# Detecting Novel Variants of Application Layer (D)DoS Attacks 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: j.g.klein@cwi.nl

Joris Pries
*Centrum Wiskunde & Informatica*
Amsterdam, the Netherlands
Email: joris.pries@cwi.nl

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*—Denial of Service (DoS) attacks and their distributed variant (DDoS) are major digital threats in today's cyberspace. Defense mechanisms such as *Intrusion Detection Systems* aim at finding these and other malicious activities in network traffic. They predominantly use *signature-based* approaches to effectively detect intrusions. 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 tools in detecting intrusions, but it has not been studied if they are also able to detect unseen variants. In this research, we study to what extent supervised learning algorithms are able to detect novel variants of *application layer* (D)DoS attacks. To be more precise, we focus on detecting *HTTP* attacks targeting a web server. 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 (D)DoS variants are successfully detected by binary classifiers learned to distinguish benign entries from another (D)DoS attack. Despite this result, we demonstrate that the performance of a classifier trained on detecting variant A and tested on finding variant B is not necessarily similar to its performance when trained on B and tested on A. At last, we show that using more types of (D)DoS attacks in the training set does not necessarily lead to a higher detection rate of unseen variants. Thus, selecting the right combination of a machine learning model with a (small) set of intrusions included in the training data can result in a higher novel intrusion detection rate.**

***Keywords*—Machine learning; intrusion detection; anomaly detection; closed-world assumption.**

## I. Introduction

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. Significant threat types for these servers are *Denial-of-Service* (DoS) attacks and their distributed variant (DDoS). Tremendous problems for organizations and individuals arise when legitimate users cannot access data due to these attacks. Modern (D)DoS 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 the burglar alarm of the cybersecurity field [1]. It monitors network traffic and aims to detect malicious activities. Generally speaking, the two mainly used methodology classes by these systems are *signature-based* and *anomaly-based* [2]. 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 which 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 intrusions. 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 [3]. ML algorithms are highly flexible and adaptive methods to find patterns in big stacks of data [4], but they seemed better at this task rather than discovering meaningful outliers [3]. Modern (D)DoS attacks are often occurring 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 [5][6]. Firstly, the performance of most of these methods is measured on outdated datasets [7]. 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 they have generated more intrusion datasets [8]. Still, the available datasets are often limited to features extracted from the transport and network layer, but 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 assumption*, training, and test classes are the same, or an *open-world assumption* with unrelated attacks. However, it is not tested how the methods perform in an open-world setting with novel variants.

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 a set of (D)DoS attacks and we evaluate them on their ability to detect unseen variants of these attacks. We examine how the selected classifiers perform when using only a single (D)DoS variant in the training dataset on this task. Afterward, we study the effect of combining (D)DoS variants in the training phase on the performance of classifiers detecting unseen variants. Furthermore, we give 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 [9].

The main contributions can be summarized as follows: Firstly, we show that ML classifiers are to a great extent able to detect known (D)DoS attacks in a closed world setting. 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. This is however not a two-way street: learning to detect attack A and being able to also detect attack B does not imply that it is vice versa. Thirdly, we show that training on imbalanced data has an adverse effect on the evaluation performance of some ML classifiers. Finally, we demonstrate that it is not necessary to use many (D)DoS 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 states which datasets are selected for this research. Section IV describes how these datasets 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 in this section. Section V outlines the conducted experiments. Section VI shows the results of the conducted experiments. Finally, we conclude and summarize in Section VII.

## 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. [10] introduces a feature-based TL approach to find novel cyberattacks by mapping source and target dataset in an optimized feature representation. This approach is however very dependent on a similarity parameter and the dimensions of the new feature space. Therefore, [11] 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 [12], 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 [13]. In their research, an intrusion detection method is introduced which 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 [14].

## III. Datasets

It is stated that 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 spans over a long time [8]. The main reasons for a lack of appropriate datasets satisfying these properties are privacy concerns regarding recording real-world network traffic and labeling being very time-consuming. However, researchers have been able to generate synthetic or anonymized datasets which satisfy some of these ideal properties. It is therefore recommended to test methodologies on multiple datasets instead of only one [3]. In this research, we focus on the detection of (D)DoS attacks and for that reason have selected the CIC-IDS-2017 [15] and the CIC-IDS-2018 [16] datasets created by the Canadian Institute for Cybersecurity (CIC). These popular datasets fulfill properties such as being correctly labeled, publicly available, up-to-date, and containing (D)DoS variants.

## IV. Methodology

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. 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 [17]. These features mainly describe network and transport protocol activities, but there are no features describing application activities. As this study focuses on detecting application layer (D)DoS variants, 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) [18]. Zeek is a passive standalone IDS and derives an extensive set of logs describing network activity. These logs include an exhaustive record for all sessions seen on the wire, but also application layer documentation. Zeek was also used as a feature extraction tool for the creation of other popular network intrusion detection datasets, e.g., DARPA98 [19] from the Defense Advanced Research Projects Agency (DARPA) and the UNSW-NB15 [20] from the University of New South Wales (UNSW). It has a good track record in creating intrusion detection datasets and therefore an appropriate tool.

Zeek generates by default 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 (D)DoS 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 attacks are excluded as we only focus on these attacks.

## B. 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 categorical features are smartly one-hot-encoded. 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) [21] 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 state with which a TCP connection was ended. 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

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 |

that classifiers were more 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 ([d, t, g, w]) are one-hot-encoded for both originator and responder.

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 on which to apply this command, 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. Common used commands are {GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, 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 `URI` (Uniform Resource Identifier) stated in a request line. This `URI` can be parsed in different components by a library called urllib [22]. Figure 1 gives an example of how this tool splits a *URL* (Uniform Resource Locator) 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) and the `referrer` (all components) and these descriptive statistics are also extracted for these features.

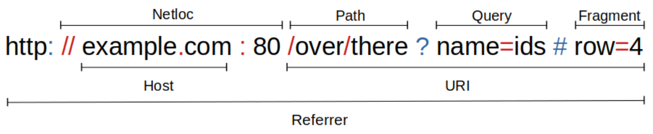

Figure 1. Example URL showing the four components parsed by urllib and the component coverage of extracted features by Zeek.

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 is registered what type of data ({application, audio, example, font, image, model, text, 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.

## C. 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 524,698 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 approximately balanced for both datasets. However, if we differentiate between (D)DoS attacks, 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*.

TABLE III. CLASS DISTRIBUTION OVER THE HTTP ENTRIES.

| | | CIC-IDS-2017 | | CIC-IDS-2018 | |
|---|---|---|---|---|---|
| **Class** | **(D)DoS** | **Amount** | **Percentage** | **Amount** | **Percentage** |
| Benign | - | 258,197 | 49.209% | 6,252,950 | 65.169% |
| Botnet | DDoS | 736 | 00.140% | 142,925 | 01.490% |
| GoldenEye | DoS | 7,908 | 01.507% | 27,345 | 00.285% |
| HOIC | DDoS | 0 | 00.000% | 1,074,379 | 11.197% |
| Hulk | DoS | 158,513 | 30.210% | 1,803,160 | 18.793% |
| LOIC | DDoS | 95,683 | 18.236% | 289,328 | 03.015% |
| SlowHTTPTest | DoS | 1,416 | 00.270% | 0 | 00.000% |
| Slowloris | DoS | 2,245 | 00.428% | 4,950 | 00.052% |

## D. 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 will be 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] |

## V. EXPERIMENTAL SETUP

In this section, we describe the experiments conducted in this research. First, the three experiments to tackle the research aim are described. Afterward, the evaluation metric to measure the performance of an ML classifier is given.

## A. Proposed approach

It is common practice in ML to split a dataset into two non-overlapping sets: a training set and a test set. To get a proper estimation of the performance of a classifier on the task at hand, the train-test split should be performed multiple times. 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. Each split is performed in a stratified manner to maintain the class distribution (Table III). In the training set, a validation set (20%) is randomly selected to obtain the best hyperparameters for each model. 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 assumption. To test this, the training data and the evaluation data will contain the same two classes: *Benign* and one attack. For example, in one setup we train on *Benign* against *Hulk* and test on the same two classes. This gives insights into the upper bound to what extent our models could achieve if we would have known about the attack.

*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 train on *Benign* against *Hulk* entries and evaluate the trained model on a test set containing *Benign* and *LOIC* (Low Orbit Ion Cannon). The labels of the attacks in the train and test set are both converted to a new label named *Malicious* so that the task is still a binary classification problem. This experiment shows how similar the training attack is to the test attack.

*3) Class Importance to Detect Novel Variants:* Finally, we study what we call *class importance*: does learning on a combination of multiple attacks help identify novel variants. We look at combinations of (D)DoS attacks 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*. To make this a binary classification problem, the attacks are again mapped to the *Malicious* class. This experiment shows if combining cyberattacks in the training set helps to detect novel variants. Also, it can be tested if adding certain classes in the training set impacts the detection of novel variants.

### B. Evaluation Metrics

In our classification task, the positive class represents malicious instances while the negative class represents benign entries. The considered evaluation metric to test the selected classifiers is the $F_1$ score, which is the harmonic mean between *recall* and *precision*. Recall shows the ratio intrusions the classifiers were successfully able to detect, while precision gives the ratio between the *true positives* and the number of positively predicted instances. For (D)DoS attacks 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 for this imbalance.

## VI. Experimental Results

In this section, we show the results of the three experiments performed in this research. First, we discuss the results of a dataset with a closed-world assumption. Secondly, this assumption is relaxed and we look at the performance of models in detecting known and novel attacks. At last, we discuss the results of classifiers trained on a set of attacks to detect a novel attack. The results of the experiments are gathered by testing the classifiers on 20 different train-test sets 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 parameter tuning. We have selected randomly 10% of the training data for hyperparameter search for the DT and RF model for the CIC-IDS-2018. Randomly 1% was selected for hyperparameter search of the KNN model and the same percentage was randomly selected from the training data to evaluate the model. For the CIC-IDS-2017, no subset sampling was required.

### A. Detecting Known Attacks

In this experiment, the classes included in the learning dataset are the same as the test dataset. Table V shows the average $F_1$ scores if classifiers are tested on the task of detecting known attacks. Each row in this table shows the attack used in the training, as well as in the testing set. It can be observed that in almost all scenarios the considered models are able to learn the relevant characteristics of the considered attacks. One exemption is the GNB model learning and testing on the *SlowHTTPTest* attack. This model obtained a sufficient recall (0.997), but an inferior score on its precision (0.154). Even though the model is able to detect most malicious instances, there are many false positives leading to a lower precision.

TABLE V. $F_1$ SCORES OF CLASSIFIERS DETECTING KNOWN INTRUSIONS.

| **CIC-IDS-2017** | **GNB** | | **DT** | | **RF** | | **KNN** | |
|---|---|---|---|---|---|---|---|---|
| **Attack** | **Mean** | **Std** | **Mean** | **Std** | **Mean** | **Std** | **Mean** | **Std** |
| Botnet | 1.0000 | 0.0000 | 0.9971 | 0.0046 | 0.9998 | 0.0008 | 0.9909 | 0.0076 |
| GoldenEye | 0.9972 | 0.0010 | 0.9997 | 0.0002 | 1.0000 | 0.0000 | 0.9983 | 0.0006 |
| Hulk | 0.9990 | 0.0002 | 0.9999 | 0.0000 | 1.0000 | 0.0000 | 0.9999 | 0.0000 |
| LOIC | 0.9999 | 0.0000 | 1.0000 | 0.0000 | 1.0000 | 0.0000 | 1.0000 | 0.0000 |
| SlowHTTPTest | 0.2339 | 0.2065 | 0.9955 | 0.0042 | 0.9956 | 0.0031 | 0.9874 | 0.0046 |
| Slowloris | 0.9013 | 0.0078 | 0.9976 | 0.0016 | 0.9969 | 0.0023 | 0.9929 | 0.0035 |
| **CIC-IDS-2018** | | | | | | | | |
| Botnet | 0.9998 | 0.0001 | 1.0000 | 0.0000 | 1.0000 | 0.0000 | 0.9974 | 0.0011 |
| GoldenEye | 0.9919 | 0.0006 | 0.9843 | 0.0010 | 0.9914 | 0.0004 | 0.9536 | 0.0051 |
| HOIC | 0.9964 | 0.0001 | 0.9964 | 0.0001 | 0.9964 | 0.0001 | 0.9961 | 0.0002 |
| Hulk | 0.9999 | 0.0000 | 1.0000 | 0.0000 | 1.0000 | 0.0000 | 0.9997 | 0.0000 |
| LOIC - HTTP | 1.0000 | 0.0000 | 1.0000 | 0.0000 | 1.0000 | 0.0000 | 1.0000 | 0.0000 |
| Slowloris | 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 variant of the learned attack? Figure 2 shows the average $F_1$ scores achieved by the classifiers in this experiment. The diagonal of this matrix shows the $F_1$

scores of the closed-world assumption, also obtainable from Table V, while numbers outside this diagonal are the scores of detecting novel attacks. It can be observed in the open-world setting that *Botnet* attacks are hard to detect in this setting, neither can it easily be used to detect other variants. However, there are situations where classifiers are able to detect novel variants. This is, however, not symmetrical: learning attack A and finding attack B does not mean it works also the other way around.

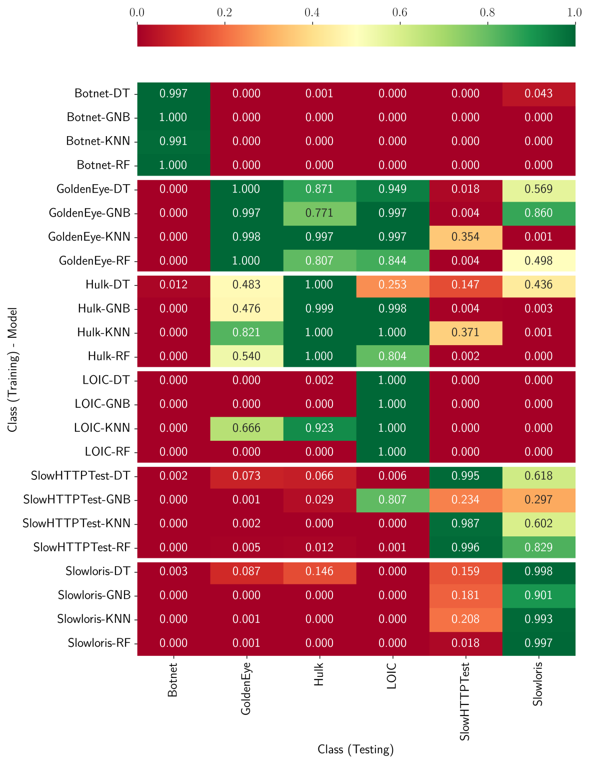

Figure 2. Average $F_1$ scores for the CIC-IDS-2017 dataset tested using 20 different train-test settings.

The same approach is applied to the CIC-IDS-2018 dataset. Figure 3 shows the results of the same experimental setup performed on the CIC-IDS-2018. Similar results are observable on the diagonal: ML algorithms are 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 are 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 it works the other way around.

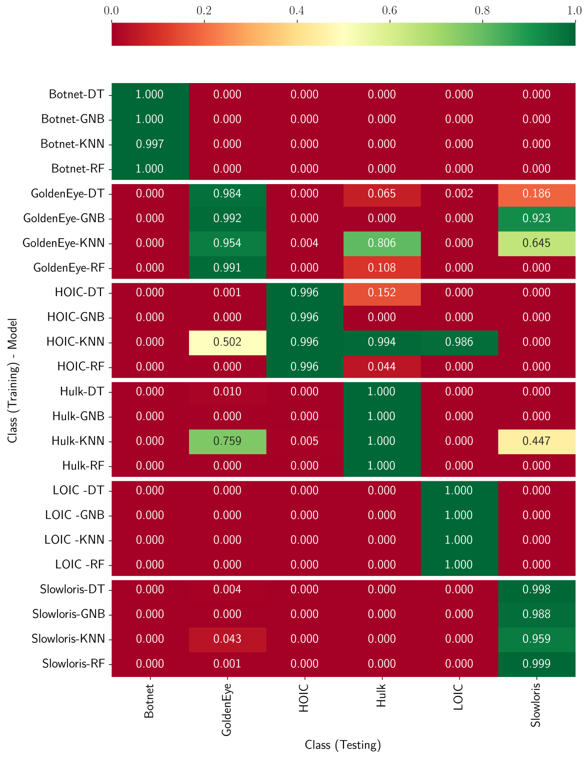

Figure 3. Average $F_1$ scores for the CIC-IDS-2018 dataset tested using 10 different train-test settings.

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

In our last experiment, we look at combining attacks in the learning phase to detect a novel variant. The objective here is to find a set of attacks leading to the highest novel attack detection performance. Table VI shows the results of the classifiers using a set of attacks to learn from on and the corresponding combination of attacks that led to the highest performance. Despite the fact that models can use more attacks to detect a novel variant, it is not necessarily the case that this yields the highest detection rate: even a few attacks are 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, as bold indicated performances show the highest, 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 as in experiment 2. These models could have been improved by downsampling benign entries so that the training classes are balanced.

TABLE VI. HIGHEST OBTAINED $F_1$ SCORE FOR EACH MODEL BY TRAINING THEM ON MULTIPLE INTRUSIONS TO DETECT A NOVEL ATTACK.

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

## VII. CONCLUSION

This research provides a procedure to construct intrusion detection datasets combining multiple layers with the tool Zeek. Zeek generates a bunch of extensive log files and two of them are selected to create a machine learning admissible dataset for the detection of (D)DoS attacks. This procedure to create such a dataset is not limited to only these protocols but can be extended to also combining other protocols, such as TCP with the *File Transfer Protocol* (FTP). The aim of this research was to test to what extent ML 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 (D)DoS variants. The focus of this research was to study the detection of variants of (D)DoS intrusions, 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. Finding patterns in large datasets is a typical task for ML algorithms. In the second experiment, it is observed that there are scenarios in which classifiers are able to detect a novel variant when trained on a different (D)DoS 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 vice versa. The third experiment showed that it is not necessary to use many (D)DoS variants to detect a novel attack. Sometimes a few known attacks can already lead to the highest detection rate. For the last two experiments, it is observed that when the training data is very imbalanced, DT and RF are inferior in detecting novel attacks in an open-world assumption. GNB seems better at detecting novel attacks when this is the case.

To sum up, this research shows that ML algorithms can detect (D)DoS 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 intrusions included in the training data can result in a higher novel intrusion detection rate.

## REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] *NIDS.* (2022). [Online]. Available: https://github.com/etiennevandebijl/NIDS

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] *A realistic cyber defense dataset,* Canadian Institute for Cybersecurity, May. 2021. [Online]. Available: https://registry.opendata.aws/cse-cic-ids2018

[17] 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.

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

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

[20] 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.

[21] 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.

[22] *urllib* (3.9) [Online]. Available: https://docs.python.org/3/library/urllib.html