecognized benign traffic could possibly be classified as malicious, leading to an extremely high FPR. Figure 6.3b is interesting, because it presents a relatively large burst of malicious malicious that Kitsune is actually able to detect.

We extracted the detected anomalies from the malicious traffic (i.e. the true positives). In order to extract the largest number of anomalies while still maintaining a consistent threshold, we chose the $FPR = 0.001$ threshold.

Correctly detected as anomalies (i.e. true positives) were mainly illegitimate connection attempts to port 443 on the web server with authentication. Also detected as anomalies were scans and illegitimate connection attempts to port 22 on several servers. As a matter of fact, we found that the burst of detected anomalies during day 41, illustrated in Figure 6.3b, was a single host sending out a large number of SYN packets to port 22 on our servers.

The false positive instances were identified as being primarily file downloads from port 443 on the web server with authentication, and large email traffic. In Figure 6.3a, there are two curves of benign traffic that extend away from the rest of the traffic. These two curves turned out to be just this: file download traffic between the aforementioned server and an outside host.

Kitsune's design is based on autoencoders. The black-box nature of ANNs such as autoencoders makes it a difficult process to identify the reasons for a particular classification result. Therefore, instead of investigating the calculations performed by the autoencoders, we chose to look at the features responsible for certain results. Kitsune attempts to reconstruct the clusters of features that are extracted from network traffic. A packet's anomaly score is determined by the accuracy of this reconstruction process (specifically, the RMSE between the original and the reconstruction). Thus, if we iden-

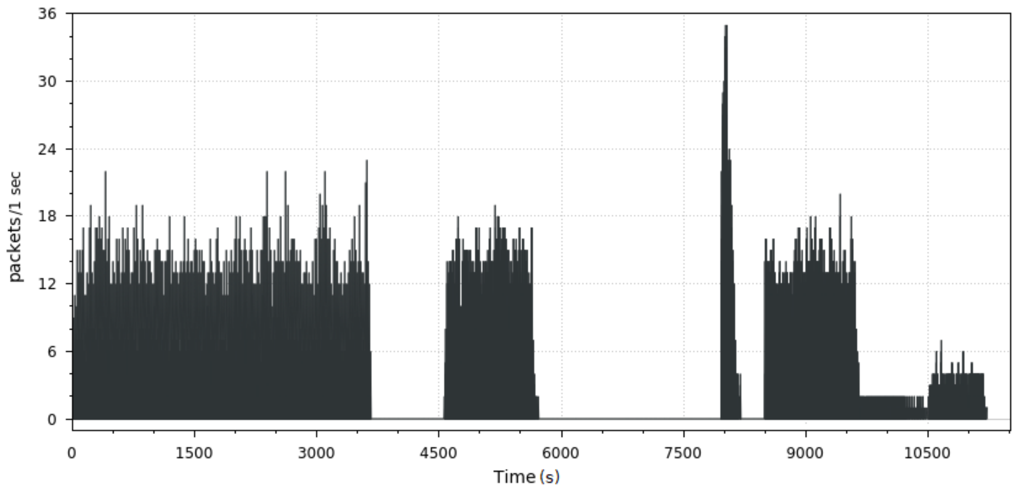


Figure 6.4: Packets sent by a malicious host in order to scan the network in the day 41 network trace

tify the feature cluster that contributes the most toward the final anomaly score, we can infer what aspects of network traffic Kitsune deems more significant when trying to detect anomalies.

The cluster that contributes the most towards the high anomaly scores illustrated in Figure 6.3a consists of eight features. These features all describe the weight (see Figure 3.2) of the incoming packets for all type of incremental statistic tuples (SrcMAC-IP, SrcIP, Channel, and Socket) for λ values 0.01 and 0.1. In other words, a sudden surge in the number of packets exchanged between two hosts, either on arbitrary ports or on a single, specific port. This makes sense, since the anomalous packets originate from the downloading of a large file.

As for day 41 (Figure 6.3b), the feature cluster with the highest contribution is 10 features large. The features in this cluster all correspond to the Socket between an outside host and each one of our servers. Five of the features describe the correlation between the sizes of the packets that are sent between the two hosts on a specific port for all time windows λ. Almost the same applies to the other five features, but the statistic is the approximate covariance, instead of correlation. Interestingly, upon further investigation of this malicious host, we found that the host had been scanning our network several times throughout the day. The I/O graph in Figure 6.4 illustrates these scans. Kitsune was only able to detect one of these instances, namely the second burst from the left, roughly between the values 4500 and 6000 on the x axis.

During the last day of capturing, day 49, Kitsune found another interesting burst of anomalous packets. As opposed to the previously discussed burst, this one was a burst of false positives. This burst is shown in Figure 6.5. Here, the false positives have a different appearance than those from the download traffic we found during day 7 (Figure 6.3a), so its cause is probably different. It turns out these instances are outgoing SSH packets over IPv6, from the shells server to an outside host. The feature cluster that contributes the most is the same as for the anomalous packets in day 41: the correlation and covariance of the sizes of the packets sent between two Sockets. This outside host had been interacting with our network throughout the day, but only a small amount of packets were seen as anomalous. Given the responsible feature cluster, we investigated the sizes of the packets exchanged between the hosts. Figure 6.6 compares the packet sizes of incoming and outgoing SSH packets. The graph shows

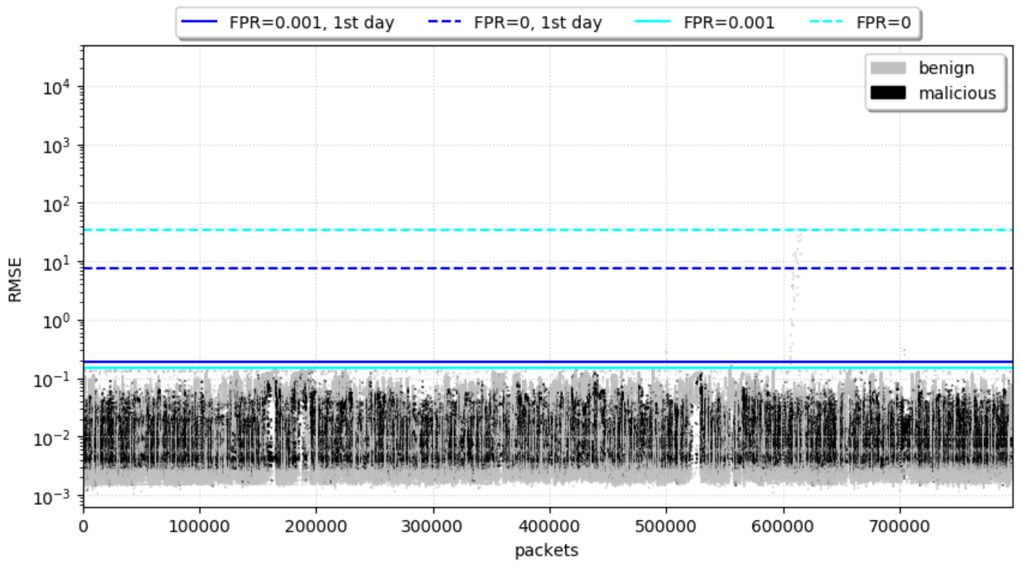


Figure 6.5: Scatter plot of Kitsune anomaly scores containing a burst of false positives around the 600,000 mark on the x axis, day 49.

us that for most of the time, there is a strong correlation between incoming and outgoing SSH packets. This correlation is broken at around the 20,000 mark on the x axis, which is also the location of the anomalous packets. At this point, we see total absence of incoming SSH packets. This is due to the scanning host not replying to the scanned server, causing packet retransmissions. Based on this graph we can infer that this sudden lack of incoming SSH packets, while SSH packets are still being sent out, is the reason for the traffic being classified as anomalous. It is possible that Kitsune expects that, for this type of traffic, the amount of packets sent between the two hosts be balanced and consistent. The sudden halt in bidirectional communication between the two hosts creates an imbalance on one side. This then causes the calculated correlation and covariance statistics to deviate from "normal" behavior, which, in turn, causes Kitsune to finally classify the scan as anomalous.

Figure 6.7 illustrates the RMSEs of all the days of captured traffic in the form of a box plot. For all but two of the days, the second and third quartiles are all below the $FPR = 0.001$ threshold. The two exceptions are days 7 and 24. This indicates that a significant portion of the traffic during these two days produces anomaly scores above this threshold. Upon investigation, this is indeed the case (see Figure 6.3a for the anomaly scores of day 7). Note the large amount of outliers above the two thresholds. This fact, together with the false positive rates in Figure 6.1a, highlights the instability of Kitsune's detection performance.

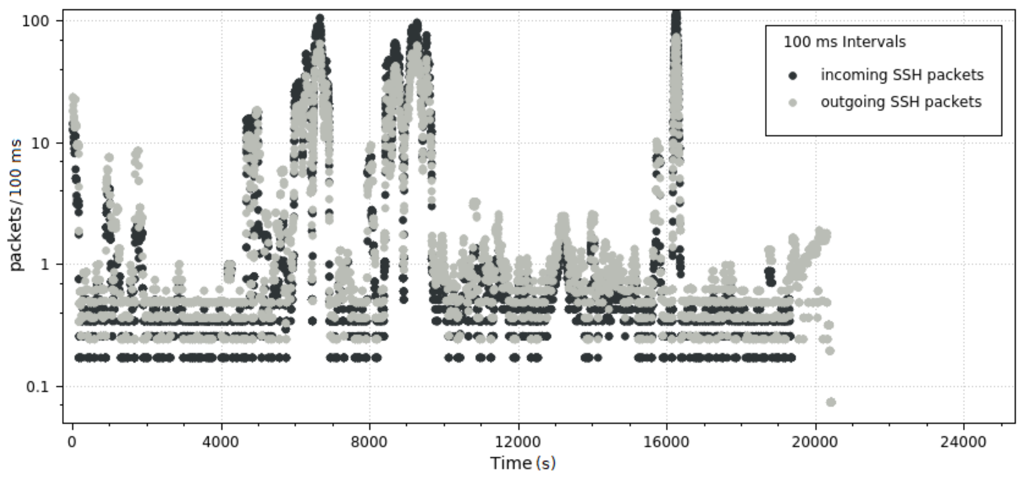


Figure 6.6: Sizes incoming SSH packets vs sizes outgoing SSH packets, day 49. The gray points that appear in the graph after the black points stop (at around the 20,000 mark on the x axis) correspond to the burst of false positives illustrated in Figure 6.5.

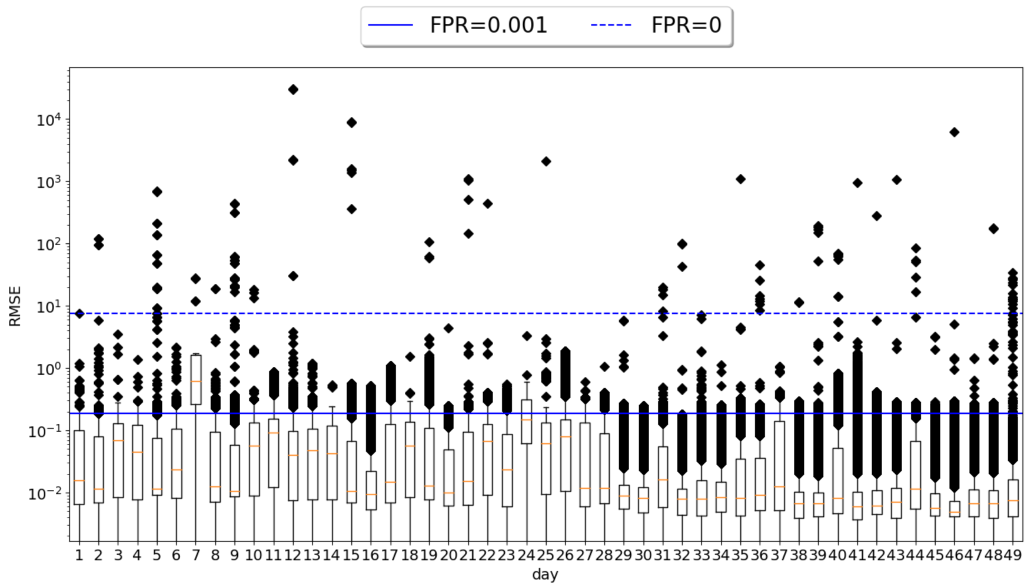


Figure 6.7: Box plot of the RMSEs for every day of captured traffic. The RMSEs are those produced by the Kitsune instance using maximum autoencoder size $m = 10$.

6.2.2. Unsupervised learning

The second aspect we wanted to evaluate was the claim by Mirsky et al. about Kitsune's practicality as an unsupervised learning system [47]. Mirsky et al. designed their system using unsupervised learning techniques in order to 1) be able to detect previously-unseen attacks, and 2) avoid the costly process of explicitly labeling network traffic as benign or malicious. The second item is true from a purely theoretical standpoint. The rationale behind using this method instead of others, however, is a flawed one. Although it is true that explicitly labeling network traffic is not necessary, this time-consuming task has merely been replaced by another equally time-consuming task. We have demonstrated this in Chapter 4, by describing the elaborate procedure of filtering all network traffic to obtain ground-truth data.

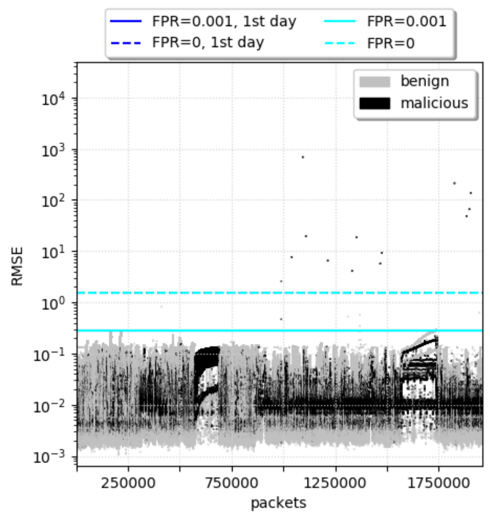
With statement 2) debunked, we focus our attention to the first statement. Due to Kitsune's use of unsupervised learning, it should be able to detect novel attacks.

We trained Kitsune on both the benign and malicious traffic of the first day. After training, we fed the rest of the traffic from the remaining days into the IDS. Because of time constraints, we performed this experiment only using $m = 10$ as parameter.

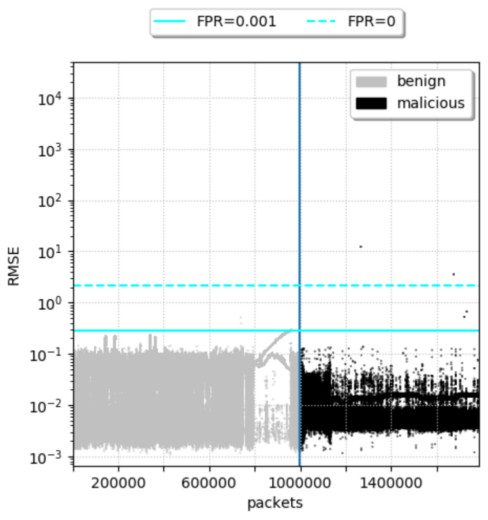
First of all, we identified the network traffic that is not present in the first network trace, but is present during other days. The following is a non-exhaustive list of some of the novel network traffic and protocols:

- BACnet [2]
- HART-IP [60]
- L2TP [84]
- UDP encapsulation of IPsec packets [21]
- X Display Manager Control Protocol [54]
- OMRON FINS Protocol [51]
- Quake III Arena Network Protocol
- DirectPlay 8 protocol [42]
- kNet Protocol [25]
- TACACS [16]

Ideally, these types of traffic would be classified as anomalous by Kitsune, as the data with which it was trained did not contain any of the aforementioned protocols. In actuality, however, none of this traffic was classified as anomalous by Kitsune. The anomalies that Kitsune could detect were generally of the same type, if not exactly the same instances. Comparisons between anomaly scores are illustrated in Figures 6.8 and 6.9. In the figures we see a clear decrease in the number of detected anomalies. Do note the vertical line separating the traffic in Figures 6.8b and 6.9b. The network traffic on the left side of the vertical line is benign traffic, while the malicious traffic is on the right. Novel traffic could be either benign or malicious, and a malicious host can first produce benign traffic before initiating an attack. Since Kitsune's detection of malicious packets is partly dependent on the traffic that surrounds it, we wanted to remove the possibility of ambiguous hosts altering detection performance. Therefore, in the case of training on both benign and malicious data, we decided to split up the data and have Kitsune process the benign and malicious portions separately. This ensures that



(a) Anomaly scores when Kitsune is trained on only benign traffic.



(b) Anomaly scores when Kitsune is trained on both benign and malicious traffic. Note that the benign and malicious traffic is separated from one another during processing.

Figure 6.8: Scatter plots of Kitsune anomaly scores when processing the day 5 network trace, both using maximum autoencoder size $m = 10$.

the classification of the benign novel traffic does not influence the classification of the malicious novel traffic, and vice versa.

6.3. Attack simulations

In addition to testing Kitsune's performance on everyday traffic, we also examined its effectiveness detecting specific attacks on a host or the network. Our process of obtaining or generating this attack traffic is described in Chapter 5. This section will discuss the results of these particular experiments.

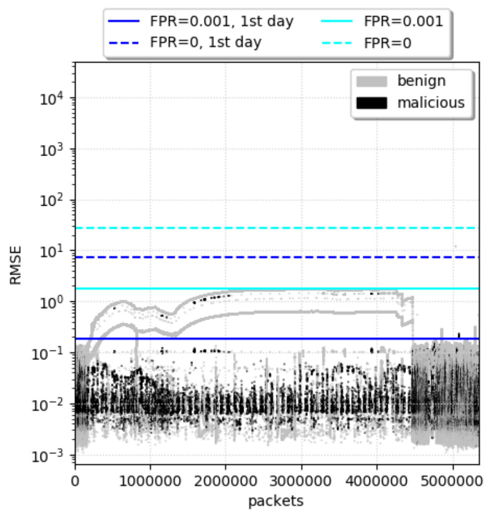
We tested Kitsune on the attack traffic in two different ways:

1. process only the attack traffic; *and*

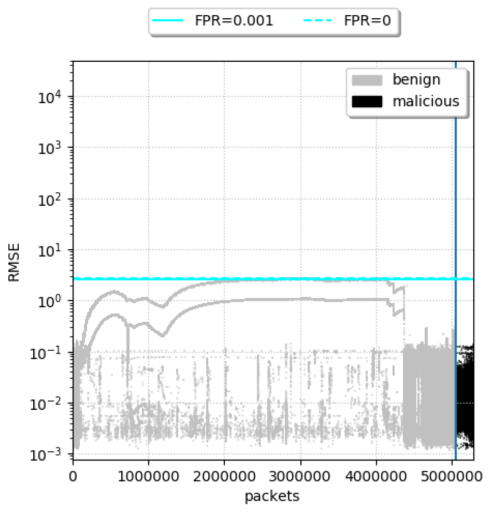
2. inject the attack traffic into our captured network traces and process the entire dataset.

Because we trained our Kitsune instance on the benign portion of the first day of captured traffic, we decided to inject the attack traffic into this benign traffic. Since all but the attack traffic was used for training, this would give us a best-case scenario of Kitsune's detection performance. The Mirai attack traffic is mostly the same as what was used by Mirsky et al. in their experiments [47].

Figure 6.10 illustrates the anomaly scores for the Mirai attack produced by Kitsune using the parameter $m = 10$. The anomaly scores from *only* the attack traffic are shown in Figure 6.10a. Figure 6.10b shows the anomaly scores when injected into network traffic. As can be seen from the figures not a single packet from the attack itself was detected by Kitsune. Instead, we can see a number of false positive instances in Figure 6.10b. These false positives were also present when the original network trace was processed by Kitsune. It is particularly strange that no single detection was made. Not only does the attack contain ARP flooding, but Telnet traffic too. Telnet is a protocol



(a) Anomaly scores when Kitsune is trained on only benign traffic.

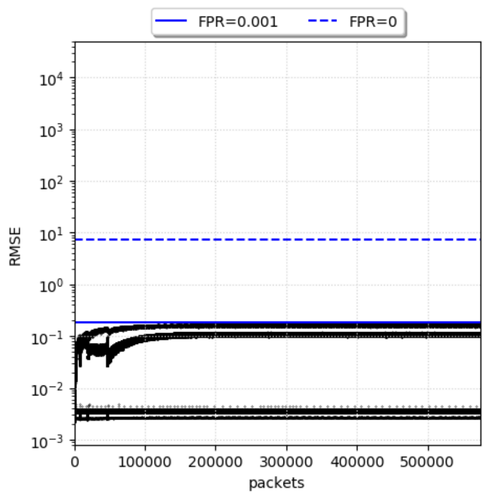


(b) Anomaly scores when Kitsune is trained on both benign and malicious traffic. Note that the benign and malicious traffic is separated from one another during processing.

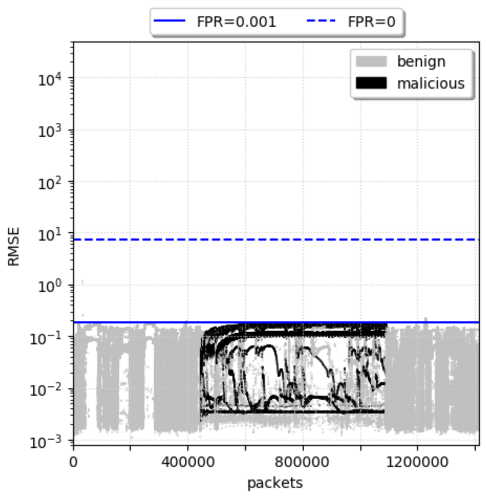
Figure 6.9: Scatter plots of Kitsune anomaly scores when processing the day 7 network trace, both using maximum autoencoder size $m = 10$.

that is never used in the training data, and neither is its corresponding port 23 used for any actual communication. Still, Kitsune does not consider the usage of this protocol and port anomalous. Interestingly, using $m = 1$ as parameter, Kitsune was able to detect a large portion of the Mirai attack traffic. This is shown in Figure 6.11. Out of the 575,000 malicious packets, Kitsune was able to detect approximately 520,000 of them, achieving a precision of 0.99, and a recall of 0.91. Kitsune detected none of the Telnet traffic, however. Instead, it detected the ARP flooding. We can see clearly that Kitsune found the ARP flooding more anomalous than the rest of the attack traffic when we look at the features responsible for the high anomaly scores. These are all weight statistics from a packet's SrcIP, SrcMAC-IP, and Channel for $\lambda = 0.1$. We note the absence of any of the Socket features, which are the ones that specifically capture behavior on specific ports. Since the weight statistic is the damped count of all the packets that have arrived [47], the flood of ARP packets will cause this statistic to spike in value.

In Figure 6.12 we can see the results of the fuzzing attack, using $m = 10$ as parameter. The results from this attack are much more positive than the Mirai results. Here, we see that the majority of the malicious traffic is detected using the $FPR = 0.001$ threshold. Even when using the stricter $FPR = 0$ threshold, Kitsune manages to detect a portion of the attack packets. The features that contribute the most toward the high anomaly scores are all based on the variation of packet sizes. First of all, the radius statistic for a packet's Channel and Socket (for all λ) weighs heavily. The radius statistic is calculated using the standard deviation of packet size [47], so we can expect the standard deviation statistic itself to also play a role in detection. As expected, the standard deviation statistics from a packet's SrcMAC-IP and Channel (for $\lambda = 0.01$) are the features that contribute the most. From this, we can see that Kitsune detected the fuzzing attack due to the inconsistent packet sizes originating from the malicious host. When processing the data using $m = 1$, detection performance improves signifi-

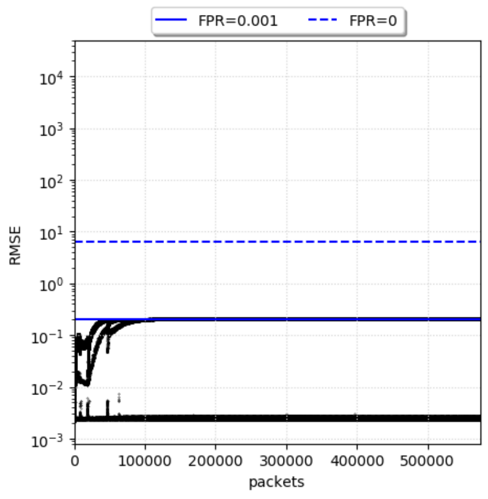


(a) Anomaly scores when the Mirai attack traffic only is processed by Kitsune. None of anomaly scores exceed the $FPR = 0.001$ threshold, meaning that not a single detection is made.

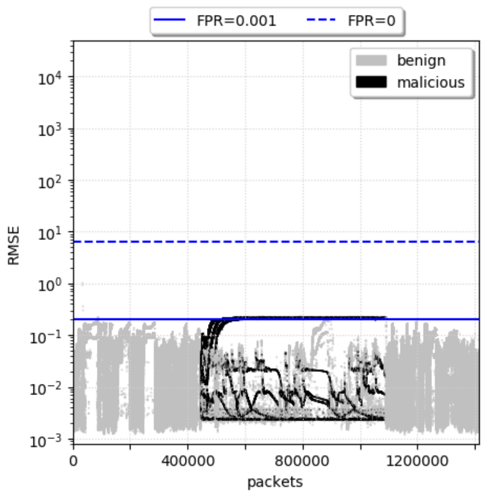


(b) Anomaly scores when the attack traffic is injected into the benign portion of the day 1 network trace. Executing the attack in the context of the rest of the network traffic has no significant effect on detection performance, since the anomaly scores here do not exceed the $FPR = 0.001$ threshold either.

Figure 6.10: Mirai attack anomaly scores produced by Kitsune using maximum autoencoder size $m = 10$

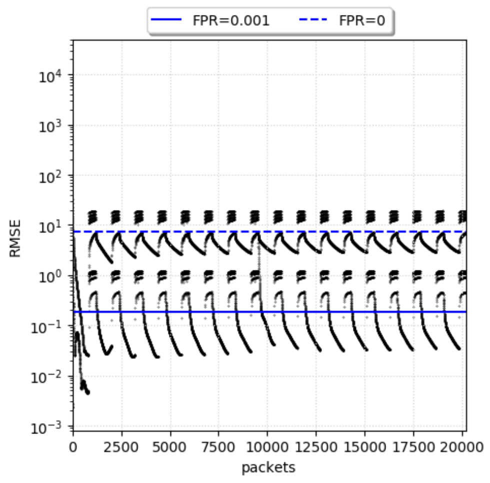


(a) Anomaly scores when the Mirai attack traffic only is processed by Kitsune. As opposed to the anomaly scores illustrated in Figure 6.10a, the anomaly scores in this case do exceed the $FPR = 0.001$ threshold, thereby allowing Kitsune to detect these instances as anomalous.

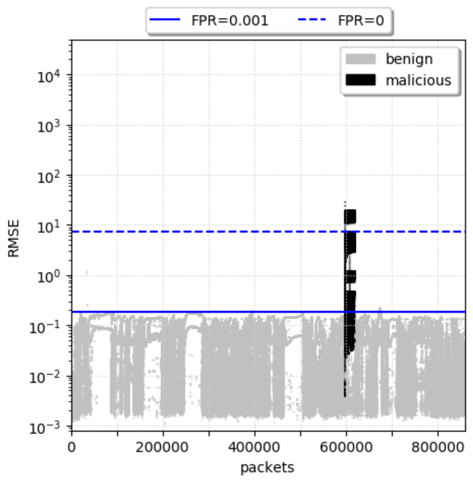


(b) Anomaly scores when the attack traffic is injected into the benign portion of the day 1 network trace. Just as in Figure 6.11a, the anomaly scores exceed the $FPR = 0.001$ threshold, meaning that Kitsune is able to detect portions of the Mirai attack when executed in the context of the rest of the network traffic.

Figure 6.11: Mirai attack anomaly scores produced by Kitsune using maximum autoencoder size $m = 1$



(a) Anomaly scores when the fuzzing attack traffic only is processed by Kitsune. Most of the malicious packets are detected by Kitsune.



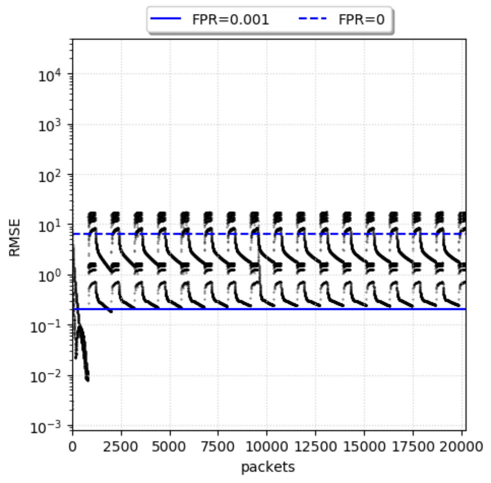
(b) Anomaly scores when the attack traffic is injected into the benign portion of the day 1 network trace. There is no significant increase in detection performance when executing the attack within the context of the rest of the network traffic.

Figure 6.12: Fuzzing attack anomaly scores produced by Kitsune using maximum autoencoder size $m = 10$

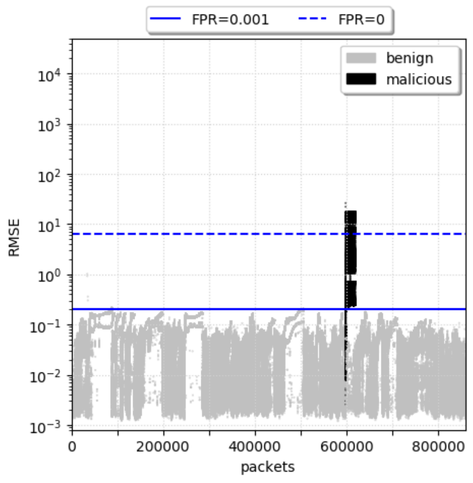
cantly. As can be seen in Figure 6.13b, almost all of the attack traffic is detected by the $FPR = 0.001$ threshold. In this case, there are many different features that contribute heavily to detection. Just as the $m = 10$ results, these are all standard deviation and radius statistics. And, again, the most significant feature is a standard deviation statistic. Specifically, the standard deviation statistic of a packet's Socket for $\lambda = 5$.

Figure 6.14 illustrates the $m = 10$ anomaly scores of our SYN flood simulation. We performed this experiment only in an out-of-context scenario, meaning that the traffic was not injected into captured traffic. This is due to Kitsune's memory consumption: the Kitsune process used up all available memory on the machine, which caused the process to stall. Despite this being an 100 Mbps SYN flood, no detection was made. It is possible that because of the large amount of legitimate SYN packets in the training dataset we extracted from a more realistic network, Kitsune is less sensitive to SYN packet anomalies. The $m = 1$ results did not differ significantly from the $m = 10$ results. Kitsune was not able to detect any anomaly. The resulting $m = 1$ anomaly score graph shown in Figure 6.14b is very similar to the $m = 10$ graph in Figure 6.14a, albeit less smooth. The features that contribute the most to the anomaly score of these packets are weight statistics, meaning that Kitsune indeed recognizes the traffic as a flood of packets.

Shown in Figure 6.15 are the anomaly score results from OS scan traffic for $m = 10$. As shown in the figure, much of this traffic is detected by Kitsune, but only when using the $FPR = 0.001$ threshold. The features most responsible for the high anomaly scores are all weight statistics for a packet's SrcMAC-IP, SrcIP, Channel, and Socket. Throughout the duration of the attack, the weight statistic remains the most anomalous. The only change is in the time window, which changes several times during the OS scan. The results when using the $m = 1$ parameter do not differ significantly from the $m = 10$ results. Most of the scanning traffic is detected. All of the features that contribute the

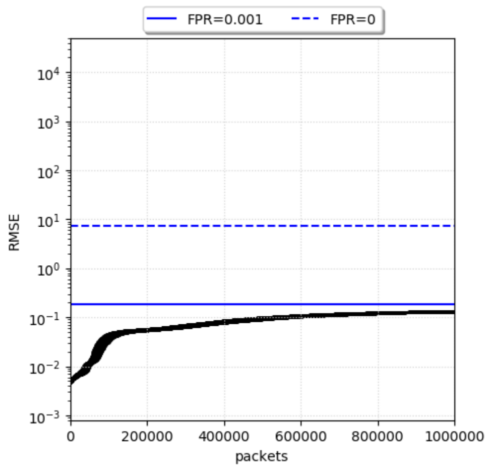


(a) Anomaly scores when the fuzzing attack traffic only is processed by Kitsune. All of the malicious packets, except for the first portion, are detected by Kitsune.

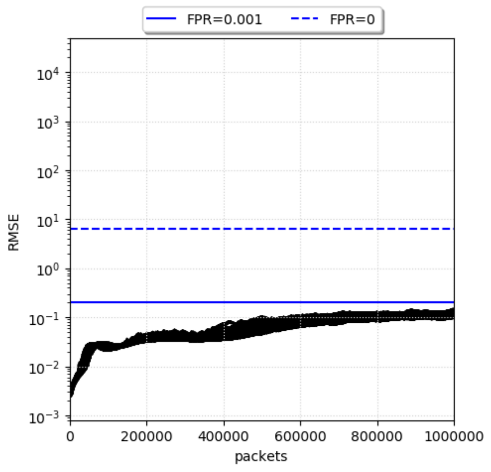


(b) Anomaly scores when the attack traffic is injected into the benign portion of the day 1 network trace. Like Figure 6.13a, only the first portion of malicious packets are not detected.

Figure 6.13: Fuzzing attack anomaly scores produced by Kitsune using maximum autoencoder size $m = 1$

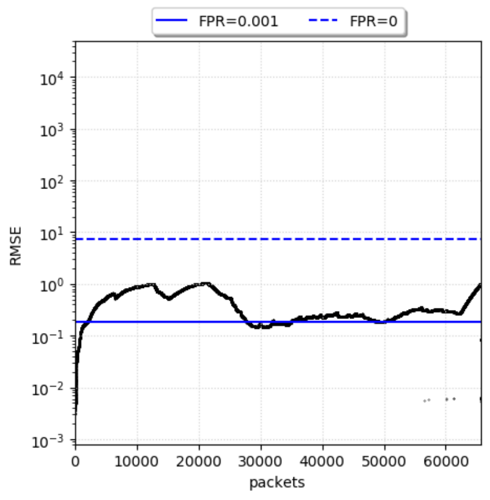


(a) Anomaly scores produced by Kitsune using maximum autoencoder size $m = 10$.

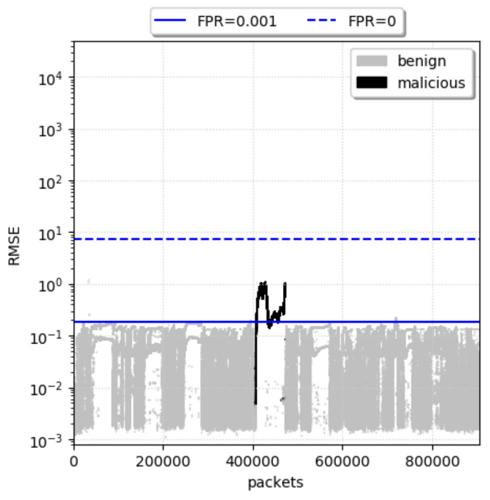


(b) Anomaly scores produced by Kitsune using maximum autoencoder size $m = 1$.

Figure 6.14: Anomaly scores produced by Kitsune when processing the SYN flood traffic. This particular attack is not injected into the day 1 network trace due to Kitsune's large memory consumption.



(a) Anomaly scores when the fuzzing attack traffic only is processed by Kitsune.



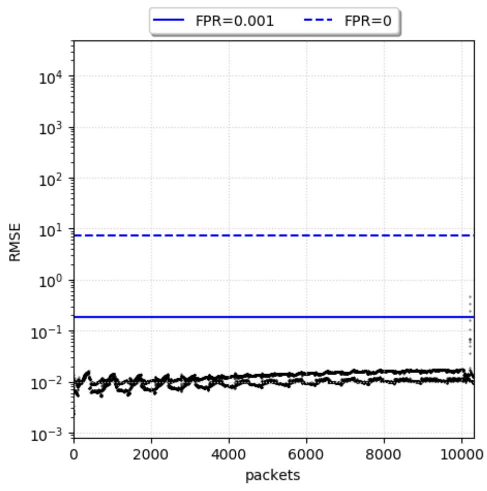
(b) Anomaly scores when the OS scan traffic is injected into the benign portion of the day 1 network trace. There is no improvement in detection performance.

Figure 6.15: OS scan anomaly scores produced by Kitsune using maximum autoencoder size $m = 10$

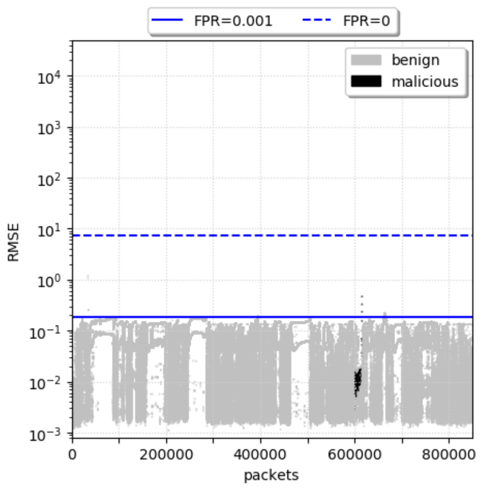
most toward the high anomaly scores are weight statistics, mostly for the Socket between the scanning host and the host being scanned.

The $m = 10$ results from the successful SSH brute-force attack are shown in Figure 6.16. As can be seen, the actual brute-force traffic that makes up the majority of the malicious traffic is not detected by Kitsune. Kitsune was only able to detect three packets at the very end of the attack. The detected packets contain a command sent to the victim's machine, and the victim's response *after* successfully guessing the password and logging in. These packets are larger than the ones sent during the brute-force attack, and this is precisely the reason why Kitsune was able to detect them. We can see this when looking at the responsible features, as they are all standard deviation statistics for a packet's SrcMAC-IP, SrcIP, Channel, and Socket, for several different time windows λ. For $m = 1$, the feature most responsible for detection is also a standard deviation statistic: the standard deviation in packet size for the Socket between the attacker and victim for $\lambda = 5$. This means that for both cases, the change in packet size is responsible for the detection of these last few attack packets.

Figure 6.17 illustrates the anomaly scores of the DNS abuse traffic using the $m = 10$ Kitsune instance. The version of the attack with 100 incoming DNS queries per millisecond is shown in Figures 6.17a and 6.17b, while the version with one query per millisecond is shown in Figures 6.17c and 6.17d. In the figures belonging to the attack traffic-only anomaly scores (Figures 6.17a and 6.17c), we clearly see that the graphs contain two separate curves. One of the curves represents the anomaly scores of the incoming DNS queries, while the other represents the anomaly scores of the outgoing DNS responses. Compared to the SYN flood attack, this attack contains much less packets, and the speed at which the packets arrive is also much slower. Despite this, Kitsune is able to detect the attack in both circumstances with the $FPR = 0.001$ threshold. Kitsune is much better able to detect the version of the attack with the faster packet rate. When we examine the feature cluster that is most responsible for the high anomaly scores, we can see why this is the case: all of the features in the cluster are weight statistics. The amount of packets that Kitsune receives within a certain time



(a) Anomaly scores when the SSH brute-force attack traffic only is processed by Kitsune. The only malicious packets that are detected are at the very end of the graph. These correspond to the sending of SSH commands and replying with the requested data. The actual brute-force traffic is not detected.



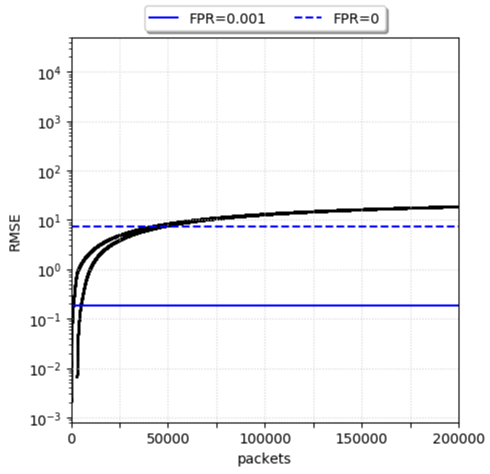
(b) Anomaly scores when the OS scan traffic is injected into the benign portion of the day 1 network trace. Executing this attack within the context of the rest of the network traffic has no effect on detection performance. The only malicious packets that are detected are the same ones as in Figure 6.16a.

Figure 6.16: SSH brute-force anomaly scores produced by Kitsune using maximum autoencoder size $m = 10$

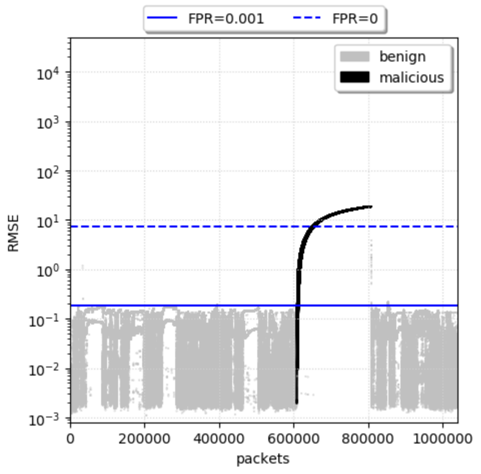
limit will make this statistic increase. Therefore, a higher packet rate will make this statistic increase faster. In Figure 6.17b we can clearly see an interesting effect that the attack has on the anomaly scores of the legitimate traffic. When the attack traffic ends, at the 800,000 mark on the x axis, we see a number of false positive instances that are not present on any of the other anomaly score graphs. These false positives are legitimate outgoing DNS packets. While the anomaly scores of the false positives quickly return to normal ranges, the attacker was able to deceive Kitsune into classifying legitimate traffic as anomalous, simply due to its proximity to the attack traffic. The results with the $m = 1$ Kitsune instance are very similar. In this case, the features most responsible for the high anomaly scores are weight statistics as well.

The $m = 10$ results of the actual DNS amplification attack are illustrated in Figure 6.18. Even though the packet rate during this attack 100 packets per millisecond (or 100,000 packets per second), the in the previous DNS abuse case, we can see that the anomaly score curve rises much slower and does not reach as high a value as in the Figures 6.17a and 6.17b. This is due to the fact that the incremental statistics are all computed per source IP address, and all the malicious traffic is divided between 10 different hosts. This is in contrast to the DNS abuse case, where the malicious traffic was coming from just a single IP address. Despite the lower anomaly scores, Kitsune is still able to detect most of the attack traffic, with a recall of 0.869 and precision of 0.990 using the $FPR = 0.001$ threshold. The $m = 1$ instance of Kitsune was also able to detect most of the attack traffic, although not as well as the $m = 10$ instance. Recall decreased slightly to 0.866, while precision saw a more considerable decrease, with its value dropping to 0.930.

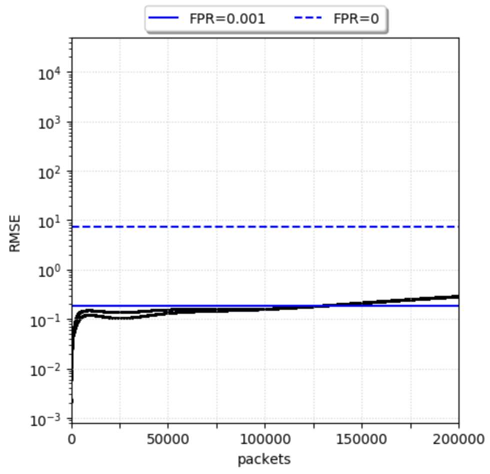
We can calculate the AUC for both $FPR = 0.001$ and $FPR = 0$ thresholds. These are illustrated in Figures 6.19a and 6.19b. We see that the AUC for $FPR = 0.001$, Kitsune



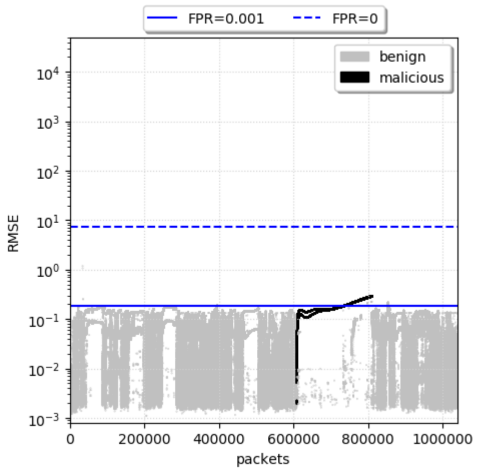
(a) DNS abuse, 100 queries/ms



(b) Attack injected into day 1 benign traffic, 100 queries/ms

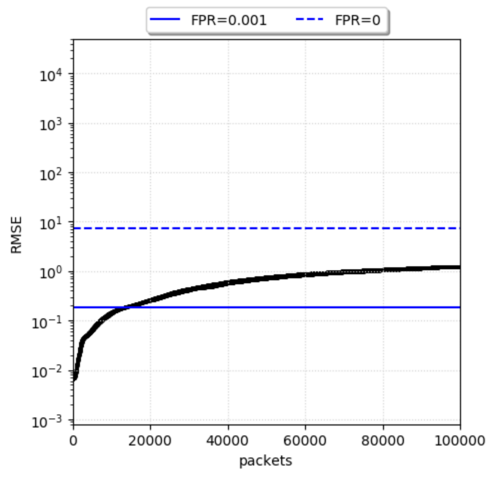


(c) DNS abuse, 1 query/ms

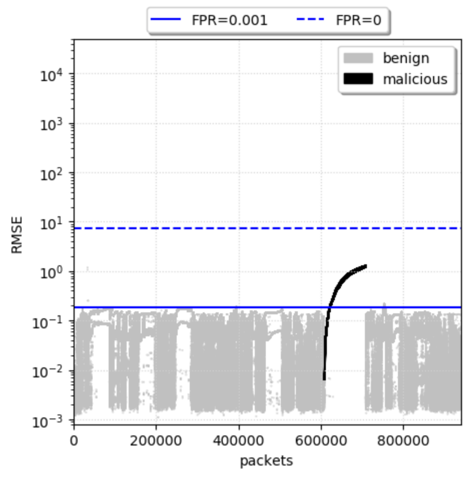


(d) Attack injected into day 1 benign traffic, 1 query/ms

Figure 6.17: Anomaly scores produced by Kitsune using maximum autoencoder size $m = 10$ when processing abusive incoming DNS queries for the purpose of creating a DNS amplification attack. As shown, the quicker we receive DNS queries, the quicker Kitsune is able to detect the malicious traffic. It makes no difference whether Kitsune processes the malicious DNS queries by themselves or within the context of the rest of the network traffic; detection performance remains the same.



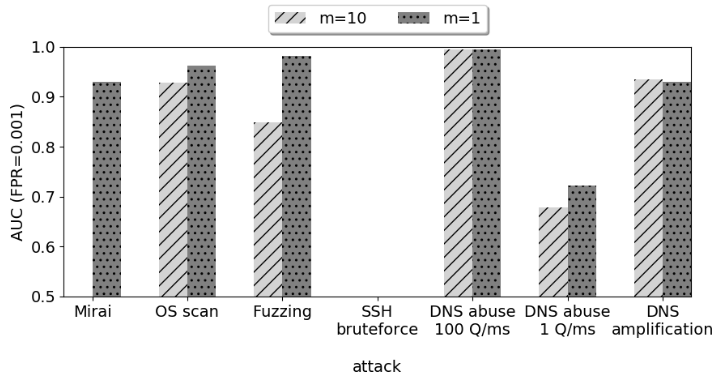
(a) Anomaly scores when the DNS amplification attack traffic only is processed by Kitsune.



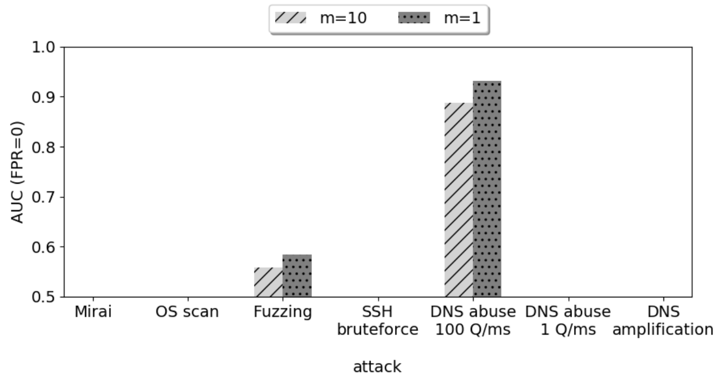
(b) Anomaly scores when the DNS amplification attack traffic is injected into the benign portion of the day 1 network trace.

Figure 6.18: DNS amplification anomaly scores produced by Kitsune using maximum autoencoder size $m = 10$. The speed at which the packets arrive is 100,000 packets per second. Though the attack consists of a single second's worth of DNS amplification traffic, Kitsune very quickly deems the traffic anomalous. As we were able to deduce from Figure 6.17, this is due to the high packet rate of the attack traffic.

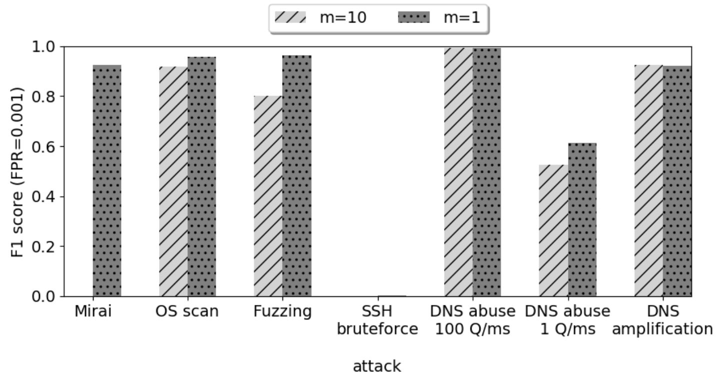
with $m = 1$ performs better than with $m = 10$. The exception here is for the SSH brute-force attack, for which Kitsune's performance is equally bad. The $FPR = 0$ threshold produces a poorly performing classifier, as we can see in Figure 6.19b, where the AUC for all attacks is rarely higher than 0.5. Since ROC curves, and therefore its AUC, are overly optimistic on imbalanced datasets [19], we also compute the F_1 scores (see Figures 6.19c and 6.19d). Interestingly, the computed F_1 scores agree with the AUC values. We see Kitsune's best performance using $m = 1$ and the $FPR = 0.001$ threshold, although SSH brute-force detection performance is very poor regardless of which threshold or parameter used. Mirai detection is much better with $m = 1$, which corresponds to the anomaly scores illustrated in Figure 6.11, compared to Figure 6.10. Using $FPR = 0$ as threshold, however, has a very negative effect on detection performance. Just as is apparent from the DNS abuse anomaly scores in Figure 6.17, the AUC and F_1 scores for the 100 queries/ms are significantly higher than for the 1 query/ms case. The SYN flood attack is absent from both figures, because it does not contain any true positive instances. Therefore, its AUC would be 0.5 and its F_1 score would be 0 in all cases.



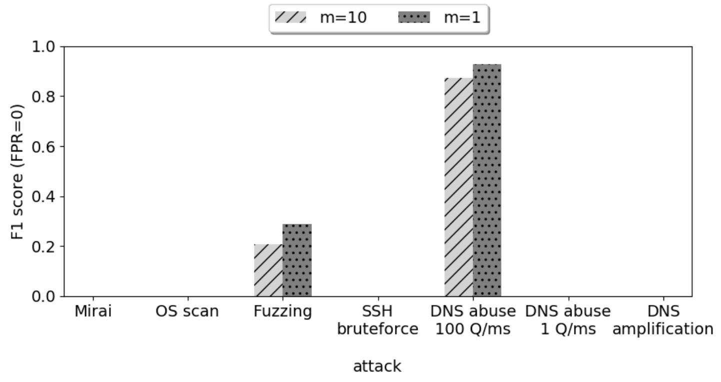
(a) AUC for $FPR = 0.001$ threshold



(b) AUC for $FPR = 0$ threshold



(c) F_1 scores for $FPR = 0.001$ threshold



(d) F_1 scores for $FPR = 0$ threshold

Figure 6.19: AUC and F_1 scores for different m and detection threshold

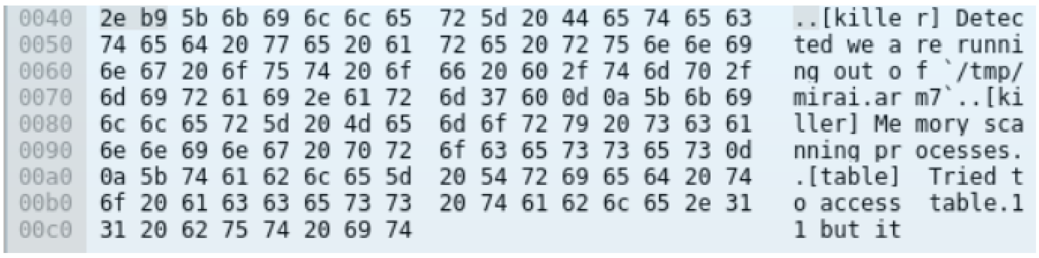
7 Discussion

The results in Chapter 6 were obtained after training Kitsune on real network traffic from a real-world network. From these results, we can make valid observations and conclusions about the performance of the Kitsune IDS. Additionally, we compare our results with those published by Mirsky et al. [47].

From the false positive rates and false negative rates illustrated in Figures 6.1 and 6.2, we can say two things. Firstly, network traffic is very difficult to predict. So much so, that an entire 24-hour network trace does not provide Kitsune with enough information to limit the amount of false positive instances in data from later days. Granted, most of the false positives corresponded to download traffic, which was not present in the training dataset to the degree of day 7, for instance (see Figure 6.3a). This just goes to show, however, that the behavior of hosts within a network varies substantially and is not fixed. This does not hold when the $FPR = 0$ threshold is used, though. In this case, however, the detection threshold is set so high that rarely any anomaly is detected, which would make the IDS useless. We must point out here that none of the packet traces used by Mirsky et al. during their research have a duration that exceeds even an hour. In the network we have used, a network trace of 24 hours is not enough to capture all legitimate behavior. In the work by Mirsky et al., however, it was decided that an hour's worth of traffic is very much representative of the network's overall behavior without any sort of substantiation. This might be explained by the fact that the networks only host a very limited number of services (mostly IP camera traffic and IoT device traffic), making the traffic within the network very uniform. This does not necessarily mean that the networks are unrealistic. The issue with this explanation is that it does not conform to the description of Kitsune in the original paper. There is absolutely no mention of Kitsune being a domain-specific IDS that is supposed to be used within networks with uniform traffic [47].

Secondly, we can also conclude from the extremely high false negative rates that the vast majority of malicious traffic within a network is very subtle. So subtle, in fact, that Kitsune regards it as "normal" traffic. From Table 6.1 we can see that all average FNRs are around 0.9999. Most of the malicious traffic in the network traces is port scanning. Kitsune deems these subtle port scans similar enough to legitimate traffic, even though none of the port scan traffic is present in the training dataset. This is in contrast to the positive results obtained by Mirsky et al [47]. As previously mentioned, the network traffic used by Mirsky et al. is very uniform. It is possible that the positive results are due to this fact. For instance, if most of the network traffic Kitsune has trained on is video traffic, any other type of network traffic will much more likely be seen as malicious, since it is a much less common type of traffic.

It is important to mention that before releasing their attack traffic datasets, Mirsky et al. made the decision to truncate all packets to 200 bytes due to privacy reasons [45]. The Mirai attack and OS scan are the two attacks for which we directly used the data released by Mirsky et al. Since the OS scan solely consists SYN packets that are 60 bytes in size, the truncation has an effect only on the Mirai attack traffic. Specifically, some of the Telnet packets between the attacker and victim IoT device were larger than 200 bytes and were truncated. See Figure 7.1 for one of such truncated Telnet packets. Notice the abrupt end of the text "`Tried to access table.11 but it`."



```
0040  2e b9 5b 6b 69 6c 6c 65  72 5d 20 44 65 74 65 63   ..[kille r] Detec
0050  74 65 64 20 77 65 20 61  72 65 20 72 75 6e 6e 69   ted we a re runni
0060  6e 67 20 6f 75 74 20 6f  66 20 60 2f 74 6d 70 2f   ng out o f `/tmp/
0070  6d 69 72 61 69 2e 61 72  6d 37 60 0d 0a 5b 6b 69   mirai.ar m7`..[ki
0080  6c 6c 65 72 5d 20 4d 65  6d 6f 72 79 20 73 63 61   ller] Me mory sca
0090  6e 6e 69 6e 67 20 70 72  6f 63 65 73 73 65 73 0d   nning pr ocesses.
00a0  0a 5b 74 61 62 6c 65 5d  20 54 72 69 65 64 20 74   .[table]  Tried t
00b0  6f 20 61 63 63 65 73 73  20 74 61 62 6c 65 2e 31   o access  table.1
00c0  31 20 62 75 74 20 69 74                            1 but it
```

Figure 7.1: Hex and ASCII sample of truncated Telnet packet

The implication of the release of this truncated data is that all results published by Mirsky et al. [47] are irreproducible. Any other researcher that would be interested in verifying their findings using their methods and datasets would be unable to do so. Consequently, we are forced to accept their results and conclusions at face value.

We know from He and Garcia [19] that ROC curves are too optimistic when evaluating classifiers that work with imbalanced data. Any statistics computed from ROC curves will therefore be overly optimistic as well. Even though anomaly detection inherently works with imbalanced datasets, Mirsky et al. use the AUC and EER statistics to evaluate their anomaly detection system [47]. Figure 7.2 illustrates the AUC for all of the attacks performed by Mirsky et al. [47] on their Kitsune instance. Notice Kitsune's high AUC values for the most of the attacks. Though they may seem impressive, little meaning can be attributed to these results due to the imbalanced datasets. Similarly to the results obtained by Mirsky et al. [47], our results in Figure 6.19a show a high AUC for the Mirai attack using $m = 1$, and the OS scan and fuzzing attack for both values of m. These AUC values are computed using a single threshold point, meaning that the true AUC will be higher than the one that illustrated in the figure [12].

Most of the attacks on which Kitsune achieves high F_1 scores are due to the presence of traffic floods. These are the Mirai, OS scan, and DNS amplification attacks.

Something to keep in mind is that even though the F_1 scores and AUC seem good for some of the attacks, this is only the case when the injected attack traffic is the *only* malicious traffic present in the dataset. If the attack traffic were injected into an unfiltered dataset, the AUC and F_1 scores would be significantly lower.

Mirsky et al. stated that using $m = 1$ as parameter, instead of $m = 10$, would yield better anomaly detection performance. This improvement in performance is almost negligible when classifying everyday traffic, as shown in Table 6.1. Classifying attack traffic that is injected into benign traffic, however, did have a notable effect. This is especially so in the case of the Mirai attack. Not a single malicious packet was detected by Kitsune using $m = 10$. When using $m = 1$, Kitsune was able to detect 91% of malicious packets. The cause of this increase in performance is the lack of feature clustering. With feature clustering, the entire cluster of features is used for computing anomaly scores. This makes Kitsune less sensitive to noise, since well-reconstructed features

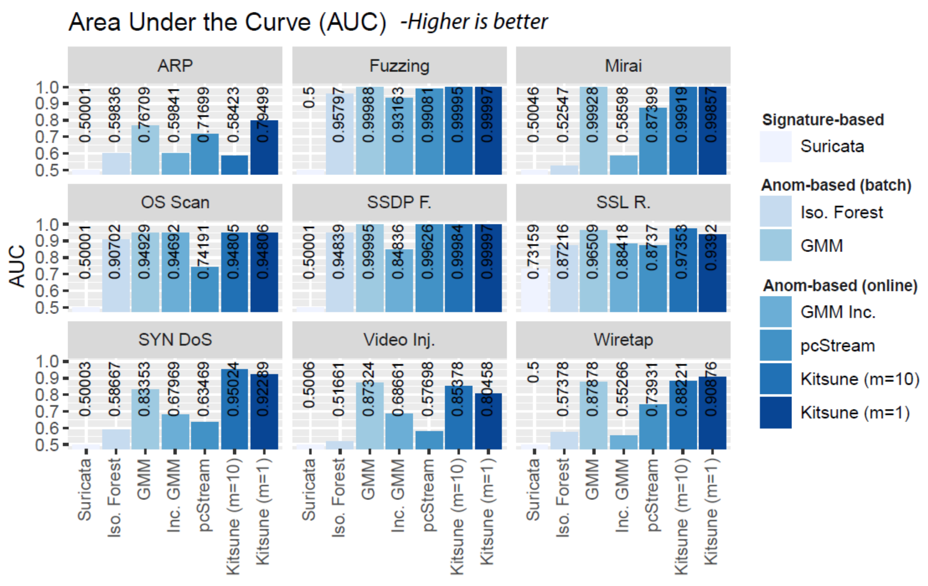


Figure 7.2: AUC values for the attacks performed by Mirsky et al. [47]

in the cluster will compensate for the single anomalous feature that would otherwise cause a large RMSE. There is a side effect to this, though. Using $m = 1$, clusters contain at most a single feature, meaning that there are no other features in the cluster that can compensate for a single anomalous feature. This can amplify the final anomaly score that is computed for a particular packet, especially if the anomalous features are reconstructed particularly poorly.

Given the anomalies that were successfully detected, it seems that Kitsune specifically looks out for two types of behavior:

1. sudden surges of traffic with little delay between each packet; *and*
2. inconsistencies in the amount of data exchanged between hosts.

These inconsistencies can be abnormally large packets coming from a certain host, as in the fuzzing attack, or an imbalance between the amount of data sent and received, as in the case illustrated in Figures 6.5 and 6.6. If we look at the design of the Kitsune IDS, we see that all features are computed using only packet counts, packet sizes and packet timestamps. Therefore, it makes sense that Kitsune would look out for the two aforementioned behaviors. No special attention is paid to the content of a packet's payload. Take the cases of the Mirai attack and OS scan, for instance. The detection of all malicious packets was based entirely on packet counts within a certain time window. While detecting the Mirai attack's ARP flood using packet counts is understandable, detection of the OS scan can be easily circumvented by limiting the speed at which the adversary sends out SYN packets. The processing of both versions of the DNS abuse cases also exemplifies Kitsune's preference for detecting sudden surges in traffic. We can see the attack with the higher packet rate produces high anomaly scores much faster than the attack with the slower packet rate. The DNS abuse traffic and DNS amplification attack also demonstrate the significance of the origin of a traffic flood. If, for a certain packet rate, all network traffic originates from a single IP address, Kitsune will more readily classify the flood as anomalous. Given that same packet rate, Kitsune will find it increasingly difficult to detect the flood as the number of participants in the flood attack increase. For example, if a botnet contains enough hosts, it is certainly possible that Kitsune will not be able to detect a DDoS attack initiated by that botnet if every host limits its packet rate properly.

In the cases of the fuzzing attack and SSH brute-force, the packets were detected by looking at the variation in the sizes of packets sent between two hosts. The SSH brute-force provides the clearest example of this. Most of the packets in the attack are under 200 bytes, which correspond to the login attempts. None of these packets are seen as malicious. It took just a single packet with a size of 3262 bytes sent between the hosts for the anomaly scores to spike. After this packet, the anomaly scores quickly return to the "normal" range. Thus, detection can be avoided by ensuring packet sizes do not exceed a certain size.

Though Kitsune was unable to make a single detection during the SYN flood, the curves illustrated in Figure 6.14 seem to have positive slopes. Given a larger SYN flood dataset, and a machine with the resources necessary to process it, the anomaly scores might eventually cross the $FPR = 0.001$ threshold. Just like this SYN flood, the OS scan is composed of only SYN packets. The only difference is that the SYN flood is directed at a single port, while the OS scan send out SYN packets to a large range of ports. Seeing that much of the OS scan was detected (see Figure 6.15), it is entirely possible that a SYN flood would be detected, given a larger dataset.

The fact that Kitsune does not take the payload into account when analyzing a packet is a serious drawback. Especially in larger networks, packets of all sizes can be found, meaning that packet size loses its value as a metric. Additionally, malicious packets can be crafted that are as large (or as small) as legitimate packets. Take Heartbleed packets, for instance, which are 74 bytes in size. It is certainly not uncommon to see legitimate SSH packets of that size (or legitimate heartbeat packets, for that matter). Malicious Heartbleed packets could erroneously be lumped together with legitimate traffic, simply because the packet is of a normal size.

With the results we obtained from our experiments, we have demonstrated two things. Firstly, we have found that when deploying Kitsune in a real-world environment and training it on real-world data, it is utterly ineffective as an IDS. It is very rarely able to detect malicious traffic, and its false positive rate is much to high for it to be useful in practice. Our network received an average of 690 MB of network traffic per day. Using the better-performing $m = 1$ version of Kitsune with the $FPR = 0.001$ threshold, Kitsune would flag 39 MB worth of packets as anomalies per day. Secondly, we have shown that Kitsune is only able to reliably detect malicious packets that are either too large or arrive to quickly one after the other (e.g. a DDoS attack). Detection of specific attacks can be evaded without much difficulty if the attacker knows the normal behavior of the network. This makes Kitsune essentially no more than a DoS or packet flood detector, as opposed to a general anomaly detection system.

These findings can be condensed into the following statement: flawed methods lead to flawed results. It is very likely that the authors could have spotted their errors if they ensured the data and methodology they used were valid. In fact, such mistakes could have been avoided in the first place. From the results obtained from this case study, as well as the numerous other IDS papers discussed in Chapter 2, it is apparent that usage of flawed data and methods are widespread. IDSs are thus evaluated in an incorrect manner, unfortunately leading to the generation of potentially erroneous information. Additionally, not every team of researchers uses the same processes when implementing and evaluating their designed IDS. This emphasizes the necessity for a set methodology that is designed to ensure proper evaluation of an IDS. For instance, many different authors often do not even use the same datasets when evaluating an IDS. On top of that, much of this data is flawed in some way, it being either unrealistic simulated data, or data extracted from an unrealistic or artificial network [36, 40, 73, 80]. The "gold standard" is network traffic that is extracted from a real-world network that is as large as one can get their hands on [73]. An issue with collecting real traffic is that none of it will be labeled. Since labeling is important for machine learning, it is necessary to engage in some form of collaboration with the administrator of the network in order to label the traffic accurately and extract the necessary ground-truth for classification. Of course, this is a time-consuming process, but there certainly is payoff in the form the ability to perform of valid experiments. This in turn leads to valid results. Finally, the evaluation process needs to be appropriate for the field of anomaly detection. Basing conclusions on different overly optimistic or uninformative metrics (e.g. ROC curves, accuracy) will also stand in the way valid evaluations and conclusions [19]. Agreeing on a common design and evaluation process would help researchers create valid research. Furthermore, such a common process will also allow for the easier reproduction of research by others, thereby helping the anomaly detection community as a whole as well. The steps that we have followed in our case study are extracted from academic literature. These steps can be aggregated and formalized to create a framework that is easy to follow and makes research more repeatable. This framework is presented in Section 8.3

8 Conclusion

Machine learning-based anomaly detection is a heavily studied field of research. Many researchers develop their own system using different techniques and evaluation methods. According to Sommer and Paxson [73], much of these methodologies are not suitable for the proper design and evaluation of intrusion detection systems. Other researchers will readily consider this previous research as flawless and completely truthful, and build their research upon this faulty foundation. To our knowledge, none of this research has been revisited in order to validate the findings. Neither have there been independent reviews that aim to replicate the results. It was for these reasons that we considered it an interesting endeavor attempting to do precisely that.

In our research, we selected one specific machine learning-based IDS that was trained and evaluated using network traffic from small and unrealistic networks. Much of this network traffic was very uniform and contained hardly any noise. With the work we performed, we have successfully demonstrated this observation. Specifically, we have shown that training and deploying a machine learning-based IDS in a real-world scenario will yield wildly different results compared to an artificial or unrealistic scenario. Real-world networks are busier and noisier than private or artificial networks. Undoubtedly, this fact will have a considerable effect on the performance of any such IDS. In this chapter, we come back to our main research question:

How can a machine learning-based intrusion detection system be correctly evaluated?

This chapter will elaborate on how we provided an answer to this question and how our research objective was achieved. Due to time constraints, we were only able to perform a single case study, using a single IDS for this project. This opens up several paths for future research, which will also be discussed in this chapter. Finally, we will present CAML-IDS, a framework that combines the best practices for anomaly detection research that we have discovered during our work on this project, which will allow for the correct assessment of machine learning-based intrusion detection systems.

8.1. Research objective

By combining the thorough literature analysis with the results obtained from our research itself, we are able to answer all of the sub-questions mentioned in Chapter 1. However, due to time constraints, we have only been able to propose a possible answer to our original research question. Several additional steps are necessary in order to provide a definitive answer to the research question. These will be discussed in Section 8.2, where we discuss possible future work. Below we will discuss the subquestions that we were able to answer.

Firstly, properly collecting data for a machine learning-based IDS requires the capture of real network traffic from a real-world network. This network is preferably as large as possible. Or rather, as large a network as one can get access to. Also, a sufficient amount of network traffic needs to be captured. This is important, because small network traces will not contain enough of the network traffic to adequately model the network's behavior.

In order to annotate the anomalous data correctly, we need a comprehensive description of the network from which the traffic is being captured. The description of the network is compiled by the administrator of the network, and, if necessary, is iteratively improved through cooperation between the researcher and administrator. A complete network description allows for the establishing of ground-truth, which is a crucial aspect of anomaly detection. By establishing ground-truth for the captured data, we can determine which data is benign and which is anomalous with absolute certainty.

When evaluating the IDS on attack traffic, we have to ensure that the injected attack traffic closely matches the actual traffic that is found within the network. If the attack traffic is simulated, or the attack was originally performed on different hosts, the traffic needs to be altered. Otherwise, these differences might influence the performance of the IDS, thereby invalidating any results that are obtained. For example, IP and MAC addresses need to be changed to ones that match the network.

Finally, the evaluation metrics that are used must be appropriate for the field of anomaly detection. Instead of simply using detection accuracy as a metric, precision and recall should be used, because these take false positive instances into account. Moreover, instead of the widely-used ROC curve, precision-recall (PR) curve should be used, since ROC curves are very optimistic when dealing with imbalanced datasets. In the case that a single value is desired that describes the performance of an anomaly-based IDS, the F_1 score can be computed as well.

By combining the aforementioned guidelines into a usable assessment framework, we provide a possible solution to the problem of properly evaluating the performance of machine learning-based intrusion detection systems. This framework still needs further research, and this will be elaborated upon in the next section.

8.2. Future work

Our work collects the best practices from across this field of research and combines them to form our proposed framework. The next step in this process is to evaluate the framework itself, which will provide definitive proof about its usefulness.

8.2.1. Evaluation of the framework

By applying this framework and to work by Mirsky et al. [47], this project demonstrates the failings in current anomaly detection research. However, even though this research demonstrates shortcomings in current research, the fact remains that this case study considers only a single IDS. There are many more anomaly-based IDS papers that can be revisited. By revisiting such previous research and reproducing it using the framework we have proposed, we can gain more insight into the validity of the framework. A key aspect of any academic studies is its reproducibility. Therefore, it is necessary to, not only test IDSs using this same framework, but also using the same data that was used in this study. This ensures that the only variable in the future experiments is the IDS itself. If further research demonstrates the validity of this framework, then, naturally, a good way forward is to adhere to our proposed framework when working on anomaly detection research in the future. Doing so would prevent researchers from making the errors that are so widespread in anomaly detection nowadays.

8.2.2. Revisiting previous research

Revisiting previous research is not only necessary for the evaluation of the framework, but it is also an opportunity to assess the validity of the research itself. More case studies such as this are necessary to make definitive conclusions about the entire field of research. As more of the research in this field is repeated using appropriate data and correct evaluation, more information will be available about the true state of anomaly detection research. Also, these repeats of existing research can be compiled together. Such a compilation of different case studies will researchers to better critique the field of anomaly detection.

8.2.3. Real-world testing

Lastly, it is certainly interesting to compare the steps as stated in this framework with the processes that are followed when implementing an IDS in a real-world scenario. This comparison can also be used to see whether this framework is appropriate for practical use. Sommer and Paxson [73] state that the most convincing evidence for the value and usefulness of an IDS is to get feedback network operators who have implemented said system in their network. This same statement can be applied to this framework. If network operators deem this framework useful for successfully implementing an effective IDS into their network, then it stands to reason that the framework certainly has value for both academic and practical use.

8.3. CAML-IDS

STEP	ACTION	REMARKS
1	Obtain access to a real-world network.	It is important that this network be as large possible
2	Obtain a detailed network description from network administrator.	
3	Create a static rule-set from the network description.	
4	Collect the necessary network traffic.	
5	Filter the collected network traffic using the static rule-set into (i) legitimate, (ii) illegitimate, and (iii) unclassified traffic.	Network traffic will be unclassified if it is not covered by any rule.
6	Take a subset of the unclassified traffic from Step 5 and refine the static rule-set by adding rules that cover the instances in the subset.	This is done in cooperation with the network administrator in order to determine with certainty whether the traffic is legitimate or not. This is important for establishing ground-truth.
7	Repeat Steps 5 and 6 until all unclassified traffic has been successfully annotated.	
8	Train the IDS on the ground-truth legitimate traffic obtained from Steps 5 and 6.	
9	Test the IDS using the entirety of legitimate and illegitimate traffic.	If attack traffic is injected into the dataset, ensure that this attack traffic matches the used network as closely as possible (e.g. IP addresses, MAC addresses, used protocols).
10	Evaluate the IDS performance using suitable statistics (e.g. precision, recall, PR curves, F1 score).	

Bibliography

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the Mirai Botnet. In *Proc. Usenix Security Symp.*, pages 1092–1110, 2017.

[2] ASHRAE. BACnet/IP. `http://www.bacnet.org/Tutorial/BACnetIP/sld004.html`. Accessed on 6/2/2019.

[3] S. Axelsson. The Base-Rate Fallacy and Its Implications for the Difficulty of Intrusion Detection. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 1–7. ACM, 1999. ISBN 1-58113-148-8.

[4] W.J. Bloem. Sluizen, gemalen en bruggen slecht beveiligd. `https://eenvandaag.avrotros.nl/binnenland/item/sluizen-gemalen-en-bruggen-slecht-beveiligd/`, 2012. Accessed on 26/2/2018.

[5] D. Bolzoni, S. Etalle, and P. Hartel. Poseidon: A 2-Tier Anomaly-Based Network Intrusion Detection System. In *Proc. IEEE International Workshop on Information Assurance (IWIA)*, pages 144–156. IEEE, 2006. ISBN 0-7695-2564-4.

[6] T. Bradley. Pros and Cons of Bringing Your Own Device to Work. `https://www.pcworld.com/article/246760/pros_and_cons_of_byod_bring_your_own_device_.html`. Accessed on 2/11/2018.

[7] CERT Division. 1996 CERT Advisories. `https://resources.sei.cmu.edu/asset_files/WhitePaper/1996_019_001_496172.pdf`. Accessed on 1/20/2019.

[8] Cloudfare. DNS Amplification Attack. `https://www.cloudflare.com/learning/ddos/dns-amplification-ddos-attack/`, . Accessed on 6/3/2019.

[9] Cloudfare. Ping of Death DDoS attack. `https://www.cloudflare.com/learning/ddos/ping-of-death-ddos-attack/`, . Accessed on 3/1/2019.

[10] A. Conole. sfuzz | Penetration Testing Tools. `https://tools.kali.org/vulnerability-analysis/sfuzz`. Accessed on 20/11/2018.

[11] S.E. Coull, C.V. Wright, F. Monrose, M.P. Collins, M.K. Reiter, et al. Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Network Traces. In *Proc. Internet Society Symposium on Network and Distributed System Security (NDSS)*, volume 7, pages 35–47, 2007.

[12] E.R. DeLong, D.M. DeLong, and D.L. Clarke-Pearson. Comparing the Areas Under Two or More Correlated Receiver Operating Characteristic Curves: A Nonparametric Approach. *Biometrics*, 44(3):837–845, 1988.

[13] O. Depren, M. Topallar, E. Anarim, and M.K. Ciliz. An Intelligent Intrusion Detection System (IDS) for Anomaly and Misuse Detection in Computer Networks. *Expert Systems with Applications*, 29(4):713–722, 2005.

[14] J. Erickson. *Hacking: The Art of Exploitation*. No Starch Press, 2nd edition, 2008. ISBN 0132126958, 9780132126953.

[15] T. Fawcett. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8): 861–874, 2006.

[16] C. Finseth. An Access Control Protocol, Sometimes Called TACACS. Technical report, 1993.

[17] J. Fogarty, R.S. Baker, and S.E. Hudson. Case Studies in the Use of ROC Curve Analysis for Sensor-based Estimates in Human Computer Interaction. In *Proc. Graphics Interface*, pages 129–136. Canadian Human-Computer Communications Society, 2005. ISBN 1-56881-265-5.

[18] D. Greene, P. Cunningham, and R. Mayer. Unsupervised Learning and Clustering. In *Machine Learning Techniques for Multimedia*, pages 51–90. Springer, 2008. ISBN 978-3-540-75171-7.

[19] H. He and E.A. Garcia. Learning from Imbalanced Data. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, (9):1263–1284, 2008.

[20] P. Hoffman. RFC 3207 - SMTP Service Extension for Secure SMTP over Transport Layer Security. RFC 3207, RFC Editor, February 2002. URL `https://www.rfc-editor.org/rfc/rfc3207.txt`.

[21] A. Huttunen, B. Swander, V. Volpe, L. DiBurro, and M. Stenberg. UDP Encapsulation of IPsec ESP Packets. Technical report, 2004.

[22] Imperva, Inc. The Top 10 DDoS Attack Trends. `https://www.imperva.com/docs/DS_Incapsula_The_Top_10_DDoS_Attack_Trends_ebook.pdf`. Accessed on 6/3/2019.

[23] J.P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani. Consensus Routing: The Internet as a Distributed System. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, pages 351–364, 2008. ISBN 978-1-931971-58-4.

[24] E. Joyce. Malfunctioning sensor adds to list of problems at San Onofre nuclear power plant. `https://www.scpr.org/news/2012/05/29/32601/malfunctioning-sensor-adding-list-things-wrong-san/`, 2012. Accessed on 26/2/2018.

[25] J. Jylänki et al. juj/kNet. `https://github.com/juj/kNet`. Accessed on 6/2/2019.

[26] Kaspersky Lab. What is Smurf Attack? `https://usa.kaspersky.com/resource-center/definitions/smurf-attack`. Accessed on 3/1/2019.

[27] K.S. Killourhy and R.A. Maxion. Toward Realistic and Artifact-Free Insider-Threat Data. In *Proc. Annual Computer Security Applications Conference (ACSAC)*, pages 87–96. IEEE, 2007. ISBN 0-7695-3060-5.

[28] R. King. Researcher: U.S. Tops the World in Critical Infrastructure Devices Connected to the Internet. `https://blogs.wsj.com/cio/2014/10/08/researcher-u-s-tops-the-world-in-critical-infrastructure-devices-connected-to-the-internet/`, 2014. Accessed on 20/2/2018.

[29] K. Labib and V.R. Vemuri. NSOM: A Tool to Detect Denial of Service Attacks Using Self-Organizing Maps. Technical report, 2002.

[30] K. Lakkaraju and A. Slagell. Evaluating the Utility of Anonymized Network Traces for Intrusion Detection. In *Proc. International Conference on Security and Privacy in Communication Networks (SecureComm)*, page 17. ACM, 2008. ISBN 978-1-60558-241-2.

[31] Lincoln Laboratory, Massachusetts Institute of Technology. Datasets. `https://www.ll.mit.edu/r-d/datasets`. Accessed on 3/1/2019.

[32] C. Liou, J. Huang, and W. Yang. Modeling Word Perception Using the Elman Network. *Neurocomputing*, 71(16-18):3150–3157, 2008.

[33] R. Lippmann, R.K. Cunningham, D.J. Fried, I. Graf, K.R. Kendall, S.E. Webster, and M.A. Zissman. Results of the DARPA 1998 Offline Intrusion Detection Evaluation. In *Proc. RAID Recent Advances in Intrusion Detection*, volume 99, pages 829–835, 1999.

[34] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, et al. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation. In *Proc. DARPA Information Survivability Conference and Exposition (DISCEX)*, volume 2, pages 12–26. IEEE, 2000. ISBN 0769504906.

[35] F. Liu, K.M. Ting, and Z. Zhou. Isolation Forest. In *Proc. IEEE International Conference on Data Mining (ICDM)*, pages 413–422. IEEE, 2008. ISBN 978-0-7695-3502-9.

[36] M. Mahoney and P.K. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In *Proc. RAID Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003. ISBN 3-540-40878-9.

[37] Mail.Ru. ICQ Privacy Policy. `https://privacy.icq.com/legal/privacypolicy/en`, . Accessed on 2/11/2018.

[38] Mail.Ru. Secure video calls with end-to-end encryption in ICQ. `https://icq.com/security-calls/en`, . Accessed on 2/11/2018.

[39] S. McCanne. libpcap: An Architecture and Optimization Methodolog for Packet Capture. `https://sharkfestus.wireshark.org/sharkfest.11/presentations/McCanne-Sharkfest'11_Keynote_Address.pdf`. Accessed on 7/10/2018.

[40] J. McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Trans. on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[41] P. Mell. Understanding Intrusion Detection Systems. In *IS Management Handbook*, pages 409–418. Auerbach Publications, 2003. ISBN 978-0849315954.

[42] Microsoft. [MC-DPLHP]: DirectPlay 8 Protocol: Host and Port Enumeration. https://msdn.microsoft.com/en-us/library/cc217240.aspx?f=255&MSPPError=-2147217396. Accessed on 6/2/2019.

[43] mIRC Co., Ltd. mIRC: Using SSL with mIRC. https://www.mirc.com/ssl.html. Accessed on 2/11/2018.

[44] Y. Mirsky. ymirsky/Kitsune-py: A network intrusion detection system based on incremental statistics (AfterImage) and an ensemble of autoencoders (KitNET). https://github.com/ymirsky/Kitsune-py. Accessed on 3/10/2018.

[45] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune Datasets. https://drive.google.com/drive/folders/1kmoWY4poGWfmmVSdSu-r_3Vo84Tu4PyE. Accessed on 10/11/2018.

[46] Y. Mirsky, B. Shapira, L. Rokach, and Y. Elovici. pcstream: A Stream Clustering Algorithm for Dynamically Detecting and Managing Temporal Contexts. In *Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 119–133. Springer, 2015. ISBN 978-3-319-18031-1.

[47] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. *Proc. Internet Society Symposium on Network and Distributed System Security (NDSS)*, 2018.

[48] S. Mukkamala, G. Janoski, and A. Sung. Intrusion Detection Using Neural Networks and Support Vector Machines. In *Proc. International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1702–1707. IEEE, 2002.

[49] R.S. Naoum, N.A. Abid, and Z.N. Al-Sultani. An Enhanced Resilient Backpropagation Artificial Neural Network for Intrusion Detection System. *International Journal of Computer Science and Network Security (IJCSNS)*, 12(3):11, 2012.

[50] Nmap. Nmap: the Network Mapper - Free Security Scanner. https://nmap.org/. Accessed on 30/1/2019.

[51] Omron Europe B.V. FINS Commands Reference Manual. https://www.myomron.com/downloads/1.Manuals/Networks/W227E12_FINS_Commands_Reference_Manual.pdf. Accessed on 6/2/2019.

[52] OpenBSD. OpenBSD. https://www.openbsd.org/. Accessed on 2/11/2018.

[53] OpenSSH. sshd(8) - OpenBSD manual pages. https://man.openbsd.org/sshd. Accessed on 2/11/2018.

[54] K. Packard. X Display Manager Control Protocol. ftp://www.x.org/pub/X11R7.6/doc/libXdmcp/xdmcp.html. Accessed on 6/2/2019.

[55] A. Palczewska, J. Palczewski, R.M. Robinson, and D. Neagu. Interpreting Random Forest Classification Models Using a Feature Contribution Method. In *Integration of Reusable Systems*, pages 193–218. Springer, 2014. ISBN 978-3-319-04717-1.

[56] R. Pang, M. Allman, V. Paxson, and J. Lee. The Devil and Packet Trace Anonymization. *ACM Computer Communication Review (CCR)*, 36(1):29–38, 2006.

[57] I. Paul. 70-plus XMPP messaging services now securing chats with TLS encryption. `https://www.pcworld.com/article/2157180/xmpp-services-push-encrypted-connections-by-default.html`. Accessed on 2/11/2018.

[58] J. Postel. Internet Control Message Protocol. RFC 792, RFC Editor, September 1981. URL `https://www.rfc-editor.org/rfc/pdfrfc/rfc792.txt.pdf`.

[59] J. Postel. Transmission Control Protocol. RFC 793, RFC Editor, September 1981. URL `https://www.rfc-editor.org/rfc/rfc793.txt`.

[60] ProComSol, Ltd. Introduction to HART-IP. `https://procomsol.com/files/downloads/White%20Paper%20-%20HART-IP%20Introduction.pdf`. Accessed on 6/2/2019.

[61] radware. Teardrop Attack. `https://security.radware.com/ddos-knowledge-center/ddospedia/teardrop-attack/`. Accessed on 2/3/2018.

[62] D. Reed. RFC 1324 - A Discussion on Computer Network Conferencing. RFC 1324, RFC Editor, May 1992. URL `https://www.rfc-editor.org/rfc/rfc1324.txt`.

[63] M. Riedmiller and H. Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Proc. IEEE International Conference on Neural Networks (ICNN)*, pages 586–591. IEEE, 1993. ISBN 0-7803-0999-5.

[64] D. Robinson and K. Coar. RFC 3875 - The Common Gateway Interface (CGI) Version 1.1. RFC 3875, RFC Editor, October 2004. URL `https://www.rfc-editor.org/rfc/rfc3875.txt`.

[65] T. Saito and M. Rehmsmeier. The Precision-Recall Plot Is More Informative Than the Roc Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS One*, 10(3):1–21, 2015.

[66] S. Sanfilippo. Encrirc - Client independent encryption proxy for IRC. `http://www.hping.org/encrirc/`, . Accessed on 2/11/2018.

[67] S. Sanfilippo. Hping - Active Network Security Tool. `http://www.hping.org/`, . Accessed on 1/3/2019.

[68] Scapy. Scapy. `https://scapy.net/`. Accessed on 7/10/2018.

[69] M.E. Schuckers. Receiver Operating Characteristic Curve and Equal Error Rate. In *Computational Methods in Biometric Authentication*, pages 155–204. Springer, 2010. ISBN 978-1-84996-202-5.

[70] N Shone, T.N. Ngoc, V.D. Phai, and Q. Shi. A Deep Learning Approach to Network Intrusion Detection. *IEEE Trans. on Emerging Topics in Computational Intelligence (TETCI)*, 2(1):41–50, 2018.

[71] M. Sokolova and G. Lapalme. A Systematic Analysis of Performance Measures for Classification Tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[72] R. Sommer. *Viable Network Intrusion Detection in High-Performance Environments.* PhD thesis, Technische Universität München, 2005.

[73] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pages 305–316. IEEE, 2010. ISBN 978-0-7695-4035-1.

[74] N. Srivastav and R.K. Challa. Novel Intrusion Detection System Integrating Layered Framework with Neural Network. In *Proc. IEEE Advance Computing Conference (IACC)*, pages 682–689. IEEE, 2013.

[75] S.J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P.K. Chan. Cost-Based Modeling for Fraud and Intrusion Detection: Results from the JAM Project. In *Proc. DARPA Information Survivability Conference and Exposition (DISCEX)*, volume 2, pages 130–144. IEEE, 2000. ISBN 0769504906.

[76] Suricata. Suricata | Open Source IDS / IPS / NSM engine. `https://suricata-ids.org/`. Accessed on 22/11/2018.

[77] Synopsys, Inc. Heartbleed Bug. `http://heartbleed.com/`. Accessed on 1/3/2019.

[78] Peter Szor and Eric Chien. CodeRed II. `https://www.symantec.com/security-center/writeup/2001-080421-3353-99`. Accessed on 3/1/2019.

[79] A.S. Tanenbaum and D.J. Wetherall. *Computer Networks.* Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2010. ISBN 1593271441, 9781593271442.

[80] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani. A Detailed Analysis of the KDD Cup 99 Data Set. In *Proc. IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pages 1–6. IEEE, 2009. ISBN 978-1-4244-9941-0.

[81] tcpdump. TCPDUMP/LIBPCAP public repository. `http://www.tcpdump.org/`. Accessed on 7/10/2018.

[82] The Hacker's Choice. vanhauser-thc/thc-hydra: hydra. `https://github.com/vanhauser-thc/thc-hydra`. Accessed on 1/3/2019.

[83] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Academic Press, 4th edition, 2008. ISBN 9781597492720.

[84] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer Two Tunneling Protocol "L2TP". RFC 2661, RFC Editor, August 1999. URL `https://www.rfc-editor.org/rfc/rfc2661.txt`.

[85] University of California. KDD Cup 1999 Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, . Accessed on 3/1/2019.

[86] University of California. KDD-CUP-99 Task Description. `http://kdd.ics.uci.edu/databases/kddcup99/task.html`, . Accessed on 20/1/2019.

[87] University of New Brunswick. NSL-KDD. `https://www.unb.ca/cic/datasets/nsl.html`. Accessed on 20/1/2019.

[88] J.C.A. van der Lubbe. *Basic Methods of Cryptography.* Cambridge University Press, 1998. ISBN 978-9065623461.

[89] K. Wang and S.J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In *Proc. RAID Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004. ISBN 3-540-23123-4.

[90] Z. Zhang, M.W. Beck, D.A. Winkler, B. Huang, W. Sibanda, H. Goyal, et al. Opening the Black Box of Neural Networks: Methods for Interpreting Neural Network Models in Clinical Applications. *Annals of Translational Medicine*, 6(11), 2018.