Enhancing Network Intrusion Detection Robustness via Dataset Augmentation: A CIC-IDS-2017 Case Study

Jozef Jankaj

May 2025

1 Introduction

A lot of time has passed since that fateful night of October 29, 1969, when the first inter-network communication took place through the ARPANET link. In his statement in an interview with Gregory Gromov

> ”Then we typed the G, and the system crashed... Yet a revolution had begun” [12]

who knows if prof. Leonard Kleinrock had fully realized just how massive the revolution was going to be. Since then, the internet has grown exponentially: from a small group of four nodes, through the invention of the World Wide Web by Sir Tim Bernes-Lee for the sharing of academic knowledge, via the dot-com bubble of the 2000's, to an inseparable part of our lives. As our lives have moved into cyberspace, so have threats against our privacy. We no longer face only the criminals on the streets, but also a threat from cybercriminals on the other side of the world. As such, protection measures have been developed, chief among which are Network Intrusion Detection Systems (NIDS). These systems monitor traffic within the network and send out alerts when malicious traffic is encountered, or even cut the connection altogether.

In recent years, NIDS research has explored how the capabilities of machine learning methods can be utilized to protect our networks. While many papers claim excellent performance and groundbreaking contributions, the field of machine learning-based NIDS (ML-NIDS) has been met with skepticism by practitioners [2], chief among them being the lack of evidence of generalization to different networks.

The aim of this work is to address this complaint by verifying a method of enhancing robustness of NIDS datasets through dataset augmentation. As a point of study, we take the CIC-2017 dataset [27] created by the Canadian Institute for Cybersecurity.

This work is structured as follows: In Section 2, we explore the literature surrounding ML-NIDS, illustrating current methods and datasets used in the field. In Section 3, we formulate a hypothesis behind the poor generalization performance of ML-NIDS models, explore the CIC-2017 dataset in-depth and recreate it using ConCap [**concap**]. In Section 4, we present results of our testing. Finally, in Section 5, we draw conclusions and reflect on future work.

2 Literature study

NIDS systems inherently belong to the category of classification problems: determine whether a particular network traffic is benign or malicious.

One of the first implementations of a Network Intrusion Detection System (NIDS) has been NADIR (Network Anomaly Detection and Intrusion Reporter) [14] from 1990. This system was developed for the Integrated Computing Network at Los Alamos National Laboratory to automate the detection of unauthorized access to the system. NADIR utilized statistical profiles of network users to detect anomalies, notifying and printing out abnormalities in the profiles as detected by manually designed expert rules.

NADIR is an example of what we call "expert-made NIDS": It follows a strict set of rules to detect and report unauthorized activity, programmed by an expert. While these systems work well as first lines of defense, they also suffer from a fundamental flaw: they can only detect what they were specifically programmed by their creators to detect. This is a difficult task: given the sheer amount of information going through the network at any particular time, one does not (or can not) know which pieces of data point to malicious traffic. Furthermore, malicious packets in one context may be harmless in another.

Instead of having to manually preprogram the NIDS with attack profiles, machine learning (ML) allows us a model to learn what (in our case) malicious traffic looks like by observing examples of benign and malicious traffic, leading directly to the concept of ML-NIDS: Machine Learning-based Network Intrusion Detection System.

In general, ML-NIDS can be split into two subgroups: signature-based and anomaly based. Signature-based ML-NIDS is trained to

2.1 Machine Learning methods

Machine-learning methods are a class of algorithms that have the computer infer relations and rules of a dataset by being exposed to training samples. As our focus is network intrusion detection, we focus on methods that perform classification: determination whether a particular sample is malicious or benign. These methods can generally be subdivided into two categories: Shallow Learning and Deep Learning.

2.1.1 Shallow Learning

Some of the most widely known and used machine learning algorithms are:

1. K-Nearest Neighbours (KNN) [6]: Initially developed by Fix and Hodges in 1951 [9] and later improved by Cover and Hart in 1967 [6], these models can be used for multiclass classification. Predictions are formed by a majority vote, where the label of samples most similar to the input is the prediction.

2. Support Vector Machines (SVM) [5]: SVMs, originally developed by Cortes and Vapnik in 1995, are models used for binary classification. The model learns to distinguish between two labels by transforming inputs into a high-dimensional space. In this ”feature” space, a hyperplane can be constructed to distinguish between the two labels. The binary classification can be extended into n-label classification by training a separate SVM (target label vs anything else) for each label and perform classification by having each SVM predict the label, with a unique SVM predicting its assigned label instead of ”anything else”.

3. Decision Trees (DT) [33]: Decision trees form a tree data structure, with each internal node testing for a particular feature in the data, reaching a class label in one of the leaf nodes.

4. Ensemble methods [22]: Ensemble methods combine multiple simpler classifiers to form a larger one. This way, weaknesses of the underlying classifier (e.g. an SVM being unable to separate multiple classes) can be overcome (a separate SVM being trained for each class label, as described above).

Liao and Vemuri [17] use a KNN classifier for intrusion detection by building a profile of the input process consisting of the occurrences of system calls the process performs.

Sahu and Mehtre [25] implement the J48 Decision Tree, testing it on Kyoto 2006+ dataset. Their trained model boasts 97.23% accuracy and a true positive rate of 99% for normal and attack packets.

Bao, Xu and Hou [3] build an NIDS system consisting of two SVM models: one for anomaly detection, the other for misuse detection using previously seen attack signatures. Authors do make claims about ”high training rate and decision rate, insensitiveness to dimensions of input data, continuous correction of various parameters with increase in training data which endows the system with self-learning ability, and so on.”, yet they do not support this with figures, measurements or experiment setup.

2.1.2 Deep Learning methods

Deep Learning methods are a subset of machine learning methods that utilize neural networks for learning. Neural networks consist of artificial neurons: nodes that have a particular activation function. Similarly to biological systems, these nodes are connected together to form layers. Inputs propagate through the layers and result in some activation in the final layer [34]. For the purposes of classification, this final layer could, for example, consist of one neuron for each classification class.

Depending on the network topology and the layers utilized, different models arise. [16]

- Fully connected network: The most general case, where each layer of neurons is fully connected to every other neuron in the previous layer. In order to perform any kind of advanced classification, handcrafted features need to be extracted, making these models difficult to work with in areas where irrelevant details should be ignored (e.g. the position of a dog in the image), yet small details matter for classification (e.g. differences between a black dog and a black cat).

- Convolutional Neural Networks (CNN): These networks are constructed to perform classification by putting together different parts of the input. The network is architectured as a series of convolutional layers and pooling layers, each progressively extracting increasingly higher-level features. Due to their ability to extract features by themselves, without expert intervention, CNNs and models based on them (such as autoencoders described below) are widely used for classification and detection.

- Autoencoders: A special type of neural network, with the input being the same as the output. The network can be separated into 3 parts: a group of layers called Encoder that gradually reduces the size of layers, a latent or hidden layer, and a group of layers called a Decoder that is, essentially, the Encoder inverted. The network attempts to learn an efficient encoding of the input such that it can recreate it in the output.

Song et al. [31] perform an analysis of autoencoders in ML-NIDS, where an autoencoder is trained to recognize normal behaviour. Any input that the autoencoder fails to recreate below a particular error margin is classified as malicious.

Pham et al. [23] propose a six-layer 1D CNN model, consisting of four convolutional layers and two fully connected layers. They compare this model with different Shallow Learning models (Gaussian Naïve Bayes, Logistic Regression, KNN, SVM, AdaBoost, Gradient Boosting, XGBoost, CatBoost, LightGBM) and show that their CNN model outperforms them on the UNSW-NB15[19] and NSL-KDD [35] datasets, achieving 99.3% accuracy on UNSW-NB15 and 99.43% accuracy on NSL- KDD.

2.2 Dataset quality

Accuracy of AI models is heavily dependent on high-quality data: the well-known principle of ”garbage in, garbage out” applies. High-quality datasets are therefore imperative for well-performing models, not only on the dataset, but also in general networks. Some of the most popular NIDS datasets (according to citations[1]) are:

- CIC-IDS-2017/CSE-CIC-IDS2018 [27, 1] (4623 citations): CIC-IDS-2017 dataset and its cousin CSE-CIC-2018 are all-purpose NIDS datasets, comprising of seven different attack classes executed over the course of multiple days. CIC-IDS-2017 is the base, comprising 14 hosts, while CSE-CIC-2018 is much larger in scale, being implemented on Amazon Web Services as

[1] As observed on Google Scholar on 03-05-2025

a simulation of a realistic company network: fifty attacker machines, 420 victim machines spread over five departments and thirty servers, 500 machines in total.

- ISCX IDS 2012 [30] (1558 citations): A predecessor to CIC-IDS-2017, ISCX IDS 2012 implements its attacks as overlapping "scenarios" defined as unambiguous α and β profiles, focusing on HTTP, SMTP/IMAP, FTP and SSH traffic.

- UNSW-NB15 [19] (4001 citations): UNSW-NB15 dataset is built using the IXIA attack generator on a testbed consisting of three virtual servers, two for normal traffic and one for attacks.

However, many issues have been found with these datasets.

Engelen et al. [8] analysed CIC-IDS-2017 dataset in depth and found numerous issues: "misimplementation of DoS Hulk attack, misunderstanding of the TCP protocol in flow construction ...", but also a "lack of documentation concerning flow construction and parameters of attacks".

Similarly, Flood et al. [10] found issues with UNSW-NB15, concluding that "without modification, UNSW NB15 is unsuitable for evaluating the ability of classifiers to generalise between attack categories". They, too, note lacking documentation in other datasets (ToN_IoT, UNSW-NB15 and CIC-IDS-2017/CSE-CIC-18).

2.3 ConCap

A prevailing trend in NIDS dataset construction is the usage of physical networks and testbeds. While useful, these datasets have inherent constraints that limit their usability.

First, these datasets are difficult, if not impossible, to reproduce. Researchers cannot always invest the time and resources to rebuild the networks utilized in the datasets and are therefore forced to "trust" the dataset authors that the methodology they utilise in the construction is not flawed. The lack of documentation found with some datasets further hinders their reproducibility and, ultimately, trust in these datasets.

Second, any potential errors that may have slipped into the dataset cannot be easily corrected by reconstructing it. As an example, we mention the misimplementation of DoS Hulk attack found by [8]. Due to the lack of code or precise descriptions of attack execution, we are left to guess what specific version of Hulk was used and with what configuration. While this example is easy to correct, it nevertheless shows a possibility of errors slipping into datasets that NIDS practitioners might not notice at first.

The ConCap tool by Verkerken et al. [**concap**] has been proposed as a new method for ML-NIDS dataset generation. ConCap generates traffic using scenario files using a Kubernetes cluster, in which the targets are specified as Docker containers. This method solves both abovementioned limitations at once: reproducibility is guaranteed, as the dataset does not only consist of traffic capture files, but primarily of scenario files. Furthermore, the dataset can be easily extended with new attack classes or variants of existing attack classes, down to individual options with which the attack tools are configured.

Verkerken et al. have previously verified these properties by simulations with the Bruteforce attack class, where a model was unable to correctly identify NetFlows coming from `patator -P=0` configuration, but after retraining with this configuration, the model was able to identify it without losing its ability to detect attacks with `patator -P=1` option.

This work builds on this research by almost fully translating the CIC-IDS-2017 dataset to ConCap scenarios.

3 Methodology

Thus, we hypothesize that the key to increasing model robustness lies in the construction and augmentation of a synthetic dataset. In-silico construction immediately solves two major problems with NIDS datasets: it is, by design, fully documented and reproducible, as the dataset no longer consists of the actual PCAP files: it is the descriptions of the network and the attacks that form the dataset. Datasets constructed in this way can also be easily extended with new classes of attacks, variants of existing classes of attacks, and new network configurations.

The previously mentioned ConCap framework [**concap**], built on Kubernetes and Docker, has been proposed for the synthetic dataset construction that we describe above. We use ConCap to verify our hypothesis and employ the following methodology: in Section 3.1, we analyse the CIC-IDS-2017 and the literature surrounding it in depth. In Section 3.2, we describe the reconstruction process and any significant choices we make. Then finally, in Section 3.3, we describe our testing methodology to prove or disprove the hypothesis.

3.1 Analysis of CIC-IDS-2017

CIC-IDS-2017 dataset has been constructed by the Canadian Institute for Cybersecurity in 2017 to address the then-prevalent issues with existing NIDS datasets. Dataset authors focus on generating realistic background traffic, in addition to seven attack classes: Bruteforce, Denial-of-Service, Heartbleed, Web Attack infiltration, Botnet, Distributed-Denial-of-Service and Portscanning. The dataset consists of traffic captures as PCAP files during the work week of July 3, 2017.

The dataset contains a relatively simple victim network (compared to its larger cousin, CIC-IDS-2018), consisting of a Windows Server, a Ubuntu 16 webserver and a Ubuntu 12 server, as well as two Ubuntu 14.04, two Ubuntu 16.04, a Windows 7, Windows 8.1-64, Windows Vista, Windows 10 Pro and a Macintosh computers, nice in total. The attacker network consists of one Kali Linux and three Windows 8.1 machines. The attackers communicate with the victim network over the internet, appearing in the PCAP files under the IP address `172.16.0.1`. There is further network infrastructure present, but it is inconsequential for our purposes. Figure 1 [27] shows a schematic of the testbed.

Most attacks (Bruteforce, DoS, Web Attacks, DDoS, Portscan) are conducted against a single host, the Ubuntu 16.04 webserver. We reconstruct this webserver as faithfully as possible, basing ourselves on the services revealed by the Portscan attack on Friday.

3.1.1 Monday

On Monday, no attacks take place, only background traffic is happening. The authors utilise a B-Profile system [28] for benign traffic generation ”to profile abstract behavior of human interactions and generates naturalistic background

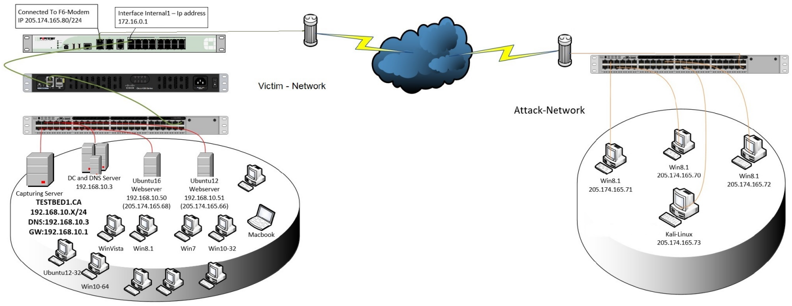

Figure 1: Architecture of CIC-IDS-2017 testbed

traffic"[27]. The traffic capture of this day consists of "abstract behaviour of 25 users based on the HTTP, HTTPS, FTP, SSH, and email protocols." [27] and provides an excellent source of benign samples that we later use to balance the training sets.

3.1.2 Tuesday

On Tuesday, in addition to the background traffic as seen on Monday, we see the first attack class take place: Bruteforce. In the morning, an FTP Bruteforce attack is performed using Patator [36]. This attack is run against the iscxtap user, with various passwords being tried from an unknown wordlist. In the afternoon, SSH Bruteforce attack is performed using presumably the same tool. Due to inherent encryption of SSH traffic, it is unclear against which user the attack is conducted or what wordlist is used for passwords.

3.1.3 Wednesday

On Wednesday, Denial-of-Service (DoS) attacks are launched against the Ubuntu 16 webserver. The dataset consists of four DoS attacks conducted with the tools Slowhttptest [29], GoldenEye [26] and HULK [4]. In the afternoon, the vulnerable OpenSSH server falls victim to the Heartbleed bug [13] exploitation.

Slowhttptest attacks the server by heavily fragmenting the HTTP request body into small chunks and slowly sending them to the server. By slowly receiving the chunks, the server is forced to maintain the connection and waste resources. As the server is limited in how many requests it can serve at any given time, a denial of service occurs.

Slowloris works similarly to Slowhttptest, but instead of fragmenting the body, the Headers section of the HTTP request is fragmented. These fragmented headers are then slowly sent to the server, again forcing it to keep the connection alive and prevent legitimate requests from being processed.

GoldenEye abuses the HTTP `Connection` and `Cache-Control` headers to maintain the open connection. Through the Connection HTTP header, the server can be instructed to keep the connection alive (through the `keep-alive` option) to allow additional requests to take place over the same connection. Cache-Control header contains instructions for the browser and shared caches. By setting this header to `no-cache,max-age=0`, it effectively disables connection busting through caches. GoldenEye abuses these headers by opening a large number of connections, effectively exhausting the connection pool of the server, and denying service to legitimate clients.

Http Unbearable Load King (HULK) attacks the server by flooding it with raw UDP packets. As UDP does not require a three-way handshake to set up a connection like TCP does, this attack can simply flood the server with packets it needs to handle, overloading it. HULK works best as a distributed attack with multiple malicious clients flooding the target. However, this attack is also easy to stop through a proper firewall configuration.

The Heartbleed bug [13] was first discovered in 2014 as a vulnerability in the implementation of the Heartbeat extension proposed by RFC6520. First introduced in December 2011, this bug allows anyone to read the memory of the SSH server due to a missing bounds check [21]. As the process memory usually contained the unencrypted secret key used to encrypt communications, it became the primary target of Heartbleed exploits, allowing subsequent decryption of previous and ongoing encrypted communications. CIC2017 authors implemented this attack using the Heartleech tool [11] against a vulnerable OpenSSL server version 1.0.1f.

3.1.4 Thursday

On Thursday, authors implement a Web Attack and an Infiltration attack.

Web attack consists of attacking the Damn Vulnerable Web App (DVWA) hosted on one the webservers. The authors attack the application by performing a bruteforce attack on the login page of the application and by performing a SQL injection and Cross-Site-Scripting (XSS) attack. SQL injection attack exploits careless replacements of variables in strings. Malicious strings can alter the behaviour of the query, leaking private information hidden within the database. XSS attacks occur when the attacker gets the webapplication to inject malicious code into the that is run for another user. In this instance, SQL injection is performed against the dedicated endpoint for practicing SQL injections provided by DVWA and the XSS attack is conducted by leaking a cookie into the browser's console.

Infiltration attack is executed by having the user execute an infected file on the victim network, which in turn sets up a link through the Metasploit Framework [24] and allows the attacker to execute portscan of the entire victim network through NMap [37].

3.1.5 Friday

On Friday morning, the authors conducted a botnet attack through the Ares [38] tool, enslaving the computers in the victim network and getting them to take a screenshot of the desktop.

On Friday afternoon, a Distributed Denial-of-Service (DDoS) attack is conducted against the Ubuntu 16 webserver using the Low Orbit Ion Cannon (LOIC) [20]. This tool is started on the Windows 8.1 hosts in the attacker network and together, they conduct the DDoS attack.

Finally, last attack class is introduced: Portscan. While we have previously seen this attack as a part the infiltration attack. here we observe it in its raw form. All Windows machines in the victim network are portscanned for running services, using ”the main NMap switches such as sS, sT, sF, sX, sN, sP, sV, sU, sO, sA, sW, sR, sL and B.” [27]. In our analysis of the raw PCAP files, we find that switches sF, sX, sP and sA are missing entirely: there is no traffic at the times specified by the dataset authors, nor is there any traffic present that would arise from these switches.

3.2 Reconstruction

As a second step of verifying our hypothesis, we reconstruct the dataset within the ConCap framework.

ConCap operates on a Kubernetes cluster, spinning up pods that communicate with each other. By defining a scenario file in the YAML format, users can easily spin up containers to perform a wide variety of tasks. ConCap is designed to capture traffic between pods and analyse it using traffic analysis tools such as CICFlowmeter [15] and Rustiflow [32]. As ConCap is primarily aimed at constructing attack traffic In order to construct a scenario, the user can define the attacker and target images, the network configuration and the labelling of generated NetFlows from the traffic occuring on the network. In order to supply the attack tools with the IP addresses of targets, an attack command can be specified with environment variables which will be resolved at runtime.

Having analysed the documentation surrounding the dataset, we notice previously mentioned issues of poor documentation. This leads us to make a number of decisions and guesses in our reconstruction. While we try to remain as faithful as possible to the original dataset, we acknowledge that our implementation may not produce 100% equivalent network traffic. As our goal is improving the robustness, we focus on reconstructing the attacks and sample the benign traffic from Monday.

3.2.1 Tuesday

In reconstructing the Tuesday attacks, we use Patator by lancejot [36], more specifically its version tagged 0.8. We construct a Docker image containing patator and a list of usernames and passwords to try against the server. We build this image under the name `themessik/patator` and push it to Dockerhub. It is unclear what dictionaries authors use for the bruteforce attacks, forcing us to make an arbitrary choice. We choose for a username and password list from the SecLists [18] repository: Top Usernames Shortlist[2] and Pwdb Top 1000[3]

3.2.2 Wednesday

For the DoS attacks, we use the respective tools mentioned by the authors. We use the shekyan's slowhttptest [29] tool to reconstruct both Slowhttptest and Slowloris attacks, as this tool provides required functionality for both. For GoldenEye and HULK, we build dedicated Docker images (`themessik/goldeneye` and `themessik/hulk` respectively) and point them at the webserver.

For the Heartbleed attack, we build dedicated target image (`themessik/heartbleed`) based on Ubuntu 16.04 and install the Heartbleed-vulnerable version of OpenSSL 1.0.1f, mirroring the authors. For the attacker image (`themessik/heartleech`),

[2] `https://raw.githubusercontent.com/danielmiessler/SecLists/refs/heads/master/Usernames/top-usernames-shortlist.txt`

[3] `https://raw.githubusercontent.com/danielmiessler/SecLists/refs/heads/master/Passwords/Common-Credentials/Pwdb_top-1000.txt`

we use the Robert David Graham's tool heartleech [11] to execute the attack, using the `--autopwn` option to extract the private key out of the server.

3.2.3 Thursday

On Thursday, we implement the infiltration attack using Damn Vulnerable Web App (DVWA). As we lack access to the authors' specific implementation of this attack, we write our own Python scripts to execute the attacks based on the traffic. For the bruteforce attack, authors seem to use a random string of characters as a password. We opt to make the attack more realistic by using same wordlists as in Tuesday bruteforce attacks. For the SQL Injection attack, where a lack of input sanitation can expose the entire table, we use the solution provided by the DVWA. For the Cross-Site scripting attack, we inject a `console.log` statement into the website, similarly to the dataset authors. We package these scripts into a Docker image (`themessik/dwva_attacker`) and launch attacks against the official Docker image of DVWA `vulnerables/web-dvwa`.

We choose to omit the infiltration attacks of afternoon for two reasons. First, we are limited by the ConCap framework. ConCap does not (yet) support multistage attack execution, making it difficult to reproduce this attack faithfully. Second, and more important, there is little interesting traffic happening on the network in this attack: a file gets downloaded and/or executed and a portscan is run against the victim network. We will get samples of portscans on Friday, so essentially repeating the attack here here serves little benefit. Furthermore, the maliciousness of a file download is due to the file itself, not the inherent act of downloading a file - this kind of detection is better suited for antivirus software / systems that inspect the files, not network statistics.

3.2.4 Friday

On Friday morning, a botnet attack using Ares is executed. We choose to omit this attack in our reconstruction due to technical limitations: Ares does not provide a way to control the botnet from Linux hosts and is reliant on Windows for its execution. Due to this, we cannot package it into Docker containers and execute attacks with it on ConCap.

For Friday afternoon DDoS attack using LOIC, we use the multi-target functionality provided by ConCap. LOIC inherently relies on Windows binaries and its graphical user interface for its function. However, LOIC also provides a way to control it headlessly, through an IRC server. We set up two targets: one the actual webserver we want to attack, the other an IRC server. We have prepared a bot that listens on the IRC channel for LOIC and waits for it to connect. Upon connection, the channel topic is set up as a command for LOIC to attack the webserver. LOIC then proceeds to execute this attack as if it was controlled through the graphical user interface.

Finally, for portscans, we have analyzed the traffic and enumerated the open ports: 21, 22, 80, 139, 445. These open ports point to following services being online on the server: FTP, SSH, HTTP and Samba. We construct the target server such that all of these services are available and we execute the attack against this server. Samba has proven exceptionally difficult to get configured this way. We solve this issue by running the portscan against a dedicated Samba Docker image `dperson/samba`. This separate traffic later gets merged into other the other portscan traffic before NetFlows are generated.

3.3 Experiments

After reconstructing the dataset by writing dedicated Docker images and ConCap scenarios, we want to experimentally verify that the generated traffic is equivalent.

For the purposes of developing an ML-NIDS, we define the equivalence of original and synthetic network traffic as follows: a model trained on original traffic can predict synthetic traffic with high degree of accuracy, and vice-versa.

In their analysis and experiments with CIC-IDS-2017 [7], D'Hooge trains a single feature, single decision Decision Tree on the entire dataset using usual ML methodology and find ROC AUC scores above .9 for multiple different features. A high imbalance between attack classes and benign samples can be noted: attack traffic accounts for 19.6% of the total flows presented. In this thesis, we follow a methodology similar to theirs to prepare the dataset for testing:

- Generate NetFlows from the PCAP files
- Strip metadata columns and clean the dataset by dropping duplicate rows and rows containing NaN values

As class imbalance can lead to skewed results, we want to address it by synthetically augmenting the attack flows with benign traffic. To do this, we separately load and clean the original Monday PCAP file. We call these flows "benign traffic".

To verify the equivalence, we perform the same procedure for every attack class. Using Wireshark, we extract all packets related to the attack class from the original PCAP files. We treat it with the same cleaning and balancing procedure as the traffic generated by ConCap. We call this cleaned and balanced dataset "CIC traffic". We also prepare synthetic network traffic generated by ConCap in the exact same way and call this dataset "ConCap traffic".

Using CIC traffic, we train a Decision Tree model with the Gini criterion and a single node, using a single datapoint in the dataset as the decision data point, as done by [7]. After training, we calculate the model's accuracy of predicting the label of the ConCap traffic. We repeat this procedure in reverse, training on ConCap traffic and evaluating on CIC traffic.

Finally, we want to evaluate the generalization performance to traffic from another dataset. To this end, we use a more mature sibling of CIC-IDS-2017, the CIC-IDS-2018 dataset. Our procedure here is similar: for each attack class, we extract the relevant traffic, clean it and preprocess it. We call this processed dataset "CIC 18 traffic" We then train the a Decision Tree on the ConCap traffic and evaluate its accuracy on the CIC 18 traffic.

4 Results

The processed Netflow files can be found on Kaggle. We perform the verification experiment for each attack class separately.

To quantify the performance of the trained models for different features, we calculate following metrics:

- **Recall**: Recall expresses the correctness of model predictions on positive samples as a ratio of true positives with respect to all positive samples:

$$\frac{TP}{TP + FN}$$

- **Accuracy**: a metric expressing the correctness of predictions of a model as a ratio of the correct predictions and total predictions

$$\frac{TP + TN}{TP + TN + FN + FP}$$

- **Precision**: Precision describes how precise the model is in its positive predictions, calcualted as a ratio of true positives and all predicted positives:

$$\frac{TP}{TP + FP}$$

- **Area Under Receiver Operator Characteristic Curve (ROC AUC) score**: By calculating the True Positive Rate and False Positive Rate of the model at set intervals, a curve can be plot through them. Area under this curve expresses how the probability of the model predicting a random malicious sample as malicious.

In graphs below, we visualize these metrics for each attack class. Depending on the traffic the model was trained on (Either original CIC-IDS-2017 or ConCap scenarios), we find different sets of features that hold most predictive power. We do note that common features can be found in these sets, and suggest that these features are most representative of the underlying traffic. Consequently, it is these common features that should be used as decision features for actual ML-NIDS systems and we only visualize these features.

The presence and absence of particular features is a peculiar artifact. We speculate that this can be explained by the difference in the methodology of dataset creation. Our reasons for this are threefold.

First, though ConCap allows for network features configuration, from bandwidth to packet corruption rate; we have insufficient information about the network conditions of the CIC-IDS-2017 dataset. For this reason, we opt to use the example configuration provided by ConCap authors and keep these conditions the same for all scenarios.

Second, ConCap networks are built fully in-silico and thus do not suffer from the same artifacts as physical networks might: arbitrary delay in packet processing, faulty connections or interference, as well as processing delay. ConCap network architecture is, without introduction of artificial defects, the perfect network.

Third, the attacks in the CIC-IDS-2017 dataset take place in addition to benign, background traffic, leading to highly imbalanced datasets. We manually balance the training and testing datasets by including known benign flows to achieve class balance. While this should have little effect on features focusing on the size and content of packets (e.g. Packet Length or Flags features), it will have an effect on features that are more concerned with timings (e.g. Flow Bytes/s or Idle features), as the benign flows had no way of influencing attack flows during traffic capture.

5 Conclusion

ML-NIDS systems are the next frontier in NIDS research. Being based on machine learning methods, these systems are capable of learning to distinguish between malicious and benign traffic without the need to manually program them with expert knowledge. These systems do rely on high quality datasets, of which there were numerous attempts to construct; each plagued with different issues.

In this thesis, we have experimentally verified a new method of constructing ML-NIDS datasets through the ConCap framework by reconstructing the CIC-IDS-2017 dataset into ConCap scenarios and showing that the resulting dataset is representative of the original. This quality has been observed by the commonality of features that the models performed best on. Furthermore, we have further shown that the robustness of ML-NIDS models can be increased by augmenting their training with ConCap traffic, which improves their generalization performance to unseen samples from other datasets. This further cements ConCap (and other such frameworks) as the next step in ML-NIDS dataset generation and augmentation.

6 Future Work

The road does not end here. While we have verified that ConCap is capable of generating representative traffic, further research is required to build a fully functioning ML-NIDS system around it. Below, we sketch out a number of directions further research could go into.

First, ConCap itself is limited to fully connected networks and simple "run attack command" attacks. In order to be fully representative of our modern networks, support for further network segmentation, switches, firewalls, routers... is needed. In an ideal world, a system administrator should be able to reconstruct their network within ConCap, with all of the fine details that go into configuring a company network. This system administrator should then be able to deploy attacks against the network from a library, train an NIDS system and verify the protections it provides. Furthermore, multi-stage and event-based attacks should be supported, as currently, one cannot easily run a tailored attack based on the current network traffic or conditions.

Second avenue could be a real-time ML-NIDS. The ML model itself is but a small part at the heart of an ML-NIDS system. We have not spent any time studying how to provide the NetFlows to the machine learning model on the fly, instead we post-processed the PCAP files into NetFlows and trained a model on that. While useful, this method will only detect an attack after the fact; and after the fact might be too late, certainly for attacks such as Heartbleed, where leaking the private key spells disaster and could be non-trivial to remediate. A real-time ML-NIDS could detect an attack while it is taking place, enhancing the security of the network that much more.

Third, ConCap can be used not just for its generation capabilities, but also for simulation. This thesis has shown that ConCap is a sound framework for traffic generation, therefore, we propose that this generative capability need not only be used in the context of machine learning, but could also be used as a practical tool for blue team training. Currently, the ConCap environment is spun up, attacks are executed, traffic is captured and the environment is shut down. We envision an environment that remains running, with defenders having the opportunity to train and improve their responses to various attacks.

7 Extensions

Here we describe extensions to the attacks we have implemented as part of dataset reconstruction to show the versatility of ConCap as well as the ease of dataset augmentation.

7.1 Tuesday attacks

For Tuesday attacks, we construct the image in such a way that the wordlist used can be easily exchanged for another one. The image `themessik/patator` does come prepackaged with dictionaries, but these are neither exhaustive nor useful for modelling actual bruteforce attacks. Instead, the attack command can be extended with a flag to download the dictionaries to be used. Instead of specifying the path to the wordlist inside the image, specify a link to download the dictionaries. These will then be downloaded and utilized for attack execution.

We provide example scenario showing this functionality in `scenarios/extensions/tuesday_download_lists.yaml`.

8 References

[1] *A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018) - Registry of Open Data on AWS — registry.opendata.aws.* `https://registry.opendata.aws/cse-cic-ids2018`. [Accessed 03-05-2025].

[2] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. "Sok: Pragmatic assessment of machine learning for network intrusion detection". In: *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2023, pp. 592–614.

[3] Xiaohui Bao, Tianqi Xu, and Hui Hou. "Network intrusion detection based on support vector machine". In: *2009 International Conference on Management and Service Science*. Beijing, China: IEEE, Sept. 2009.

[4] Sumalya Chatterjee. *R3DHULK / HULK*. 2024. URL: `https://github.com/R3DHULK/HULK`.

[5] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". en. In: *Mach. Learn.* 20.3 (Sept. 1995), pp. 273–297.

[6] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: `10.1109/TIT.1967.1053964`.

[7] Laurens D'hooge. *Distrinet-CIC-IDS2017-01-EDA-OneR: 0.989 ROC-AUC*. 2025. URL: `https://www.kaggle.com/code/dhoogla/distrinet-cic-ids2017-01-eda-oner-0-989-roc-auc`.

[8] Gints Engelen, Vera Rimmer, and Wouter Joosen. "Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study". In: *2021 IEEE Security and Privacy Workshops (SPW)*. 2021, pp. 7–12. DOI: `10.1109/SPW53761.2021.00009`.

[9] Evelyn Fix and J. L. Hodges. "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties". In: *International Statistical Review / Revue Internationale de Statistique* 57.3 (1989), pp. 238–247. ISSN: 03067734, 17515823. URL: `http://www.jstor.org/stable/1403797` (visited on 05/02/2025).

[10] Robert Flood et al. "Bad Design Smells in Benchmark NIDS Datasets". In: *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. 2024, pp. 658–675. DOI: `10.1109/EuroSP60621.2024.00042`.

[11] Robert David Graham. *robertdavidgraham / heartleech*. 2014. URL: `https://github.com/robertdavidgraham/heartleech`.

[12] Gregory Gromov. *Roads and Crossroads of the Internet History*. Accessed: 30-04-2025 16:29. 1995. URL: `https://www.netvalley.com/cgi-bin/intval/net_history.pl?chapter=1`.

[13] *Heartbleed*. URL: `https://heartbleed.com/`.

[14] K A Jackson, D H DuBois, and C A Stallings. “NADIR (Network Anomaly Detection and Intrusion Reporter): A prototype network intrusion detection system”. In: Los Alamos National Lab., NM (USA). Jan. 1990. URL: https://www.osti.gov/biblio/6192985.

[15] Arash Habibi Lashkari. *CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is a network traffic Bi-flow generator and analyser for anomaly detection. https://github.com/ISCX/CICFlowMeter*. 2018.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. en. In: *Nature* 521.7553 (May 2015), pp. 436–444.

[17] Yihua Liao and V.Rao Vemuri. “Use of K-Nearest Neighbor classifier for intrusion detection11An earlier version of this paper is to appear in the Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, August 2002”. In: *Computers & Security* 21.5 (2002), pp. 439–448. ISSN: 0167-4048. DOI: https://doi.org/10.1016/S0167-4048(02)00514-X. URL: https://www.sciencedirect.com/science/article/pii/S016740480200514X.

[18] Daniel Miessler. *danielmiessler / SecLists*. 2025. URL: https://github.com/danielmiessler/SecLists.

[19] Nour Moustafa and Jill Slay. “UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)”. In: *2015 military communications and information systems conference (MilCIS)*. IEEE. 2015, pp. 1–6.

[20] Jorge Oliveira. *NewEraCracker / LOIC*. 2022. URL: https://github.com/NewEraCracker/LOIC.

[21] *OpenSSL Security Notice*. URL: https://openssl-library.org/news/secadv/20140407.txt.

[22] D Opitz and R Maclin. “Popular ensemble methods: An empirical study”. In: *J. Artif. Intell. Res.* 11 (Aug. 1999), pp. 169–198.

[23] Duc Minh Pham et al. “Network Intrusion Detection with CNNs: A Comparative Study of Deep Learning and Machine Learning Models”. In: *2024 2nd International Conference on Computer, Vision and Intelligent Technology (ICCVIT)*. 2024, pp. 1–6. DOI: 10.1109/ICCVIT63928.2024.10872423.

[24] Rapid7. *rapid7 / metasploit-framework*. 2025. URL: https://github.com/rapid7/metasploit-framework.

[25] Shailendra Sahu and Babu M Mehtre. “Network intrusion detection system using J48 Decision Tree”. In: *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE. 2015, pp. 2023–2026.

[26] Jan Seidl. *jseidl / goldeneye*. 2020. URL: https://github.com/jseidl/GoldenEye.

[27] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP*. INSTICC. SciTePress, 2018, pp. 108–116. ISBN: 978-989-758-282-0. DOI: `10.5220/0006639801080116`.

[28] Iman Sharafaldin et al. "Towards a Reliable Intrusion Detection Benchmark Dataset". In: *Software Networking* 2017 (Jan. 2017), pp. 177–200. DOI: `10.13052/jsn2445-9739.2017.009`.

[29] Sergey Shekyan. *shekyan / Slowhttptest*. 2024. URL: `https://github.com/shekyan/slowhttptest`.

[30] Ali Shiravi et al. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". en. In: *Comput. Secur.* 31.3 (May 2012), pp. 357–374.

[31] Youngrok Song, Sangwon Hyun, and Yun-Gyung Cheong. "Analysis of Autoencoders for Network Intrusion Detection". In: *Sensors* 21.13 (2021). ISSN: 1424-8220. DOI: `10.3390/s21134294`. URL: `https://www.mdpi.com/1424-8220/21/13/4294`.

[32] Miel Verkerken and Matisse Calleawert. *idlab-discover / RustiFlow*. 2025. URL: `https://github.com/idlab-discover/RustiFlow`.

[33] Barry de Ville. "Decision trees". en. In: *Wiley Interdiscip. Rev. Comput. Stat.* 5.6 (Nov. 2013), pp. 448–455.

[34] Jinming Zou, Yi Han, and Sung-Sau So. "Overview of artificial neural networks". In: *Artificial neural networks: methods and applications* (2009), pp. 14–22.

[35] Ghulam Mohi ud din. *NSL-KDD*. 2018. DOI: `10.21227/425a-3e55`. URL: `https://dx.doi.org/10.21227/425a-3e55`.

[36] lanjelot. `https://github.com/lanjelot/patator`.

[37] nmap. *nmap/ nmap*. 2025. URL: `https://github.com/nmap/nmap`.

[38] sweetsoftware. *sweetsoftware / Ares*. 2017. URL: `https://github.com/sweetsoftware/Ares`.

9 Appendices