Title

Ludovico Toscano

# Abstract

In this thesis, we will see how explainable algorithms such as Shap can play a key role in improving the niche of network intrusion detection systems by helping to analyze HIKARI-2021 and NFS-2023-TE. From a recent study showing how most recent NIDS works use complex models without a real need, we demonstrate how a decision tree could be better than a complex GA2Mmodel such as the explainable boosting machine. Then, thanks to Shap, we analyze different models and demonstrate why these datasets are unsuitable for real-world usage.

### Section 1 - Introduction

According to (Catillo et al., 2023) nowadays is not difficult to find papers that achieve almost perfect performance scores in machine learning (ML) applied to intrusion detection systems (IDS), but as the authors say the problem is far from being solved. The study breaks down these issues into two parts: the first is related to the dataset itself, and the second is about the common mistakes in building an ML model. The paper takes many times as an example CIC-IDS 2017 (Sharafaldin et al., 2017) since among the most popular datasets, it is the newest.

This work aims not only to be a practical example of what has been exposed (Catillo et al., 2023), but we will work on newer datasets and add other requirements related to building an IDS using an ML model.

The dataset choice was NFS-2023-TE (Pekar & Jozsa, 2024) and HIKARI-2021 (Ferriyan et al., 2021). NFS-2023-TE is the latest of a series of refinements of CIC-IDS 2017, and it improves the already existing WMTC-2021 (Engelen et al., 2021) (Liu et al., 2022) and CRiSIS-2022 (Lanvin et al., 2023). While HIKARI-2021 is a completely new dataset which is inspired by CIC-IDS 2017 featuring 7 new content requirements on top of the 11 criteria (Gharib et al., 2016) used by the CIC-IDS 2017 authors. Since these datasets improve what has been done in previous work, we expect to match some of the requirements expressed in (Catillo et al., 2023). Moreover, since, to the best of our knowledge, HIKARI 2021 has not been analyzed yet, we will include an analysis of the dataset in this work.

The analysis of HIKARI 2021 plays an important role since datasets become quickly obsolete (Guerra et al., 2022), which means that practitioners must choose between obsolete but well-known datasets or ones that may contain some unknown flaws. So, since it is acknowledged that using obsolete datasets should be avoided, we think that double-checking on the datasets is a need. This process is divided into two parts. The first will check if the same issues found on CIC-IDS 2017 affect HIKARI 2021, while the second will use eXplainable AI (XAI) algorithms for EDA. Moreover, by using XAI algorithms such as Shap (Lundberg & Lee, 2017) we will go beyond creating a model that achieves good metrics performance; we will gain knowledge about feature importance and how the model behaves.

The non-goal of this work is to make a definitive guide on how to make IDS using ML models, as we will see some of the issues still need to be addressed. While taking a step further in the right direction, we know this work could contain some of the abovementioned issues.

### Section 2 – Background Knowledge

This section contains some technical knowledge needed for those who are not computer scientists. The explanations have been simplified to make this section as short as possible.

What does an IDS do?

An IDS is usually a physical device connected to the router. Its job is to read all the connections flowing through the router and search for malicious traffic. If an IDS finds some malicious traffic, it tells the router to close all the related connections.

To be effective, an IDS needs not only to detect all the attacks but also to be as fast as possible in closing the connection with the attacker. For example, in an attack that sends all your data to the attacker's server, you want the attack stopped as soon as possible. In this case, the slower the IDS, the higher the number of stolen files.

What is an IP address, and what is a port?

An IP address is a sequence of numbers that identifies the host in a network. The IP addresses are limited and assigned only when the host is active on the network. So, a host can have two different addresses at two different moments. At the same time, two hosts can have the same address at two different moments. IP should not be used as a training feature since the same IP address can not be assigned to different hosts simultaneously. Doing this will block normal users while an attacker can change its IP to avoid the identification.

A port is used to identify a specific service from a host. For example, a server can host a website and an email service. The website will be an HTTP service at port 80, while the email will be a POP service at port 110. It is essential to understand that the port can be changed, so it is not used to train a model. So, while the standard for HTTP is port 80 if I want, I can use port 80 for another service, so if a model is trained to think that all port 80 traffic is fine, it can be fooled in this way.

In the case of these datasets, IP and port are still important even if they are not supposed to be used because, if paired with a timestamp, they can make it possible to match what we see in the dataset with what we see in the pcap file. This is necessary to eventually find issues on how the dataset was built or to make the generation of new datasets with new tools possible.

What is a flow?

For example, if I download an image from the Internet, I open a connection from the client to the server; this is a flow. The problem related to this explanation of a flow is what to consider a flow and what multiple flows are. For example, if I am browsing a website, I open different connections for each element of the website that I am downloading, such as all the images, the text of the website, and so on.

So, while browsing a website, I create many different flows. In the case of our dataset, these flows are based on the TCP or UDP standard. The difference between these two protocols is that a TCP connection is supposed to send some opening and closing packages, making it easy to understand when the flow begins and ends. In contrast, UDP does not have an opening and closing sequence. So, in the case of UDP, we will assume that the first package we see going from the client to the server is the opening one, while for deciding when the connection is closed, we usually use a timeout based on the latest seen packet.

Deciding on the correct threshold for this timeout is a problem without a correct answer. A longer timeout can be problematic since it can delay the identification of the threat and lead to seeing two connections as one connection. The fact that two different connections can be identified as one is rare because while the IP remains the same during the capturing session and the server port remains the same, the client port changes; it selects randomly from a pool of numbers.

While the timeout problem is always there for UDP connections, TCP connections, for some reason, may not be closed properly. In that case, the TCP connection will be closed after a timeout.

How these datasets are made

The process of making a dataset for training an IDS starts with setting up a testbed to do the data capturing part, which will lead to storing the data captured in pcap files and processing them to make a csv file. All the steps we will discuss here are described in CIC-IDS 2017 and HIKARI 2021 papers.

The testbed consists of a series of devices in the case of the datasets that I have chosen they are all computers and servers that are connected to the same network so that they can simulate normal and abnormal network behavior.

Once the testbed is set up, the capturing device starts to capture some session of normal traffic, which means that a computer is instructed to browse the Internet as a normal person would do, and some session of attack, in which one or most hosts in the network are attacked by a hacker.

The data we are talking about are IP packets containing TCP or UDP as payloads. Usually, this task is performed with tools such as tcpdump, which reads the flow of packets in the network and makes a pcap file of the data capturing session. These files contain the packet header, which contains all the metadata of the packet, such as destination, source, size of the packet, and a bunch of flags. We will talk about flags in detail later. Besides, the header packets also contain the payload, which is the data transported over the network.

Usually, these pcap files are released along with the dataset so that practitioners can analyze them and even create a new dataset from an existing pcap file. WTMC-2021, CRiSIS-2022, and NFS-2023-TE are all made from the pcap files of CIC-IDS 2017; their goal was not to create a new dataset from scratch but to improve an existing one.

Once we have made a pcap file, we can build our csv with a flowmeter tool, like CICFlowMeter (Lashkari et al., 2017) which was used for CIC-IDS 2017 or NFStream (Aouini & Pekar, 2022) used for NFS-2023-TE. These tools make the observations for our dataset, not about the single packet but the entire flow.

We must label once we have created a CSV file with all the connections. There are different ways to analyze the traffic, which are disclosed in the dataset papers. It is essential to know that once the attack's packet has been found, it will be labeled using the unique combination of IP, port, and timestamp. The timestamp of the flow is the same as the first packet that has started the flow.

Why do we use ML models?

Historically, IDS were working based on the signature of the payload data. Signature-based IDS make a signature of the data in the payload and then check if the signature matches a known attack in a database. The problem with this kind of solution is that they can only detect known attacks, and with the growing dimension of these databases, they are slower than ML models. Moreover, since traffic is ciphered nowadays, this technique is less effective. ML models do not rely on the payload; instead, they use the flow statistics the flowmeter makes. So, if a new attack is created, it is

possible that an ML model can detect the attack if it behaves similarly to a known attack.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Paper | *Avoid attack-revealing features and ease of detection* | *Avoid unmotivated complexity* | *Use of the evaluation metrics* | *Source code* | *Explainability* | *Updated dataset* | *All attacks* | *Dataset used* |
| (Kwon et al., 2023) | Yes | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Noori et al., 2023) | Yes | Yes | Yes | No | No | No | No | Hikari-2021 |
| (Louk & Tama, 2023) | - | No | Yes | No | No | No | No | Hikari-2021 |
| (Rajak et al., 2022) | - | No | Yes | No | No | No | Yes | Hikari-2021 |
| (Fernandes & Lopes, 2022) | No | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Fernandes et al., 2023) | No | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Chauhan & Shah Heydari, 2020) | Yes | No | Yes | No | No \* | No | Yes | CIC-IDS 2017 |
| (Sarhan et al., 2022) | Yes | Yes | Yes | No | No | No | Yes | CIC-IDS 2017 |
| (Zavrak & Iskefiyeli, 2020) | Yes | Yes | No | No | No | No | Yes | CIC-IDS 2017 |
| (Kurniabudi et al., 2020) | Yes | Yes | No | No | No | No | Yes\*\* | CIC-IDS 2017 |
| (Yulianto et al., 2019) | Yes | Yes | Yes | No | No | No | No | CIC-IDS 2017 |
| (Maseer et al., 2021) | Yes | Yes | Yes | No | No | No | Yes\*\* | CIC-IDS 2017 |
| (J. Liu et al., 2021) | Yes | Yes | No | No | No | No | Yes | CIC-IDS 2017 |
| This work | Yes | Yes | Yes | Yes | Yes | Yes | Yes |  |

# Section 1 - Datasets used

In February 2024 the three datasets with more citations in Google Scholars were CIC-IDS 2017 (Sharafaldin et al., 2018) with 3149 citations, NLS-KDD (Tavallaee et al., 2009) with 5131 citations, and UNSW-NB15 (Moustafa & Slay, 2015) with 2740 citations these two are improvements based on the raw data of KDD Cup ’99 (Lee et al., 1999). Our focus is on two datasets derived from CIC-IDS 2017, which illustrate why it may no longer be the best option.

By making CIC-IDS 2017 the authors have achieved for the first time the goal of covering all the 11 criteria given by (Gharib et al., 2016) by using CICFlowMeter (Lashkari et al., 2017). However, the analysis of the pcap files by (Engelen et al., 2021) and (Liu et al., 2022) revealed over 5% corruption in dataset labeling and non-compliance of CICFlowMeter with TCP connection closure standards (Brownlee et al., 1999). This led to the creation of WTMC-2021. One year later WTMC-2021 was improved with the name of CRiSIS-2022 (Lanvin et al., 2023) by adding the attack port scan that was not labeled and sorting and removing the duplicates in the pcap file. In 2023 (Pekar & Jozsa, 2024), noticed that even after the patching CICFlowMeter was not closing the connection after a FIN or RST flag and was also making some missing or negative values, so they decided to make NFS-2023-TE by using NFStream (Aouini & Pekar, 2022). NFS-2023-TE creates a new flow at first FIN or RST flag like most of the flow analyzers do (Hofstede et al., 2014), while NFS-2023-nTE creates a new flow after a timeout like WTMC-2021 and CRiSIS-2022 do, this second dataset has been made only for the sake of comparison with its predecessor.

Is possible that shortly things will change, since the authors of CICFlowMeter said in an email exchange that they are making a new tool written in Python with more than 50 new features for a total of 130 features.

Hikari 2021 (Ferriyan et al., 2021) is a dataset inspired by CIC-IDS 2017 but includes 7 new requirements; they also provide the pcap files, a csv file, and a pkl of the dataset. They provided 84 traffic features and 2 target features, one multi-categorical and the other one binary, among the 84 traffic features most of them are inspired by the ones of CIC-IDS 2017. In this dataset, the authors have focused only on web attacks since 80% of the attacks nowadays are done on the application layer. So, they have performed 3 different attacks, but on the dataset, a fourth one is present since while they were analyzing the background traffic, they found a crypto miner attack. The raw data were extracted using tcpdump, while the labeling and analysis of the background traffic were done with Zeek, which also added some features that are not present in CIC-IDS 2017. These features are the IP address, source and destination ports, and uid. Since the release of the paper, the dataset has received an update that improves the dataset by increasing the size of the minority classes.

We decided to build our models using NFS-2023-TE and HIKARI-2021, which, as far as we know, are the best datasets available.

Section 2 - Explainable algorithm

Shap (Lundberg & Lee, 2017) is a Python library that makes use of a game theoretic approach by computing an approximation of the Shapley values to create post hoc explanations for any machine learning models. With this work among other explainers, they introduced Kernel Shap a model-agnostic explainer adapted from Lime (Ribeiro et al., 2016), and Deep Shap a model-specific explainer for neural networks that has adapted the algorithm of Deep Lift (Shrikumar et al., 2017).

Lately, other two papers (Lundberg et al., 2018, 2020) introduced Tree Shap, a model-specific explainer for trees and ensembles of trees, which is so fast compared to Deep Shap and Kernel Shap that in most cases, can explain the entire training set in a few seconds. However, the library Fast Tree Shap (Yang, 2021) introduced an optimized version of the Tree Shap algorithm, which is faster in single-core but can overcome the original algorithm's single-core limitation, allowing even faster parallel execution. Since Fast Tree Shap has all the characteristics of Tree Shap, when we talk about Tree Shap, we will refer to both Tree Shap and Fast Tree Shap algorithms.

Tree Shap explainers and Linear Shap can provide true-to-the-model explanations (H. Chen et al., 2020) instead of other explainers that are true to the data. By breaking the dependence between features by following the rules of causal inference (Janzing, 2019), true-to-the-model explanations assign a value different from zero to a feature only if the model uses it. In the case of a true-to-the-data explanation, if two features correlate 100%, the same importance will be assigned to them even if one is not used.

Even though Tree Shap is a fast post hoc explainer, a glass box model like the EBM (Explainable Boosting Machine (Lou et al., 2013)), a GA2M model that does not require any post hoc explanation. It promises to be accurate as a state-of-the-art black box model without sacrificing explainability.

Section 3 – Methodology

In the survey (Zhang et al., 2022), the authors highlight some limitations of explainable AI for cyber security, a big subject that includes intrusion detection. We will focus on how two future improvements highlighted in this survey can be solved by using boosted models such as the Random Forest (Breiman, 2001), LightGBT (Ke et al., 2017), XGBoost (T. Chen & Guestrin, 2016), CatBoost (Dorogush et al., 2017) with Fast Tree Shap, or EBM.

The first limitation highlighted is the trade-off between model performance and explainability, this is because most of the related works make use of complex black box deep models. These black boxes can usually be explained only with model agnostic models like Kernel Shap. However, this process is computationally expensive, leading to a tradeoff in the accuracy of the explanation. Moreover, adversarial attacks (Slack et al., 2020) have proven to work against LIME (Ribeiro et al., 2016) and Kernel Shap. Using EBM or Tree Shap makes it impossible to realize this attack.

While the mentioned issues are related to the explainable cyber security world (Catillo et al., 2023) have written a paper about some of the most common issues encountered in machine learning models applied to network intrusion detection systems. They have listed two kinds of issues: those about public intrusion datasets and their implications from the machine learning perspective. In Table 1, we have the list of issues related to the datasets. As for the ML implications, the only problem we cannot deal with is the lack of transferability. It is technically possible to make an IDS using NFStream, but given the open issues with the dataset, obtaining the same results in the real world is impossible. In (table 2) we have a comparison with our work and a list of selected papers related to the ML implication. On top of the existing requirements, we added the availability of source code, the lack of explainability, missing attacks, and the use of the updated dataset version. The source code is the most important since some authors forget to mention things like the features selected or

how they do data partitioning. Moreover, the availability of the source code makes it possible to rerun the model, even with different performance metrics, added explainability, or a new dataset refinement. At the same time, it is clear that using an explainable algorithm and a dataset without flaws is essential. At the same time, some papers did not use all the categories available in the dataset, sometimes even without mentioning the fact. We selected all the papers that in February 2024 have used Hikari-2021 flows for making ML models, which are 6, the top 4 most cited papers about CIC-IDS 2017, and the top 2 most cited papers about CIC-IDS 2017 that made use of Shap.

|  |  |  |
| --- | --- | --- |
| Issue | HIKARI-2021 | NFS-2023-TE |
| *Simplification of the data collection environment* | Not addressed | Not addressed |
| *Contemporaneity and effectiveness of the attack* | Hikari 2021 focuses only on encrypted traffic at the application layer, saying that 80% of attacks are done at this level. As far as we know, this is the most updated dataset available; this would be a good option until a new one emerges. | The data used was generated 7 years ago. Moreover, some attacks in the dataset have been proven ineffective nowadays. In (Catillo et al., 2021) they |
| *Representativeness of the normal baselines* | Not addressed | Not addressed |
| *Bugs of the feature extractor and incorrect ﬂow records* | Hikari 2021 does not provide any source code and mentions that the labeling was made with Zeek alongside an undefined Python tool, so we analyzed the dataset before using it. This part will be discussed later. However, in an email exchange with the authors, they said that the source code would be released. | To avoid this problem NFS-2023-TE was made with NFStream which was chosen since it is an open-source tool with a wide user base, and the labeling process has been documented and released alongside the Python code used. |
| *Data Labeling (Was the traffic analyzed or labeled based on IP, port, and timestamp?)* | The background data were analyzed with Zeek, which led to the discovery of an attack. HIKARI 2021 does not provide the timestamp, and part of the payload is encrypted to ensure privacy; it is not possible to prove if there is any flaw in the labeling. | NFS-2023-TE should deliver on this point since it has been made on top of the refinement that has improved the original issues of the labeling part. |
| *Class imbalance* | Probing is around 2,9 times bigger than Bruteforce, which has 7988 samples. This difference is considerably smaller than NFS-2023-TE, making over-sampling possible if needed. | PortScan is around 14574 times bigger than Hearthbleed, with only 11 samples. With such a small sample, it is impossible to make a good model. |

Table 1- Typical dataset issues

After having seen all the issues from the dataset side, is time to focus on the ML implications.

Attack-revealing features and ease of detection

(D’hooge et al., 2023) they proved how some revealing features make it easy for an OneR model to achieve high scores, while we could avoid ease of detection if we handled attack-revealing features by selecting features. In both datasets, all the features related to IP addresses, ports, Mac addresses, and the binary target label have been removed. NFS-2023-TE and NFS-2023-nTE also contain the timestamps of the connections that have been removed, moreover, there is a feature called protocol, which led to some biases since all the attacks are performed over TCP. HIKARI 2021 contains a label called uid and an index that has been removed.

Data partitioning

The random under-sampling, training split, and cross-validation were all made by stratifying using the attack classes, preventing the loss of the smallest classes. We choose 80% of the data for training and the remaining for testing.

Unmotivated complexity

All the mentioned models have been compared to a decision tree. Moreover, the fitting and classification time and explanation time have been measured.

Use of the evaluation metrics

We used two metrics that sklearn calls F1 macro and F1 weighted; the first computes the F1 score of each class and then does the unweighted mean while the latter does the weighted mean.

Lack of transferability

For now, creating a network intrusion detection system with a public dataset and running it in the real world while maintaining the same results is still an open problem.

Section 5 – Results

Data set analysis

While in NFS-2023-TE we did not find any flaw in the dataset, in HIKARI 2021 we noticed that the flag count for FIN and RST is higher than how it is supposed to be having a maximum of 140 FIN packages and 110 RST packages. Moreover, the longest flow lasted 17942 seconds, about 4.9 hours. Since the capturing sessions of HIKARI 2021 lasted between 3 to 5 hours, this means that the tool used did not have a timeout, nor was it closing the connection at the first RST or FIN flag. To the best of our knowledge, none of the papers about HIKARI 2021 have noticed this issue, which, if not noticed during the pre-processing of the dataset can be highlighted by the explanations of the algorithms.

Since there is no documentation about the labeling and the authors did not release the timestamps, this dataset is not suitable for real-world usage. Instead, we decided to show how explainability can be used to discover these flaws.

Given that we found these flaws in the CSV files, we also analyzed the Pcap files using the same tools as the CRiSIS 2022 reordercap and editcap with a window of 500 microseconds. We found that 0,03% of the packages were duplicated, and 0,23% were out of order. The authors confirmed that issue and did not perform preprocessing before making the dataset.

To achieve better results, we opted to under-sampling all the datasets. For HIKARI-2021, all the classes were sampled to have 7988 samples for each class; moreover, we merged Bening and background traffic. This number has been chosen since the smaller class is that big. Meanwhile, for NFS-2O23-TE, under-sampling was performed to have 738 samples for each class except for smaller classes. In this case, we decided to test different numbers of samples, and this was the one performing the best. This is a big limitation of this study, but the alternative of removing some attacks is not that better. With only 11 samples in the smallest class, over-sampling was not considered to avoid overfitting the synthetic data.

Model performance

The following training, classification, and explanation times have been measured on a Dell XPS 13 9315 with a 12th gen i7-1250u and 16gb of ddr5 ram running Fedora 39 with Linux 6.8. For the environment information, there is a YAML file on the GitHub repository with all the versions of the packages in the conda environment. Is important to notice that Sklearnex and daal4py have been used to improve inference time, and all the models have been set to use all the cores available.

The parameter tuning of Table 1 was done following the suggestions of the documentation for each library and by looking at what was working for the other models. We used the metrics f1 macro of sklearn to evaluate the results, and when different parameters were giving similar results shorter training time was used to lead the decision.



Table 2 - NFS-2023-TE and NFS-2023-nTe parameters

In Table 2 we have the training time, explanation time, and the sum of both in ms for each model tested with 6087 samples. For the explanation fast tree shap is set to automatically choose the best algorithm. These times are inaccurate; they have some variance but show the order of magnitude necessary to run each model. We can notice that even if EBM is the slowest during training, it can be faster than lightgbm and xgboost if we consider the explanation time. The random forest is close to the decision tree thanks to using only 10 trees which is considerably less than the GAM models. Another thing to consider is that even if Catboost uses fewer trees than the lightgbm and xgboost is slower to train.



Table 3 - NFS-2023-TE training times



Table 4- NFS-2023-TE – average F1 score of 5 fold

In Table 3, we compare the prediction time in μs for 1522 samples, in this case, we have used the magic function %timeit of ipython to compute the mean of different runs. Doing the mean is important because it is statistically more significant. With such a smaller duration, a warmup of the function is necessary to reduce the computation time. Without the warmup run, the classification time of the decision tree was 3 ms because, at the first run, the computer loaded the prediction function into memory, which took more time than the prediction itself.

Looking at the EBM we can notice that is the best when it comes to F1 macro but the worst with F1 weighted. Comparing these results with the decision tree shows that the decision tree has a problem with Hearthbleed and Web Attack - SQL Injection, which misclassifies 2 samples for the first and 4 with the latter. In comparison, the ebm misclassifies 111 samples in the DoS Slowhttptest. These results expose the need for a balanced dataset; we can argue that EBM is the best one because can handle well each attack, but on the other hand building a model over 11 samples will lead to something that will not work in the real world.



Table 5 - HIKARI 2021 parameters

Moreover, we need to consider the prediction time by looking at both the F1 results. Lightgbm is probably the winner by being the second best in both macro and weighted scores but is the worst in prediction time. Lightgbm is twice as slow as the random forest, which scores nearly the same in the F1 metrics, making the random forest the best alternative overall. The reason why the classification time is so important is that in case of a DoS attack that opens and closes the connection at a fast enough speed, the model needs to keep up with each new flow generated otherwise the NIDS will run out of service. Another reason why this classification time is important is that can make a difference between enabling or not the use on the edge and lead to less expensive devices when the GPU is not required for running an NIDS.

Table 4 shows how the parameters for HIKARI-2021 lead to less complex models except for the ebm. This is possible because the dataset is balanced after the undersampling.



Table 6- HIKARI-2021 training times

Therefore, the training times for 31952 training samples except for the ebm are lower than the ones of NFS-2023-TE.



Table 7 - HIKARI-2021 F1 score and times

Since HIKARI-2021 is balanced, there is no difference between F1 macro and F1 weighted. Since all the models score nearly the same, the analysis comes down to the decision tree, which is the fastest.

Model explanations

Figure 1 shows the feature importance of a decision tree for the class bot of NFS-2023-TE. The model has misclassified only 1 sample out of 737 using only a small subset of the available features. The same pattern of using a few features applies to the other attacks. The only exception is benign traffic, which requires more features to be detected. The reason why benign assigns positive importance to more features is that most of the splits lead either to an attack or benign traffic.

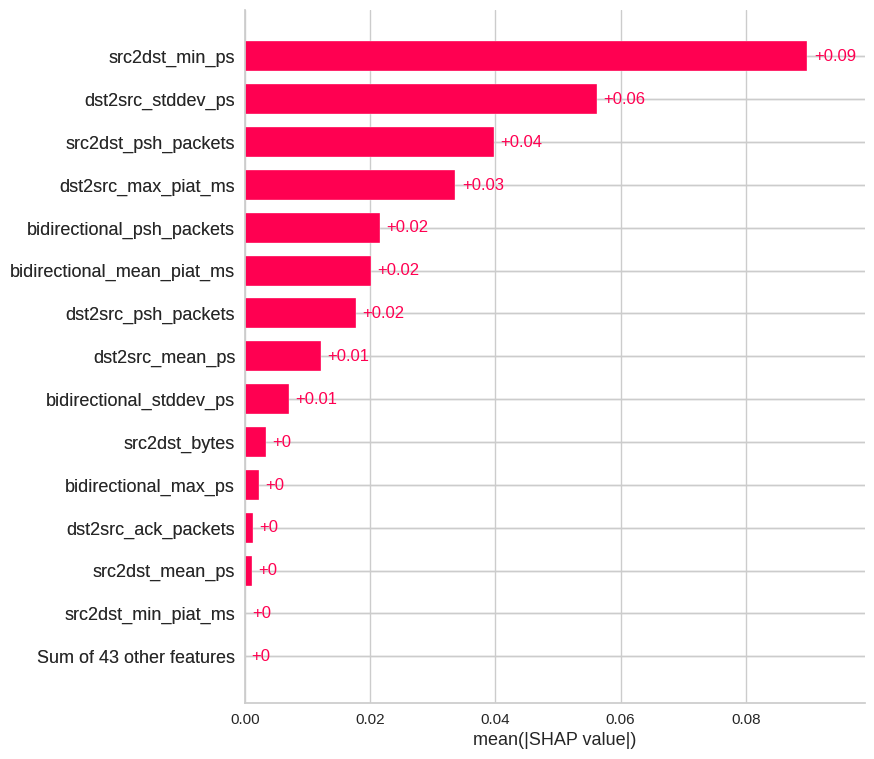


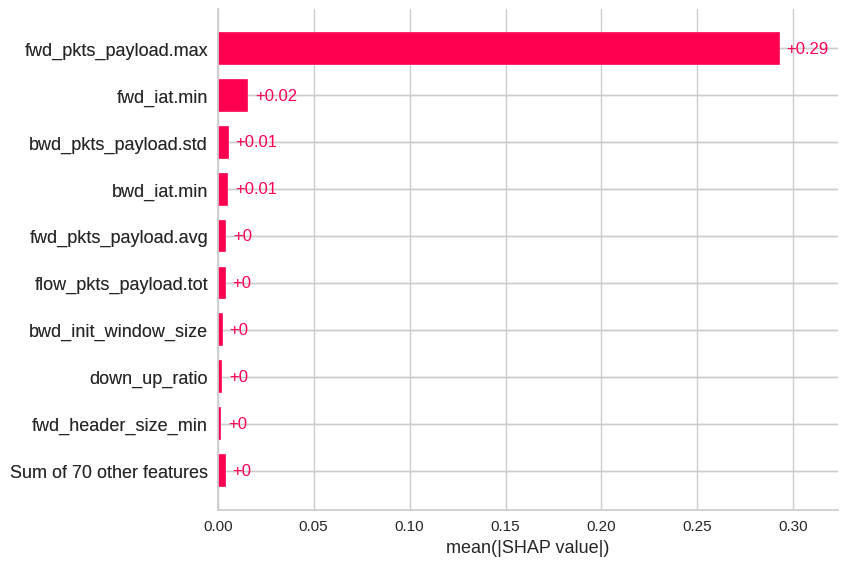
Figure 1 - NFS-2023-TE - decision tree feature importance with shap of bot class

Figure 2 - HIKARI-2021 - decision tree feature importance with shap of bruteforce-XML class

In Figure 2, we can see that the decision tree has given almost all the importance to one feature, which has been used as a root node and in a node at the first branch classifying the node as bruteforce-XML with a gini coefficient of 0.041. This explanation has led to an analysis of the feature by grouping all the samples by class and what has been found is that the value of this feature at the 25th percentile is 746 which is higher than the 75th percentile of all the other classes. In the class probing, all the samples have 517 as the value for this feature, which is the second most important.

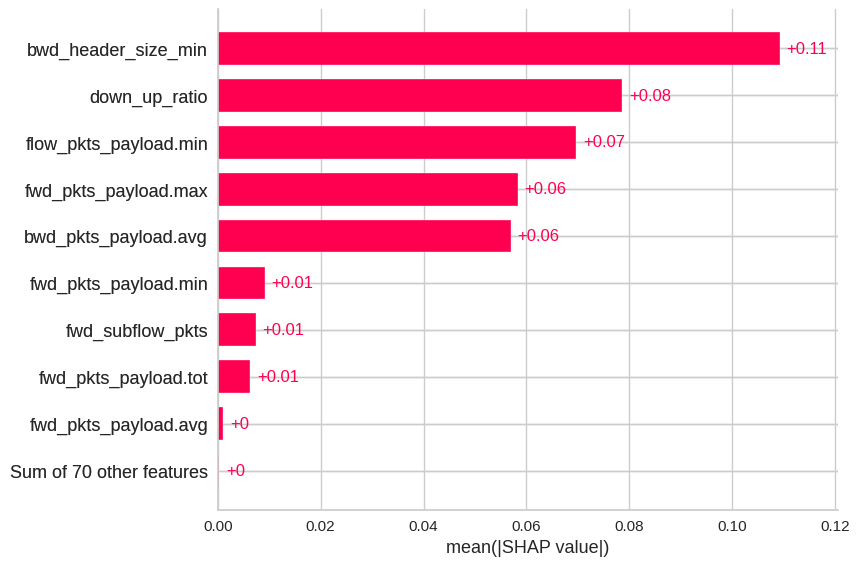


Figure 3- HIKARI-2021 - decision tree feature importance with shap of XMRIGCC CryptoMiner

Figure 3 is the feature importance of XMRIGCC CryptoMiner with the decision tree, again only a few selected features have been used the most important is bwd\_header\_size\_min which for this attack is always zero then we have down\_up\_ratio. Down\_up\_ratio for this class is always zero and is the most important feature for probing and the second most important for background, the reason is that background has an average of 0.8, probing is 1.3 while the remaining attacks have more than 8. Also, benign which has been merged with background shows an average of 2.

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Figure 4 - NFS-2023-TE - catboost impact on the model output of DDoS with shap sorted by highest magnitude of impact

Figure 4 is the explanation of the catboost algorithm where we can see that the feature with the highest magnitude of all is bidirectional\_fin\_packets, which shows up for Bot, DDoS, DoS Slowhttptest, DoS Slowloris, Heartbleed, Infiltration, and Web Attack – XSS as one of the two most important feature in average or the one with the highest magnitude of impact. By analyzing this feature, we can notice that Bot and Web Attack -XSS always have 1 FIN packet, while DDoS averages 99% of the samples with 1 FIN packet. This characteristic makes the difference in the other attacks cited at the 85th percentile of the distribution showing 0 FIN packets. To understand why we have a lot of 0 we analyzed the duration and for Infiltration and Heartbleed, most of the connection gets closed because of the timeout, while for DoS Slowhttptest and DoS Slowloris, it is hard to tell since they did not reach the timeout either they send an RST packet.

All these analyses conclude that using a more complex model than a decision tree or a random forest is difficult to justify for these datasets. However, this does not mean that the decision tree is ready for deployment in the real world. The reason behind this algorithm's success is the oversimplified test bed.

Section 6 – Conclusion

We first saw that CICIDS-2017 is nowadays one of the most popular datasets cited in Google Scholar, and then we saw how new proposals such as NFS-2023-TE and HIKARI-2021 aim to improve the work done with CICIDS-2017. In particular, NFS-2023-TE uses the same raw data as CICIDS-2017, while HIKARI-2021 proposes a new, updated test bed.

Then, we analyzed the flaws in the current literature and the dataset in use, which led to the proposal of a new methodology, with the addition of the explainability part, as seen in other work.

HIKARI-2021 has been analyzed using the criteria proposed by NFS-2023-TE, which led to the discovery of some flaws in the raw data and the dataset generation.

Finally, the analysis of the results shows how comparing simpler models such as the decision tree is necessary to have a benchmark for more complex models. Moreover, we have shown how explainable AI algorithms such as Shap need to be used to help build better models and to increase the accountability of the proposed models. Thanks to the explanations, we discovered something about the dataset, which has helped us avoid mistakes during the work. More than once, features that were not supposed to be in the training set were identified thanks to Shap.

Aouini, Z., & Pekar, A. (2022). NFStream: A flexible network data analysis framework. *Computer Networks*, *204*. https://doi.org/10.1016/j.comnet.2021.108719

Breiman, L. (2001). Random forests. In *Random Forests* (Vol. 45). https://doi.org/10.4324/9781003109396-5

Brownlee, N., Mills, C., & Ruth, G. (1999). *Traffic Flow Measurement: Architecture*.

Catillo, M., Pecchia, A., Rak, M., & Villano, U. (2021). Demystifying the role of public intrusion datasets: A replication study of DoS network traffic data. *Computers & Security*, *108*, 102341. https://doi.org/10.1016/j.cose.2021.102341

Catillo, M., Pecchia, A., & Villano, U. (2023). Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS? *Proceedings - 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume, DSN-S 2023*, 132–136. https://doi.org/10.1109/DSN-S58398.2023.00038

Chen, H., Janizek, J. D., Lundberg, S., & Lee, S.-I. (2020). *True to the Model or True to the Data?* https://github.com/slundberg/shap/blob/

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*, 785–794. https://doi.org/10.1145/2939672.2939785

D’hooge, L., Verkerken, M., Wauters, T., De Turck, F., & Volckaert, B. (2023). Castles Built on Sand: Observations from Classifying Academic Cybersecurity Datasets with Minimalist Methods. *International Conference on Internet of Things, Big Data and Security, IoTBDS - Proceedings*, *2023-April*, 61–72. https://doi.org/10.5220/0011853300003482

Dorogush, A. V., Ershov, V., & Yandex, A. G. (2017). *CatBoost: gradient boosting with categorical features support*. https://github.com/Microsoft/LightGBM

Engelen, G., Rimmer, V., & Joosen, W. (2021). Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study. *Proceedings - 2021 IEEE Symposium on Security and Privacy Workshops, SPW 2021*, 7–12. https://doi.org/10.1109/SPW53761.2021.00009

Ferriyan, A., Thamrin, A. H., Takeda, K., & Murai, J. (2021). Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. *Applied Sciences (Switzerland)*, *11*(17). https://doi.org/10.3390/app11177868

Gharib, A., Sharafaldin, I., Habibi, A., & Ghorbani, A. (2016). *An Evaluation Framework for Intrusion Detection Dataset*.

Guerra, J. L., Catania, C., & Veas, E. (2022). Datasets are not enough: Challenges in labeling network traffic. *Computers & Security*, *120*, 102810. https://doi.org/10.1016/j.cose.2022.102810

Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., & Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX. *IEEE Communications Surveys and Tutorials*, *16*(4), 2037–2064. https://doi.org/10.1109/COMST.2014.2321898

Janzing, D. (2019). *Causal Regularization*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *31st Conference on Neural Information Processing Systems*. https://github.com/Microsoft/LightGBM.

Lanvin, M., Gimenez, P. F., Han, Y., Majorczyk, F., Mé, L., & Totel, É. (2023). Errors in the CICIDS2017 Dataset and the Significant Differences in Detection Performances It Makes. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13857 LNCS*, 18–33. https://doi.org/10.1007/978-3-031-31108-6\_2

Lashkari, A. H., Gil, G. D., Mamun, M. S. I., & Ghorbani, A. A. (2017). Characterization of tor traffic using time based features. *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, *2017-Janua*, 253–262. https://doi.org/10.5220/0006105602530262

Lee, W., Stolfo, S. J., & Mok, K. W. (1999). *Mining in a Data-flow Environment: Experience in Network Intrusion Detection*.

Liu, L., Engelen, G., Lynar, T., Essam, D., & Joosen, W. (2022). Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. *2022 IEEE Conference on Communications and Network Security, CNS 2022*, 254–263. https://doi.org/10.1109/CNS56114.2022.9947235

Lou, Y., Caruana, R., Gehrke, J., & Hooker, G. (2013). Accurate intelligible models with pairwise interactions. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *Part F1288*, 623–631. https://doi.org/10.1145/2487575.2487579

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S. I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, *2*(1), 56–67. https://doi.org/10.1038/s42256-019-0138-9

Lundberg, S. M., Erion, G. G., & Lee, S.-I. (2018). *Consistent Individualized Feature Attribution for Tree Ensembles*. http://github.com/slundberg/shap

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, *2017-Decem*, 4766–4775. https://github.com/slundberg/shap

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

Pekar, A., & Jozsa, R. (2024). *Evaluating ML-Based Anomaly Detection Across Datasets of Varied Integrity: A Case Study*. https://intrusion-detection.distrinet-research.be/WTMC2021/tools

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). *“Why Should I Trust You?” Explaining the Predictions of Any Classifier*. https://doi.org/10.1145/2939672.2939778

Sharafaldin, I., Gharib, A., Lashkari, A. H., & Ghorbani, A. A. (2017). Towards a Reliable Intrusion Detection Benchmark Dataset. *Software Networking*, *2017*(1), 177–200. https://doi.org/10.13052/jsn2445-9739.2017.009

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, *2018-Janua*(Cic), 108–116. https://doi.org/10.5220/0006639801080116

Shrikumar, A., Greenside, P., Shcherbina, A. Y., & Kundaje, A. (2017). *Not Just A Black Box: Learning Important Features Through Propagating Activation Differences*. https://arxiv.org/

Slack, D., Hilgard, S., Jia, E., Singh, S., & Lakkaraju, H. (2020). Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods. *AIES 2020 - Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 180–186. https://doi.org/10.1145/3375627.3375830

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, *Cisda*, 1–6. https://doi.org/10.1109/CISDA.2009.5356528

Yang, J. (2021). *Fast TreeSHAP: Accelerating SHAP Value Computation for Trees*. http://arxiv.org/abs/2109.09847

Zhang, Z., Hamadi, H. Al, Damiani, E., Yeun, C. Y., & Taher, F. (2022). Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research. *IEEE Access*, *10*(July), 93104–93139. https://doi.org/10.1109/ACCESS.2022.3204051