Detecting Novel Application Layer Cybervariants using Supervised Learning

Etienne van de Bijl
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands
Email: evdb@cwi.nl

Jan Klein
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands
Email: jan_g_klein@outlook.com

Joris Pries
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands
Email: jorispries@gmail.com

Rob van der Mei
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands
Email: mei@cwi.nl

Sandjai Bhulai
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
Email: s.bhulai@vu.nl

***Abstract*—Cyberdefense mechanisms such as Network Intrusion Detection Systems predominantly use *signature-based* approaches to effectively detect known malicious activities in network traffic. Unfortunately, constructing a database with signatures is very time-consuming and this approach can only find previously seen variants. Machine learning algorithms are known to be effective software tools in detecting known or unrelated novel intrusions, but if they are also able to detect unseen variants has not been studied. In this research, we study to what extent binary classification models are accurately able to detect novel variants of *application layer* targeted cyberattacks. To be more precise, we focus on detecting two types of intrusion variants, namely *(Distributed) Denial-of-Service* and *Web* attacks, targeting the *Hypertext Transfer Protocol* of a web server. We mathematically describe how two selected datasets are adjusted in three different experimental setups and the results of the classification models deployed in these setups are benchmarked using the Dutch Draw baseline method. The contributions of this research are as follows: we provide a procedure to create intrusion detection datasets combining information from the transport, network, and application layer to be directly used for machine learning purposes. We show that specific variants are successfully detected by these classification models trained to distinguish benign interactions from those of another variant. Despite this result, we demonstrate that the performances of the selected classifiers are not symmetric: the test score of a classifier trained on A and tested on B is not necessarily similar to the score of a classifier trained on B and tested on A. At last, we show that increasing the number of different variants in the training set does not necessarily lead to a higher detection rate of unseen variants. Selecting the right combination of a machine learning model with a (small) set of known intrusions included in the training data can result in a higher novel intrusion detection rate.**

***Keywords*—Cybersecurity; network intrusion detection; anomaly detection; binary classification; open-world learning.**

I. Introduction

A partial and preliminary version of this paper was presented in [1]. In our increasingly digitized world, network security has become more challenging as the Internet is used for virtually all information operations, such as storage and retrieval. The rat race between attackers and defenders is perpetual as new tools and techniques are continuously developed to attack web servers containing this information. Tremendous problems for organizations and individuals arise when legitimate users cannot access data due to cyberattacks. Modern attacks are designed to mimic legitimate user behavior and target vulnerabilities in *application-layer* protocols, such as the *Hypertext Transfer Protocol* (HTTP). This mix makes detecting them a challenging and complex task.

Defenders often use an *Intrusion Detection System* (IDS) to perform the task of detecting intrusions. An IDS can be viewed as a burglar alarm in the cybersecurity field [2]. It monitors network traffic, aims to detect malicious activities, and an alarm is triggered when this is the case. Generally speaking, the two used methodology classes by these systems are *signature-based* and *anomaly-based* [3]. A signature-based detector compares observed network events against patterns that correspond to known threats.

In contrast, anomaly-based detectors search for malicious traffic by constructing a notion of normal behavior and flags activities that do not conform to this notion. Where signature-based is time-consuming but effective, anomaly-based often suffers from a high false-positive rate. Within anomaly detection methods, *Machine Learning* (ML) algorithms are getting more attention as they might overcome this problem.

The thought of using ML algorithms to detect intrusions is not new. Various studies are performed on using ML for detecting cyberattacks. Unfortunately, there is a striking imbalance between the extensive amount of research on ML-based anomaly detection techniques for intrusion detection and the rather clear lack of operational deployments [4]. ML algorithms are highly flexible and are adaptive methods to find patterns in big stacks of data [5], but they seem better at this task than discovering meaningful outliers [4]. Modern cyberattacks often occur in large quantities and thus do not entirely conform to the premise that patterns cannot be found for these outliers. Therefore, using ML for the task of detecting these attacks should be appropriate.

There appear two issues when looking at anomaly-based ML research in intrusion detection [6][7]. Firstly, the performance of most of these methods is measured on outdated datasets [8]. This makes it hard to estimate the performance of these methods on modern network traffic. A major issue is that the composition of benign and malicious traffic in these datasets does not represent modern real-time environments.

Also, there used to be a lack of representative publicly available intrusion detection datasets, but this lack was noticed by the cyberdefense community and recently more intrusion datasets have been generated [10]. Still, the available datasets are often limited to features extracted from the transport and network layer and lack application layer features. Thus, not all attainable features are extracted in these datasets. Secondly, it is not examined how supervised learning methods perform in detecting novel variants of known attacks. The performance of these methods is measured in either a *closed-world learning setting* in which training and test classes are the same or an *open-world learning setting* with unrelated attacks. However, it is not tested how the methods perform in an open-world setting with novel variants.

Similar to our previous work [1], the aim of this paper is to study to what extent ML models are accurately able to detect novel variants of known cyberattacks. To be more precise, we use supervised binary classifiers to learn from a dataset containing benign and application layer cyberattacks and we evaluate them on their ability to detect unseen variants of these attacks. In this work, we do not only focus on one cyberattack, the *Denial-of-Service* (DoS) or its distributed form (DDoS) but include a second variant: Web attacks. We examine how the selected classifiers perform when using a single cyberattack in the training dataset on this task. Afterward, we study the effect of combining malicious variants in the training phase on the performance of classifiers detecting unseen variants. The results of this binary classification problem are benchmarked using the Dutch Draw baseline method [9]. Furthermore, we provide a procedure to transform raw network traffic data into ML-usable datasets containing information from the network, transport, and application layer. The code of this procedure is publicly available [11].

The main contributions can be summarized as follows: Firstly, we show that ML classifiers are to a great extent able to detect known cyberattacks in a closed-world setting when presented with sufficient data. Secondly, we show that there are situations where these classifiers are able to detect a novel variant when they are trained to detect a different variant. However, this is not a two-way street: learning to detect attack A and being able to also detect attack B does not imply that the reverse is the case. Thirdly, we show that training on imbalanced data has an adverse effect on the evaluation performance of some ML classifiers. We have demonstrated that variants included in the CIC-IDS-2017 seem not identical to the same variants in the CIC-IDS-2018. Finally, we demonstrate that it is not necessary to use many variants to detect a novel attack. Sometimes a few known attacks can already lead to the highest detection rate.

The organization of this paper is as follows. Section II gives a literature overview regarding detecting novel intrusions with ML. Section III describes the selected datasets and how they are modified into ML-applicable datasets and states metadata about them. In addition, a set of ML models used for conducting the experiments are given. Section IV outlines the conducted experiments. Section V shows the results of the conducted experiments. Finally, we conclude and summarize in Section VI.

II. Related Work

Detection of novel attacks with supervised learning techniques has been studied before in the context of *Transfer Learning* (TL). TL is an ML paradigm where a model trained on one task is used as a starting point for another task. [12] introduces a feature-based TL approach to find novel cyberattacks by mapping source and target datasets in an optimized feature representation. This approach is however very dependent on a similarity parameter and the dimensions of the new feature space. Therefore, [13] extended this method by proposing another approach to automatically find a relationship between the novel and known attacks. Both of these approaches are tested on an outdated dataset and it does not contain variants of a single cyberattack. In our research, we are interested in the detection of novel variants rather than novel variants. In [14], a Convolutional Neural Network is used to detect novel attacks also in a TL setup, but it is not studied if learning one specific attack affects the detection of another novel variant. The experiments conducted in our research resemble the experiments performed in [15]. In their research, an intrusion detection method is introduced that transfers knowledge between networks by combining unrelated attacks to train on. More recent work focuses on applying Deep Neural Networks in the context of TL for intrusion detection tasks [16].

III. Data

We discuss the procedure to convert raw network traffic into usable intrusion detection datasets containing information from the network, transport, and application layer for ML purposes. The converted and extracted features are described in detail so it is clear which features are included. Furthermore, we provide metadata describing the final datasets. At last, the classification models and their set of considered hyperparameters are given for detecting novel variants.

A. Data Sources

A perfect intrusion detection dataset should at least be up-to-date, correctly labeled, publicly available, contain real network traffic with all kinds of attacks and normal user behavior, and span over a long time [10]. The main reasons for a lack of appropriate datasets satisfying these properties are (1) privacy concerns regarding recording real-world network traffic and (2) labeling being very time-consuming. However, synthetic or anonymized datasets have been generated that satisfy some of these ideal properties. It is therefore recommended to test methodologies on multiple datasets instead of only one [4]. In this research, we focus on the detection of malicious variants and for that reason, we have selected the CIC-IDS-2017 [17] and the CIC-IDS-2018 [18] datasets created by the Canadian Institute for Cybersecurity (CIC). These datasets are correctly labeled, publicly available, up-to-date, and contain several malicious cyberattacks.

B. Feature Extraction

The selected datasets are provided by the CIC in two formats: a set of raw network traffic (`pcap`) files and a set of files containing extracted features by a network analysis tool called CICFlowMeter [19]. These features mainly describe network and transport protocol activities. However, there are no features describing application activities. As this study focuses on detecting application layer cyberattacks, it is desirable to have a dataset also containing application layer features. Therefore, we start with the raw internet traffic format and have selected a feature extraction tool matching this requirement.

The feature extraction tool used in this study is the open-source network traffic analyzer called Zeek (formerly Bro) [20]. Zeek is a passive standalone IDS and derives an extensive set of logs describing network activity. These logs include an exhaustive record of all sessions seen on the wire. Zeek was also used as a feature extraction tool for the creation of other popular network intrusion detection datasets, e.g., DARPA98 [21] from the Defense Advanced Research Projects Agency (DARPA) and the UNSW-NB15 [22] from the University of New South Wales (UNSW). Zeek has a good track record in creating intrusion detection datasets and therefore an appropriate tool.

By default, Zeek generates a large set of log files, but not all of them are required for this research. We limit ourselves to the *Transmission Control Protocol* (TCP) entries given in the connection logs (`conn.log`), describing network and transport layer activity, and HTTP interactions given in HTTP logs (`http.log`). These log files include entries showing malicious activities. The entries in the connection log files are transport-layer sessions, while the HTTP log file consists of entry logs showing conversations between a client and a web server. Entries between these logs are unilaterally linked as each HTTP entry is assigned to a single connection entry. Malicious activities that are not (D)DoS or Web attacks are excluded as we only focus on these attacks.

C. Feature Engineering

We describe how the extracted features are converted into ML-admissible features. This section states the additional created features, which features are replaced for better extraction of patterns, and how we smartly one-hot-encode categorical features. We start with describing the feature engineering steps in the connection log file and afterward do the same for the HTTP log file.

a) Connection log: Zeek counts the number of packets and bytes transferred in each connection. Table I shows additional created features from these counters. A higher-level statistic called the *Producer-Consumer Ratio* (PCR) [23] shows the ratio between sending and receiving packets between the hosts. In a TCP connection, an originator host is an uploader if a PCR is close to 1.0 and purely a downloader if it is close to -1.0.

The feature `conn_state` constructed by Zeek refers to the final state of a TCP connection. This state is determined by registering flags exchanged during the communication between hosts. Looking only at the end of a connection implies that the establishment and termination of the connection are merged. Preliminary results showed that classifiers were better able to find patterns in (D)DoS traffic when differentiating between the establishment of a connection and the termination of it. On this note, we replaced the `conn_state` feature with features describing both ends of a connection. The *3-Way Handshake* is the correct way to establish a TCP connection before data is allowed to be sent. This procedure is however not always correctly executed and incorrect establishments can indicate misuse. Hosts can terminate TCP connections gracefully, or not. A graceful termination occurs when both hosts send a packet with a *final* (FIN) flag. When a host sends a packet containing a *reset* (RST) flag, it will abruptly end a TCP connection, which is very common in practice. If neither is the case, connections are in theory still open. In Table II, we distinguish different establishment and termination scenarios by looking at the exchanged flags between the hosts. Each of these scenarios is included in the data as a binary feature. Other Zeek connection log flags are one-hot-encoded for both the originator and responder.

TABLE I. NETWORK LAYER ENGINEERED FEATURES.

Feature	Description	Type
orig_bpp	$\frac{\text{orig\_bytes}}{\text{orig\_packets}}$	Float
resp_bpp	$\frac{\text{resp\_bytes}}{\text{resp\_packets}}$	Float
PCR	$\frac{\text{orig\_bytes}-\text{resp\_bytes}}{\text{orig\_bytes}+\text{resp\_bytes}}$	Float

TABLE II. TCP CONNECTION ESTABLISHMENT AND TERMINATION SCENARIOS.

Feature	Description
S0	No SYN packet is observed
S1	Merely a connection attempt (SYN), but no reply
REJ1	A connection attempt but replied with a RST packet
S2	A connection attempt followed by SYN-ACK, but no final ACK
REJ2O	Scenario S2 but originator sends RST packet
REJ2R	Scenario S2 but responder sends RST packet
S3	Connection is established according to the 3-way handshake
WEIRD	A connection attempt but none of the above cases were observed
OPEN	A connection was established, but no FIN or RST flag is observed
TERM	Connection gracefully terminated by originator and receive
CLSO	Originator sends a FIN flag but receiver did not respond
CLSR	Receiver sends a FIN flag but originator did not respond
RSTO	Originator abruptly ends connection by sending an RST flag
RSTR	Receiver abruptly ends connection by sending an RST flag

b) HTTP log: Communication in this protocol starts with a client sending a request message to a web server and this server will, hopefully, reply with a response message. Both message types consist of a start-line, zero or more header fields, an empty line indicating the end of the header fields, and possibly a message body. The start-line of a request message, called the request-line, contains three components: a method (command), a path to apply this command on, and an HTTP version indicating the version a client wants to use. Hosts must agree on the HTTP version to use before they continue talking.

If they did not agree on the HTTP version, a "−1" is imputed to distinguish it from other versions.

The feature `method`, showing the command given in the request message, is a feature showing the one-word command given in the request line. Commonly used commands are 'GET', 'HEAD', 'POST', 'PUT', 'DELETE', 'CONNECT', 'OPTIONS', 'TRACE', and 'PATCH', but other commands also exist. This categorical feature is one-hot-encoded to one of those common commands to limit the number of options. In case an uncommon command is given, it will be assigned to a feature called `method_other`, while if no command is given at all, it is assigned to `method_-`.

A web server applies a `method` on the Uniform Resource Identifier(`URI`) stated in a request line. This `URI` can be parsed in different components by a library called *urllib* [24]. Figure 1 gives an example of how this tool splits a Uniform Resource Locator (*URL*) into four components. We extracted descriptive statistics from each component by counting the number of special characters (not letters or digits), the number of characters, and the number of unique characters. A typical `URI` constitutes three components: a path, a query, and a fragment. Statistics are extracted for each of those components. For example, one extracted feature called `URI_path_len` describes the length of the path of a URI. In addition, Zeek extracts `host` (only netloc), the `referrer` (all components), and these descriptive statistics are also extracted for these features.

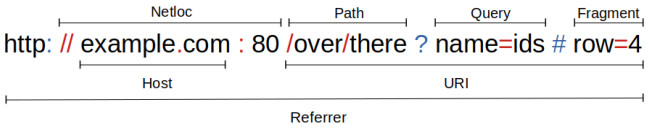

Figure 1. Example URL showing the four components parsed by urllib and the component coverage of extracted features by Zeek.

Web servers process received request messages and reply to them with a response message. In the status line of this message is the agreed HTTP version stated and a response code if the web server is able to process the request. The response codes are grouped by their first digit. So, for example, the error code 404 is assigned to the `4xx` code. Furthermore, it registered what type of data (e.g., application, audio, example, font, image, model, text, or video) is sent by the web server to the client or vice versa. This info is one-hot-encoded in a similar manner as the `method` for both directions.

D. Final Dataset

The log files are merged into a single dataset after feature engineering them. The resulting dataset consists of HTTP interactions, while in contrast, the datasets provided by the CIC consist of connection flows. Connection log features are added to the HTTP entry features to combine application, network, and transport layer features. This merge gives a dataset with a total of 103 features. The CIC-IDS-2017 consists of 533,845 instances and the CIC-IDS-2018 has 9,595,037 instances.

Table III shows the distribution of the labels of the entries. The benign/malicious ratio is roughly balanced for the CIC-IDS-2017, while it is more imbalanced for the CIC-IDS-2018. If we differentiate between cyberattacks, we observe that there is a clear imbalance between the malicious classes. For example, the *Hulk* (HTTP Unbearable Load King) attack generated a lot more HTTP entries in comparison to a *Slowloris* or *GoldenEye*. The same can be observed for Web attacks. The *Brute Force* and *XSS* web attacks are more occurring in the dataset than the *SQL injection* attack.

TABLE III. CLASS DISTRIBUTION OVER THE HTTP ENTRIES.

		CIC-IDS-2017		CIC-IDS-2018	
Class	**Type**	**Amount**	**Percentage**	**Amount**	**Percentage**
Botnet	DDoS	736	00.28%	142,925	04.28%
GoldenEye	DoS	7,908	02.97%	27,345	00.82%
HOIC	DDoS	0	00.00%	1,074,379	32.15%
Hulk	DoS	158,513	59.48%	1,803,160	53.95%
LOIC	DDoS	95,683	35.90%	289,328	08.65%
SlowHTTPTest	DoS	1,416	00.53%	0	00.00%
Slowloris	DoS	2,245	00.84%	4,950	00.15%
		266,501	100.00%	3,342,807	100.00%
Brute Force	Web	7,311	79.93%	13,144	54.02%
SQL Injection	Web	12	00.13%	57	00.23%
XSS	Web	1,824	19.94%	11,134	45.75%
		9,147	100.00%	24,335	100.00%
Benign	-	258,197	48.37%	6,252,950	65.00%
Malicious	-	275,648	51.63%	3,366,422	35.00%
		533,845	100.00%	9,619,372	100.00%

E. Models

Four ML algorithms are selected for our classification problem: Decision Tree (DT), Random Forest (RF), K-Nearest Neighbors (KNN), and Gaussian Naive Bayes (GNB). A grid search approach is performed to find the optimal hyperparameters for these algorithms. Table IV shows the considered parameters for each model. The optimal set of parameters for each model is used on the test dataset by selecting the highest F_1 score achieved on a validation set. As there was a limited amount of computational time, the hyperparameter space of computationally expensive models like KNN is smaller than simpler models like DT.

TABLE IV. HYPERPARAMETERS OPTIONS FOR THE SELECTED CLASSIFIERS.

Model	**Scikit Parameter**	**Options**
GNB	var_smoothing	1e-200
DT	criterion	[Gini , Entropy]
	splitter	[Best, Random]
	class_weight	[None, Balanced]
	max_features	[Auto, None, Sqrt, log2]
RF	criterion	[Gini, Entropy]
	class_weight	[None, Balanced]
	max_features	Auto
	n_estimators	[10, 50, 100, 250]
KNN	n_neighbors	5
	algorithm	[Ball Tree, KD Tree]

IV. EXPERIMENTAL SETUP

In this section, we elaborate on the conducted experiments. Three different experimental setups were created in which we tested ML models to detect cyberattacks. Before we elaborate

in detail on those three setups, we start with mathematical preliminaries, how it is split into the train, validation, and test sets, and how we turned each experiment into a binary classification problem. After describing the experiments we will discuss how the models are evaluated and tested against a benchmark method named the Dutch Draw.

A. Preliminaries

Suppose we have an intrusion detection dataset $\mathbb{X}$ consisting of M instances and K feature values each. Without loss of generality, we assume $\mathbb{X} \in \mathbb{R}^{M \times K}$. Say $S \coloneqq \{1, 2, \ldots, M\}$ are the indices of the instances. We assume that each of those instances is labeled. Each instance s has a corresponding label y_s. Let $y \coloneqq \{y_1, y_2, \ldots y_M\}$ be the vector containing all labels. The datasets consist of *Benign* traffic and a set of malicious cyberattack variants. Therefore, let $C \coloneqq \{n, p_1, p_2, \ldots p_L\}$ be the set of possible values each instance y_i could have as label. Here, label n is the *Benign* label, and $\{p_1, p_2 \ldots, p_L\}$ is the set of different cyberattack labels included in our data. Let $B \coloneqq \{s \in S | y_s = n\}$ be the set of instances that are of the *Benign* class and let $P_k \coloneqq \{s \in S | y_s = p_k\}$ indicate the instances that have malicious class p_k as a label.

B. Train-Test split

It is common practice in ML to split a dataset into two non-overlapping sets: a training set and a test set. Binary classification models use the training set to learn a relationship between the response variables and the explanatory variable. Let us denote S_{train} as the instances that are assigned to the training dataset and S_{test} as the instances that are assigned to the test dataset. It should hold that $S_{train}, S_{test} \subset S$ with the property that $S_{train} \cap S_{test} = \emptyset$, $|S_{train}| + |S_{test}| = M$ and we should select a ratio R such that $|S_{train}| \cdot R = |S_{test}|$. Typically, R is selected in such a way that there is an 80:20 train-test ratio. As we investigate stochastic and deterministic prediction models, the train and test procedure is repeated multiple times to get a proper average performance for these models.

There are multiple classes in the dataset, so we have added another requirement for the train-test split: we want to have a similar class distribution in both the training dataset as well as the test dataset. This is also known as *stratified sampling*. The train-test split of the instances should match the class distribution of the original dataset as closely as possible:

$$\frac{|S_{train} \cap P_k|}{|S_{train}|} \approx \frac{|S_{test} \cap P_k|}{|S_{test}|} \quad \forall k \in \{1, 2, \ldots, L\}.$$

Furthermore, we enforce that there are at least two observations of each class selected to make hyperparameter tuning possible.

C. Hyperparameter tuning

In Section III-E, we discussed the considered hyperparameters for the selected ML classifiers. Selecting the right hyperparameters is vital for good performance. Hyperparameters are compared by taking the average performance of the ML models tested on different validation datasets. Train-validation splits are created in the same manner as the train-test split. The hyperparameters yielding the highest F_1 score are selected to test the models on the testing dataset.

D. Setups

The classification problem at hand could be a multiclass classification problem when $L > 1$. We will, however, treat each problem as a binary classification problem by mapping all malicious classes towards a single label $p_{Malicous}$ when y is presented to the model to train on and when evaluating. The classifiers are tested on these datasets in three different experimental setups:

1) Detecting Known Attacks: Firstly, we study to what extent the selected classifiers are able to detect known attacks in a closed-world learning setting. The achieved detection rate by the different ML models could indicate an upper bound to the novel detection rate of the corresponding malicious class. To test this, the training data and the evaluation data contain the same two classes: *Benign* and one malicious variant. More specifically: the training dataset for this first experiment $T_{1,k}$ consists of instances given to a model:

$$T_{1,k} = (B \cup P_k) \cap S_{train}.$$

And we evaluate the performance of the models on the following test set $E_{1,k}$ on which we evaluate:

$$E_{1,k} = (B \cup P_k) \cap S_{test}.$$

For example, we let a model train to distinguish *Benign* from *Hulk* and test on the same two classes.

2) Detecting Novel Variants: Secondly, we examine to what degree classifiers are able to detect a novel variant when the training dataset only contains *Benign* traffic and one different variant. For example, we let a classifier train on distinguishing *Benign* from *Hulk* entries and evaluate the trained model on a test set containing *Benign* and *LOIC* (Low Orbit Ion Cannon). Both the *Hulk* and *LOIC* labels are converted to one Malicious label, to keep the binary classification setting. This experiment shows us how similar the novel test attack is to the known training attack. Let us define $T_{2,i}$ as the training instances:

$$T_{2,i} = (B \cup P_i) \cap S_{train}.$$

In contrast to the previous experiment, the included malicious instances are not from the same class and the test dataset E consists of instances:

$$E_{2,j} = (B \cup P_j) \cap S_{test}.$$

When $i = j$ holds, we simply get the same results as experimental setup 1.

3) Class Importance to Detect Novel Variants: Finally, we study what we call *class importance*: how crucial is including a variant in the training dataset on the novel cyberattack detection performance? Does learning on a combination of multiple attacks help identify novel variants? We look at combinations of cyberattacks in the training set and test the

trained model on detecting a novel attack. For example, we train on *Benign* and a combination of attacks such as *LOIC* and *Hulk* entries and test on a dataset containing *Benign* and a different novel attack such as *SlowHTTPTest*. More formally, let us first select an evaluation dataset $E_{3,j}$ containing variant p_j:

$$E_{3,j} = (B \cup P_j) \cap S_{test}.$$

Now, we want to find which set of malicious variants leads to the highest novel cyberattack (p_j) detection rate. Say $\mathcal{P}(S)$ is the powerset of set S. By definition, the powerset of a set is the set of all subsets. So, the considered cyberattack combinations in the training dataset for novel attack p_j are derived as follows:

$$C_j \coloneqq \mathcal{P}(C \setminus \{n, p_j\}) \setminus \{\emptyset\}$$

The empty set is excluded from this powerset as there needs at least one cyberattack to be included in the training dataset. Suppose we take now a random cyberattack set $c \in C_j$. Let us define the set of instances having a label included in this set c as $\mathcal{C} \coloneqq \{s \in S | y_s \in c\}$. Then the training dataset of this third experiment $T_{3,c}$ is:

$$T_{3,c} = (B \cup \mathcal{C}) \cap S_{train}.$$

With this formulation, we can study which set of known variants $c \in C_j$ leads to the highest novel cyberattack p_j detection rate.

E. Evaluation Metrics

In our classification task, the *positive* class represents malicious instances while the *negative* class represents benign entries. Let us denote $y_s \in \{0, 1\}$ as the actual label of an instance s where 0 is the negative class and 1 represents the positive class. A confusion matrix is constructed by comparing the binary predictions $\hat{y}$ of a classifier with the actual labels y. This 2×2 dimensional matrix contains four base measures: the number of *true positives* (TP), the number of *true negatives* (TN), the number of *false positives* (FP), and the number of *false negatives* (FN). TP and TN show the number of instances that are correctly predicted, while FP and FN two show the number of mistakes. These four measures form the basis of any binary evaluation metric.

The performance of a binary classification model is quantified by one or more evaluation metrics, which are a function of one or more base measures. The considered evaluation metric to test the selected classifiers is the F_1 score, which is the harmonic mean between *recall* and *precision*. Recall is the ratio of intrusions the classifiers were successfully able to detect, while precision is the ratio between the *true positives* and the number of *positively predicted* instances. Hence:

$$Recall = \frac{TP}{TP + FN}, \quad Precision = \frac{TP}{TP + FP}.$$

Taking the *harmonic* mean between those two gives the F_1 score, which is defined as:

$$F_1 = \frac{2}{Recall^{-1} + Precision^{-1}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

For cyberattacks aiming to exhaust a resource, it is better to have a low false alarm rate than a high recall as it is not necessary to block all malicious traffic. We simply want to prevent the resource from being overloaded and prevent blocking legit HTTP requests. This makes the task at hand different in contrast to detecting intrusions in general as there the cost of a false negative is higher. Still, optimizing only precision is not desirable. Therefore, the F_1 score is an appropriate middle ground as it optimizes the harmonic mean of those metrics. When data is imbalanced, this score is more suitable than accuracy as it corrects this imbalance.

F. Dutch Draw Baseline

Stating the evaluation metric scores of the selected classifiers makes it possible to compare their performances. It does, however, not indicate what the scores themselves mean without some frame of reference. Baselines help interpret results as models can only be considered appropriate when outperforming them. Therefore, we have selected the Dutch Draw (DD) [9] as this baseline method helps in comparing the performances of the selected ML models. This method gives as a baseline value the score of the optimal random classifier that is input-independent. The selected evaluation metric to compare classifiers is the F_1 score and it follows from [9] that the corresponding *DD baseline* is given by:

$$\frac{2P}{2P + M}.$$

To construct a baseline for a test dataset E, the number of positives is given by $P = |E \setminus (B \cap S_{test})|$ and $M = |S_{test}|$.

V. Experimental Results

Now, we show the results of the three experimental setups performed in this research. The results of the experiments were gathered by testing the selected classifiers on 20 different train-test splits for the CIC-IDS-2017 and 10 different for the CIC-IDS-2018. Furthermore, as the CIC-IDS-2018 is very large and there was limited computational time, a subset of the data was used for hyperparameter tuning. For the DT and RF techniques, 10% was randomly selected for hyperparameter tuning. As the KNN model, with the selected hyperparameter options, is computationally very expensive, we were limited to only using 1% (randomly) of the training data for hyperparameter tuning. The same percentage of data was required in the training process to be able to evaluate this model in a reasonable time. For the CIC-IDS-2017, no subset sampling was required for training purposes. In our experiments, we have performed multiple hold-out-cross validation splits with each split an 80/20 split in a random manner. Before splitting the data, all redundant features (features with only 0 values) are removed as these features do not contain any new information. In the training set, a validation set (20%) is randomly selected to obtain the best hyperparameters for each model.

A. Detecting Known Attacks

ML classifiers were tested on whether they were able to distinguish benign HTTP interactions from interactions that were labeled with a predefined known cyberattack by training and testing on the same classes. We start with discussing the results for the Web attacks and afterward show the results for (D)DoS attacks. Table V shows the average F_1 scores for the selected ML models and the corresponding standard deviations. Here, each row corresponds to the selected cyberattack for the setup (P_k). The relatively lowest scores are highlighted in red. It can be observed that almost in all scenarios the classifiers outperform the Dutch Draw baseline, for which we have taken the average expectation over all train-test splits. For the CIC-IDS-2018, we observe that the GNB and KNN model outperform the Dutch Draw baseline, but the scores were relatively low. Here, the models were not able to detect any of the *SQL injection* instances in all train-test splits. All other setups indicate that the ML models were able to find patterns to distinguish normal traffic from a malicious variant.

TABLE V. EXPERIMENT 1 F_1 SCORES OF CLASSIFIERS DETECTING KNOWN WEB ATTACKS.

CIC-IDS-2017	**DD**	**GNB**		**DT**		**RF**		**KNN**	
Attack	**Exp**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
Brute Force	0.0536	0.5711	0.0034	0.9994	0.0008	0.9994	0.0003	0.9983	0.0005
SQL Injection	0.0001	0.9500	0.2236	0.8419	0.2690	0.8833	0.2484	0.0833	0.2059
XSS	0.0139	0.9044	0.0065	0.9975	0.0033	0.9950	0.0022	0.9901	0.0043
CIC-IDS-2018									
Brute Force	0.0042	0.8108	0.0059	0.9994	0.0003	0.9996	0.0002	0.9861	0.0018
SQL Injection	0.0000	0.0134	0.0006	0.8834	0.0591	0.8655	0.0834	0.0000	0.0000
XSS	0.0035	0.9977	0.0009	0.9998	0.0003	0.9999	0.0002	0.9927	0.0027

The same analysis is performed for (D)DoS attacks. Table VI shows the average F_1 scores if classifiers were tested on the task of detecting known (D)DoS attacks. It can be observed that in almost all scenarios the considered models were able to learn the relevant characteristics of the considered attacks. One exception is the GNB model that was trained and tested with the on the *SlowHTTPTest* attack. This model obtained a high recall (0.997), but a poor score on its precision (0.154). Even though the model is able to detect most malicious instances, there were many false positives.

TABLE VI. EXPERIMENT 1 F_1 SCORES OF CLASSIFIERS DETECTING KNOWN (D)DOS ATTACKS.

CIC-IDS-2017	**DD**	**GNB**		**DT**		**RF**		**KNN**	
Attack	**Exp**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
Botnet	0.0057	1.0000	0.0000	0.9971	0.0046	0.9998	0.0008	0.9909	0.0076
GoldenEye	0.0577	0.9972	0.0010	0.9997	0.0002	1.0000	0.0000	0.9983	0.0006
Hulk	0.5511	0.9990	0.0002	0.9999	0.0000	1.0000	0.0000	0.9999	0.0000
LOIC	0.4257	0.9999	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000
SlowHTTPTest	0.0108	0.2339	0.2065	0.9955	0.0042	0.9956	0.0031	0.9874	0.0046
Slowloris	0.0171	0.9013	0.0078	0.9976	0.0016	0.9969	0.0023	0.9929	0.0035
CIC-IDS-2018									
Botnet	0.0437	0.9998	0.0001	1.0000	0.0000	1.0000	0.0000	0.9974	0.0011
GoldenEye	0.0087	0.9919	0.0006	0.9843	0.0010	0.9914	0.0004	0.9536	0.0051
HOIC	0.2558	0.9964	0.0001	0.9964	0.0001	0.9964	0.0001	0.9961	0.0002
Hulk	0.3658	0.9999	0.0000	1.0000	0.0000	1.0000	0.0000	0.9997	0.0000
LOIC	0.0847	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000
Slowloris	0.0016	0.9876	0.0018	0.9982	0.0012	0.9986	0.0007	0.9586	0.0054

B. Detecting Novel Attacks with One Attack Learned

Let us relax the closed-world assumption: What if our trained algorithm sees a different variant of the learned attack? Figure 2 shows the average F_1 scores achieved by the classifiers in this experiment when detecting Web attacks. The diagonal of this matrix shows the F_1 scores of the closed-world assumption, also obtainable from Table V, while the off-diagonal values were the scores of detecting novel attacks. We can observe that the GNB model is not able to detect all *Brute Force* attacks. Surprisingly, a high score is obtained when this model is not trained on the *Brute Force* attack but on the *XSS* attack.

Figure 2. Experiment 2 average F_1 scores for the CIC-IDS-2017 to detect known and novel Web attacks.

If we now look at the results extracted from the CIC-IDS-2018 dataset, we see a difference in comparison to the CIC-IDS-2017. Figure 3 shows that the DT model is very useful to detect a *Brute Force* attack when trained on the *SQL injection* and is also able to detect the *XSS* when using the *Brute Force* attacks. The scores here were actually higher on the diagonal for the CIC-IDS-2018 than the CIC-IDS-2017.

Figure 4 shows the average F_1 scores achieved by the classifiers when detecting novel and known (D)DoS attacks. The diagonal of this matrix shows again the F_1 scores of the closed-world assumption, also obtainable from Table VI, while the off-diagonal values were the scores of detecting novel attacks. We observe in this open-world learning setting that *Botnet* attacks were hard to detect in this setting, and neither can they easily be used to detect other variants. However, there were situations where classifiers were able to detect novel variants. This is, however, not symmetrical: learning attack

Figure 3. Experiment 2 average F_1 scores for the CIC-IDS-2018 to detect known and novel Web attacks.

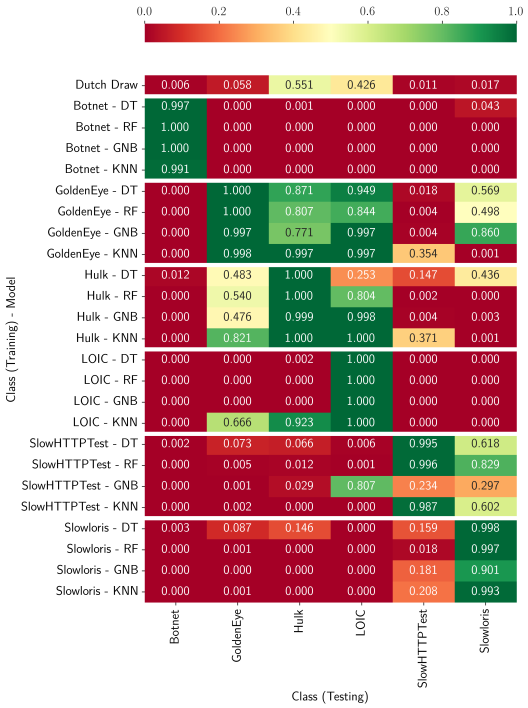

Figure 4. Experiment 2 F_1 scores averages for the CIC-IDS-2017 dataset to detect known and novel (D)DoS attacks.

A and finding attack B does not mean it works also the other way around.

Let us now look at the results of the CIC-IDS-2018 dataset containing *Benign* instances and (D)DoS attacks. Figure 5 shows the results of the same experimental setup performed on the CIC-IDS-2018. Similar results were observable on the diagonal: ML algorithms were indeed able to detect attacks it has trained on. In these results, it is less apparent that learning one (D)DoS attack leads to the model being able to detect another attack. Only a few combinations of train and test attacks were successful. For example, learning the *HOIC* (High Orbit Ion Cannon) with the KNN model results in high scores for testing on the *LOIC* and the *Hulk*. Results showed that classifiers such as DT and RF were not able to learn sufficiently from the training data as a striking class imbalance between benign and the attack led to low performance. Still, the same observation as in the CIC-IDS-2017 is apparent: when training on attack A and being able to detect B, it does not imply the reverse also holds.

Despite the 2017 and 2018 datasets having the same cyber-attacks, the HTTP interactions of the attacks are not identical. Figure 6 shows the results of the selected ML classifiers trained to detect a Web cybervariant of the CIC-IDS-2017 and tested whether the classifiers were able to detect CIC-IDS-2018 Web variants. We observe that there is no clear consistency between the ML models and whether they are able to detect known or novel attacks. Again, we see there are no symmetric results observable in the heatmap. It is not conclusive that the Web attacks in the CIC-IDS-2017 are identical to the CIC-IDS-2018.

Figure 7 shows the results of the selected ML classifiers trained to detect a (D)DoS variant of the CIC-IDS-2017 and tested whether the classifiers were able to detect CIC-IDS-2018 (D)DoS variants. It can be observed that in almost no situations the classifiers were able to do so. An exception here is the *Slowloris* attack, which has actually a high performance on all models except the GNB. This indicates that despite the datasets containing the same attacks, HTTP interactions were not necessarily identical.

C. Learning on a Set of Variants to Detect a Novel Variant

In our last experiment, we study which combination of cyberattacks in the learning phase results in the highest novel detection rate. Table VII shows the results when classifiers were trained on one or more Web variants to detect a novel Web variant. In bold is indicated the highest score obtained by the models and in the last column, the set of attacks that resulted in the bold score is stated. We observe that, for both datasets, the KNN is practically useless to detect novel Web variants. The *Brute Force* attack is always used in the training dataset to achieve the highest novel detection rate.

The same procedure and analysis were performed for the (D)DoS variant. Table VIII shows the results of the classifiers using a set of attacks to learn from and the corresponding combination of attacks that led to the highest performance. Despite the fact that models can use more attacks to detect

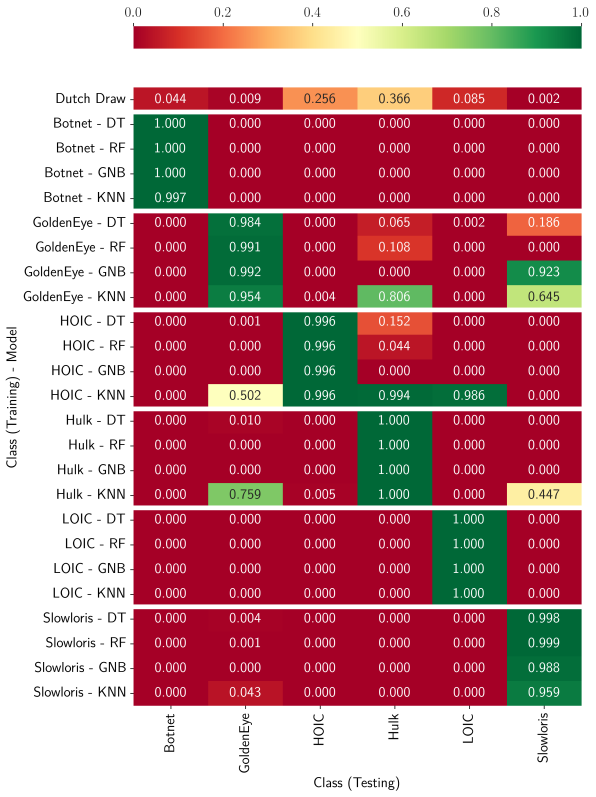

Figure 5. Experiment 2 F_1 scores averages for the CIC-IDS-2018 dataset to detect known and novel (D)DoS attacks.

Figure 6. Experiment 2 F_1 average classifier scores when trained on the CIC-IDS-2017 Web attacks and tested whether they were able to detect CIC-IDS-2018 Web attacks.

TABLE VII. EXPERIMENT 3 HIGHEST OBTAINED F_1 SCORE FOR EACH MODEL BY TRAINING THEM ON MULTIPLE INTRUSIONS TO DETECT A NOVEL WEB ATTACK.

CIC-IDS-2017	**DD**	**DT**	**GNB**	**KNN**	**RF**	**Train Set Opt Model**
Brute Force	0.054	0.202	**0.967**	0.000	0.100	{XSS}
SQL Injection	0.000	0.257	**0.400**	0.000	0.000	{Brute Force, XSS}
XSS	0.014	0.144	0.140	0.000	**0.327**	{Brute Force}
CIC-IDS-2018						
Brute Force	0.004	0.600	**0.767**	0.000	0.152	{SQL Injection, XSS}
SQL Injection	0.000	**0.366**	0.236	0.000	0.318	{Brute Force}
XSS	0.004	0.438	**0.735**	0.000	0.200	{Brute Force, SQL}

TABLE VIII. EXPERIMENT 3 HIGHEST OBTAINED F_1 SCORE FOR EACH MODEL BY TRAINING THEM ON MULTIPLE INTRUSIONS TO DETECT A NOVEL (D)DOS ATTACK.

CIC-IDS-2017	**DD**	**DT**	**GNB**	**KNN**	**RF**	**Train Set Opt Model**
Botnet	0.006	**0.460**	0.291	0.000	0.000	{Hulk, LOIC, Slowloris}
GoldenEye	0.058	0.664	0.476	**0.821**	0.782	{Hulk}
Hulk	0.551	0.870	0.986	**0.997**	0.833	{GoldenEye, LOIC}
LOIC	0.426	0.949	0.998	**0.999**	0.999	{Hulk}
SlowHTTPTest	0.011	0.240	0.181	**0.399**	0.100	{Hulk, Slowloris}
Slowloris	0.017	**0.878**	0.860	0.601	0.874	{Bot, Eye, Hulk, HTTP}
CIC-IDS-2018						
Botnet	0.044	0.000	0.000	0.000	0.000	-
GoldenEye	0.009	0.290	**0.862**	0.773	0.100	{LOIC, Hulk, Slowloris}
HOIC	0.256	0.000	**0.853**	0.500	0.000	{LOIC, Hulk}
Hulk	0.366	0.899	**0.999**	0.997	0.986	{GoldenEye, Slowloris}
LOIC	0.085	0.100	0.288	**0.985**	0.000	{HOIC}
Slowloris	0.002	0.539	**0.922**	0.837	0.000	{GoldenEye}

a novel variant, it is not necessarily the case that this yields the highest detection rate: even a few cyberattack classes were enough to obtain the highest performance. It can be observed that for the CIC-IDS-2017 the KNN model is dominantly getting the highest average F_1 scores, while for the CIC-IDS-2018 it is the GNB model. In neither case does the RF model outperform other models, which is unexpected as this model outperforms other models in detecting known attacks. For the CIC-IDS-2017 dataset, the *Hulk* attack is almost always used to obtain the highest scores with the least number of attacks required. The strong imbalance affects the learning process of the DT and the RF, similar to the results in experiment 2. These models could have been improved by downsampling benign entries so that the training classes were balanced.

VI. CONCLUSION

This research provides a procedure to construct intrusion detection datasets combining multiple layers with the tool Zeek. Zeek generates a set of extensive log files and two of them are selected to create an ML-admissible dataset for the detection of cyberattacks. This procedure to create such a dataset is not limited to only these protocols but can be extended to also combine other protocols, such as TCP connection with *File Transfer Protocol* interactions.

The aim of this research was to test to what extent ML

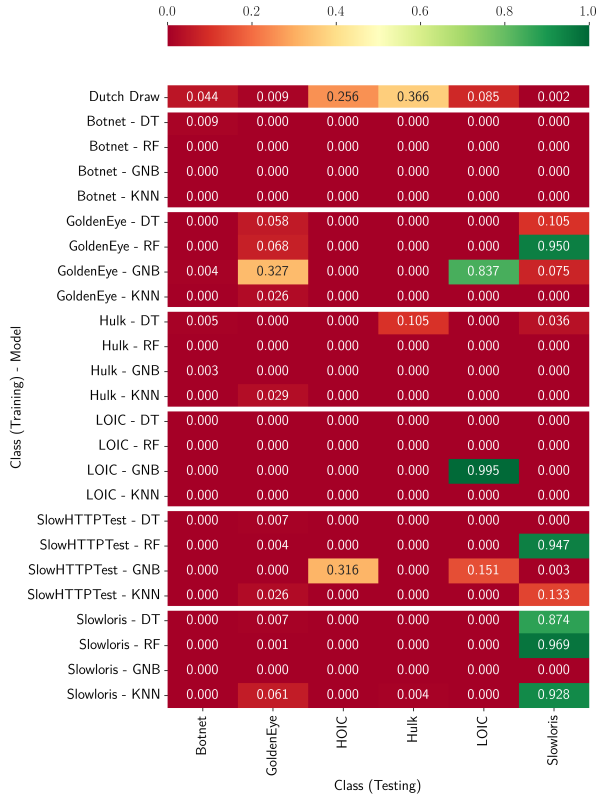

Figure 7. Experiment 2 F_1 average classifier scores when trained on the CIC-IDS-2017 (D)DoS attacks and tested whether they were able to detect CIC-IDS-2018 (D)DoS attacks.

classifiers are able to detect novel variants of known intrusions. A set of classifiers were applied in three different experimental setups and we studied their ability to detect variants. The focus of this research was to study the detection of variants of (D)DoS and Web attacks, but the same analysis can be performed on variants of another cyberattack. It has been shown in the first experiment that ML classifiers are to a great extent able to detect known (D)DoS attacks in a closed-world setting. For the Web attacks, the classifiers were not in all situations able to distinguish benign from malicious variants. Especially detecting *SQL Injection* instances with a GNB or KNN model was not accurate.

In the second experiment, it was observed that there are scenarios in which classifiers are able to detect a novel variant when trained on a different variant. Detecting novel variants is however not a two-way street: learning to detect attack A and being able to also detect attack B does not have the property that it is symmetrical. We have observed that for the CIC-IDS-2017 dataset the classifiers had a higher novel detection rate for (D)DoS variants than the results achieved on the CIC-IDS-2018. This was remarkable as the CIC-IDS-2018 contained similar attacks and more instances. It has been shown that the attacks are, however, not identical between the two datasets. Only the Slowloris seemed to have similar results between the datasets.

The third experiment showed that it is not necessary to use many malicious variants to detect a novel attack. Sometimes a few known attacks can already lead to the highest detection rate. Looking at the results of the (D)DoS attacks, DT and RF perform poorly in detecting novel attacks. The high imbalance in the training data caused this effect. The GNB model seemed more robust against this high imbalance in the training dataset and still achieved reasonable detection rates. For Web attacks, the results varied much between the model and cyberattack variant combination. The KNN model turned out to be effective in detecting known *Brute Force* and *XSS* attacks but was useless to detect novel Web attacks.

To sum up, this research shows that ML algorithms can, when sufficient training data is presented, detect cyberattacks almost as well as signature-based approaches, but also have the capability to detect novel variants. Selecting the right combination of an ML model with a (small) set of intrusion classes included in the training data can result in a higher novel intrusion detection rate.

References

[1] E. van de Bijl, J. Klein, J. Pries, R. D. van der Mei, and S. Bhulai, "Detecting novel variants of application layer (D)DoS attacks using supervised learning," in *IARIA Congress 2022: The 2022 IARIA Annual Congress on Frontiers in Science, Technology, Services, and Applications*, pp. 25–31, 2022.

[2] S. Axelsson, *Intrusion detection systems: A survey and taxonomy*, 2000, unpublished, http://www.cse.msu.edu/~cse960/Papers/security/axelsson00intrusion.pdf, retrieved: December, 2022.

[3] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[4] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 305–316.

[5] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1–2, pp. 18–28, 2009.

[6] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[7] Y. Xin et al., "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[8] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, pp. 1–29, 2020.

[9] E. van de Bijl, J. Klein, J. Pries, S. Bhulai, M. Hoogendoorn, and R. D. van der Mei, "The dutch draw: constructing a universal baseline for binary prediction models," 2022, arXiv:2203.13084.

[10] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, pp. 147–167, 2019.

[11] *NIDS*, December. 2022. [Online]. Available: https://github.com/etiennevandebijl/NIDS

[12] J. Zhao, S. Shetty, and J. W. Pan, "Feature-based transfer learning for network security," in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 2017, pp. 17–22.

[13] J. Zhao, S. Shetty, J. W. Pan, C. Kamhoua, and K. Kwiat, "Transfer learning for detecting unknown network attacks," *EURASIP Journal on Information Security*, vol. 2019, no. 1, pp. 1–13, 2019.

[14] P. Wu, H. Guo, and R. Buckland, "A transfer learning approach for network intrusion detection," in *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*. IEEE, 2019, pp. 281–285.

[15] Z. Taghiyarrenani, A. Fanian, E. Mahdavi, A. Mirzaei, and H. Farsi, "Transfer learning based intrusion detection," in *2018 8th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2018, pp. 92–97.

[16] M. Masum and H. Shahriar, "TL-NID: Deep neural network with transfer learning for network intrusion detection," in *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2020, pp. 1–7.

[17] I. Sharafaldin, A. H. Lashkari, and A. 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 2018)*. SCITEPRESS, 2018, pp. 108–116.

[18] *A realistic cyber defense dataset,* Canadian Institute for Cybersecurity, December. 2022. [Online]. Available: https://registry.opendata.aws/cse-cic-ids2018

[19] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*. SCITEPRESS, 2017, pp. 253–262.

[20] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[21] M. Bijone, "A survey on secure network: Intrusion detection & prevention approaches," *American Journal of Information Systems*, vol. 4, no. 3, pp. 69–88, 2016.

[22] N. Moustafa and J. 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.

[23] J. Klein, S. Bhulai, M. Hoogendoorn, R. Van Der Mei, and R. Hinfelaar, "Detecting network intrusion beyond 1999: Applying machine learning techniques to a partially labeled cybersecurity dataset," in *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 2018, pp. 784–787.

[24] *urllib* (3.9), December. 2022. [Online]. Available: https://docs.python.org/3/library/urllib.html