FACULTY OF SCIENCES

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

Jozef Jankaj
Student number: 01904081

Supervisors: Dr. ir. Tim Wauters, Prof. dr. Bruno Volckaert
Counsellors: Laurens D'hooge, Miel Verkerken

Master's dissertation submitted in order to obtain the degree of Master of Science in Computer Science

Academic year 2024-2025

GHENT UNIVERSITY

Samenvatting

Met het snel veranderende landschap van de cybersecurity verschijnen er voortdurend nieuwe tools, aanvalstechnieken en exploits om misbruik van te maken. Onderzoek en ontwikkeling van nieuwe verdedigingsmiddelen is essentieel om de vertrouwelijkheid, integriteit en beschikbaarheid van onze netwerken en waardevolle data te waarborgen.

Het stoppen van een cyberaanval begint bij de detectie ervan. Hiervoor zijn Network Intrusion Detection Systems ontwikkeld, die automatisch aanvallen kunnen detecteren en een alarm kunnen genereren. Met de opkomst van artificiële intelligentie worden steeds vaker machine learning-technieken in dit domein toegepast. Echter, hoogwaardige modellen vereisen hoogwaardige data. Er zijn veel datasets geconstrueerd, maar elk van hen kampt met tekortkomingen die de bruikbaarheid beperken. Bovendien kunnen deze modellen, door de snelle evolutie van tools en technieken, snel verouderd raken.

Deze scriptie onderzoekt een nieuwe methode om de robuustheid van machine learning-modellen voor netwerkdetectie te verbeteren via het ConCap-framework. Aan de hand van CIC-IDS-2017 als casestudy wordt deze dataset gereconstrueerd en uitgebreid om te verifiëren of ConCap op haalbare wijze als vervanger voor CIC-IDS-2017 kan dienen en hoe het uitbreiden van CIC-IDS-2017 met ConCap de robuustheid van modellen kan versterken.

Summary

With the fast-evolving landscape of cybersecurity come new tools, new attack techniques and new exploits to abuse. Research and development of new defensive tools is imperative for maintaining the confidentiality, integrity and availability of our networks and our precious data.

The process of stopping any cyberattack starts at its detection. For this purpose, Network Intrusion Detection Systems have been developed to automatically detect attacks and raise an alarm. With the advent of artificial intelligence, more and more machine learning techniques are being used in this area. However, high quality models require high quality data. Many datasets have been constructed, but each came with a set of issues that reduce its usability. Furthermore, with the fast evolving tools and techniques, these models can be quickly rendered obsolete.

This thesis studies a new way of improving robustness of machine learning models intended for network intrustion detection through the ConCap framework. Using CIC-IDS-2017 as a case study, it gets reconstructed and extended to verify that ConCap can feasibly be used as a stand-in for CIC-IDS-2017 and how extending CIC-IDS-2017 through ConCap can enhance model robustness.

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

Jozef Jankaj, Miel Verkerken, Laurens D'hooge, Tim Wauters, Bruno Volckaert
IDLab, Department of Information Technology at Ghent University - imec, Ghent, Belgium
{jozef.jankaj, miel.verkerken, laurens.dhooge, tim.wauters, bruno.volckaert}@UGent.be

***Abstract*—As our world becomes increasingly interconnected, our networks face threats from more places than ever before. Existing protections have a hard time keeping up with the quickly evolving threats in the cyberworld, calling for quick improvements. One avenue for improvements is the area of machine learning based network intrusion detection systems (ML-NIDS), providing automated protection to the networks. However, the usefulness of these models quickly diminishes as new forms of attacks are discovered, and their robustness is questionable when facing variations of attacks. In this article, we explore a new way of increasing robustness of NIDS models through dataset augmentation using ConCap.**

I. Introduction

With the advent of the internet, physical threats to our security have spread to the virtual world. Our privacy is continuously threatened as we give more and more of it up for the convenience of the interconnected world. Plenty of adversaries are more than happy to attack, steal and destroy our systems. To address this Sword of Damocles, network security defenses have been a hot topic of research. While significant improvements in network security have been made thanks to the improvements in the message encryption, implementations of firewalls and general security practices, attacks against our networks are still prevalent. Timely detection and interruption of these attacks is of paramount importance for maintaining the confidentiality, integrity and availability of our networks. A Network Intrusion Detection System (NIDS) fulfills this purpose by monitoring the traffic and raising an alarm upon detection of malicious traffic. Existing systems [16, 24, 28] are implemented using preprogrammed rules or features of traffic.

However, the fast-paced nature of cybersecurity renders these systems quickly ineffective, as new attacks and variants of existing attacks are discovered and deployed. For this and other reasons, researchers turned their attention to Machine Learning based Network Intrusion Detection Systems (ML-NIDS). Both classical, shallow-learning methods [10, 17, 3] and deep learning methods [15, 23] have been previously explored in the literature. These methods rely heavily on high-quality data, and naturally, many datasets [19, 1, 13, 22] have been constructed and proposed as the benchmark for the ML-NIDS research.

Unfortunately, these datasets have been shown to contain mistakes [5, 6], reducing their usefulness. Issues such as poor documentation, misimplementations of attacks and faulty labeling, have been raised and as a result, the NIDS practitioners are reserved about implementing these models [2].

To address these and other issues, the ConCap framework [25] has been developed and utilized, leveraging the power of Kubernetes clusters to build out networks in-silico, conduct attacks and capture traffic for dataset creation. In this article, we look at using this technology for enhancing robustness of existing ML-NIDS datasets through dataset augmentation and take the CIC-IDS-2017 dataset [19] as a case study.

We do this as follows: In Section II, we describe our process of reconstructing the CIC-IDS-2017 dataset into ConCap scenarios and choices we make along the way. In Section III, we experiment with the dataset to show that our reconstruction is faithful and useful. In Section IV, we explore potential improvements to model robustness through adversarial training by augmenting the CIC-IDS-2017 dataset with ConCap traffic.

II. CIC-IDS-2017 reconstruction

The CIC-IDS-2017 dataset consists of traffic captures over the course of a workweek in July 2017, during which a test network is hit with numerous attack classes: Bruteforce, Denial-of-Service, Heartbleed, Web Attacks, Infiltration, Botnet, Distributed Denial-of-Service and Portscan.

We reconstruct the attacks in the dataset by collecting and writing Docker images for the various attackers and targets. By utilizing these images in ConCap scenarios, we can effectively reconstruct the dataset as PCAP files representing the attacks.

A. Monday

No attacks take place on Monday, only benign traffic generated by the B-Profile [20]. This benign traffic is present in all days and serves as background activity in the network. We do not specifically reconstruct this traffic, but rather reuse the traffic from this day in our experiments.

B. Tuesday

FTP and SSH bruteforce attacks take place on Tuesday. These attacks are implemented using Patator [11]. It is unclear what wordlists the authors use to conduct these attacks, we therefore opt to use well-known dictionaries from the SecLists [12] repository.

C. Wednesday

Denial-of-Service and Heartbleed attacks take place on Wednesday. Authors perform attacks with the Slowhttptest and Slowloris [21], GoldenEye [18] and HULK [4] tools, each attacking a different part of the HTTP protocol. We reconstruct these attacks with the respective tools, reusing Slowhttptest to implement both Slowhttptest and SlowLoris attacks, as this tool is capable of both.

Heartbleed bug [8] is also exploited. We follow the authors in using the Heartleech [7] tool to conduct the attack against a vulnerable OpenSSH server version 1.0.1f.

D. Thursday

Web Attacks take place on Thursday, consisting of three subclasses: Bruteforce, SQL-Injection and Cross-Site Scripting. Each of these is conducted against a part of the Damn Vulnerable Web App [26] using Selenium framework. As we do not have access to the original code, we write our own Python scripts to conduct these attacks.

Infiltration attack also takes place on this day, by having the victim download and run a malicious executable, which creates a reverse Meterpreter shell to the attacker, through which a Portscan attack is executed. We opt not to reconstruct this attack for two reasons: First, a lack of support for multi-stage attack execution by ConCap limits our ability to reproduce this attack faithfully. Second, there is little interesting traffic happening on the network: it is the specific file that makes a file transfer malicious, not the act of transferring a file.

E. Friday

On Friday, the remaining attack classes are executed. A Botnet attack is launched using the Ares [27] botnet. We do not include this attack in our reconstruction due to technical limitations: Ares does not provide a way of controlling the botnet from Linux hosts, limiting our ability to conduct an attack from inside Docker containers.

Next, Distributed Denial-of-Service attack is performed using LOIC [14]. Though LOIC is primarily controlled through a GUI, the author does provide a way to control it remotely using an IRC server. We use the multi-target functionality of ConCap to launch both the target and an IRC server. A bot connects to this server and waits for LOIC to make the connection, before directing LOIC to perform the attack by changing the channel topic to the corresponding attack command.

Finally, a Portscan is executed. After traffic analysis, we crafted a container with open ports 21, 22, 80, 139 and 445, pointing to respectively an FTP server, SSH server, HTTP server and an SMB service. The portscan attack is executed against this container. Due to difficulties with setting up Samba, we have decided to conduct SMB portscans separately, merging the traffic with the other portscans during preprocessing.

III. Experimental verification

We first want to verify that the traffic generated by ConCap has similar features to the physical traffic in CIC-IDS-2017. To this end, we preprocess the generated PCAP files into NetFlows using the CICFlowmeter [9]. CICFlowmeter extracts more than eighty features out of the traffic capture, on which we can train ML models. We employ two-way methodology in our verification: Train a model on ConCap traffic and measure its performance on CIC-IDS-2017 traffic, and vice-versa.

To guarantee soundness of our experiments, we utilize the fixed version of CIC-IDS-2017 provided by Engelen et al. [5] as a stand-in for the actual dataset provided by Sharafaldin et al., where numerous labeling mistakes are fixed, chief among which incorrectly implemented attacks.

For model training, we use a Decision Tree with a single root node, utilizing a single feature to make the decision on, and Gini criterion. After preprocessing the datasets, we add equal amount of benign NetFlows to the attacks NetFlows to achieve class balance. We take these benign NetFlows from CIC-IDS-2017 Monday. Afterwards, we train the model on each feature and measure its accuracy, precision and recall, as well as ROC AUC score.

We are looking for a set of features that show high predictive power in both ways. While preferable, we do not expect the set of best performing features to be the same both ways, due to the underlying networks being fundamentally different: the ConCap network is a theoretically perfect network implemented fully in-silico. Though ConCap does provide a way to configure network artifacts (packet drops, corruption, delays, reordering...), we lack this information about the CIC-IDS-2017 network, making it impossible to reproduce.

These experiments show the presence of these common features with high predictive power as measured by the ROC AUC score. Naturally, different attacks require different features for detection. In Table I, we present the top three features for each attack class found according to their average ROC AUC Score for the two ways of experimenting.

Overall, we find plenty of features that perform well for both ways of training, regularly scoring above .9 on the ROC AUC score and none going below .8. It must be noted that due to low amount of samples of some classes in CIC-IDS-2017 (Heartbleed and Web attacks), models of these attack classes show signs of overfitting. Nevertheless, we have shown that the ConCap framework can be reliably used to replicate datasets and potentially extend them, as the models based on either the dataset or the ConCap traffic can reliably predict the other.

IV. Dataset Augmentation

Model robustness refers to a model's ability to maintain performance over unseen samples or over variations of previously seen samples. ConCap can easily be used for increasing model robustness, by generating traffic of variations of attacks. After verifying ConCap's usefulness as a stand-in for the CIC-IDS-2017 dataset in the previous section, we perform additional experiments to see the effect of ConCap on the robustness of datasets. More specifically, we are interested in the effect of dataset augmentation through adversarial training.

We augment the existing dataset with scenarios that try out different options of the existing tools. More specifically:

TABLE I
VERIFICATION RESULTS

Attack class	Feature	ROC-AUC Score
FTP Bruteforce	Bwd RST Flags	0.978288
	Packet Length Mean	0.976290
	Average Packet Size	0.976290
SSH Bruteforce	Fwd Seg Size Min	0.922024
	Fwd IAT Min	0.854907
	Bwd Segment Size Avg	0.817511
DoS Slowloris	Total TCP Flow Time	0.910773
	Bwd Packet Length Max	0.868125
	Total Length of Bwd Packet	0.868125
DoS Slowhttptest	Total TCP Flow Time	0.905697
	Fwd IAT Min	0.875804
	Fwd IAT Total	0.842901
DoS GoldenEye	Bwd Packet Length Std	0.939286
	Packet Length Variance	0.927954
	Packet Length Std	0.927954
DoS HULK	Bwd Packet Length Std	0.977942
	Fwd RST Flags	0.954837
	Subflow Bwd Bytes	0.951484
Heartbleed	Bwd Packet Length Std	1.000000
	Flow Bytes/s	0.931818
	Packet Length Std	0.905702
Web Attack Bruteforce	Fwd Seg Size Min	0.934932
	Fwd IAT Min	0.891781
	FIN Flag Count	0.828767
Web SQL Injection	Fwd Seg Size Min	0.884615
	FIN Flag Count	0.865385
	Bwd IAT Min	0.816719
Web XSS	Fwd Seg Size Min	0.930556
	Bwd Packet Length Std	0.869792
	Packet Length Max	0.845486
DDoS LOIC	Fwd Seg Size Min	0.934932
	Fwd IAT Min	0.860959
	FIN Flag Count	0.821918
Portscan	Fwd Packet Length Max	0.949676
	Total Length of Fwd Packet	0.948696
	Fwd Segment Size Avg	0.947596

- **FTP Bruteforce**: Turn persistence off, forcing one attempt per connection
- **DoS GoldenEye**: Attack with the POST HTTP verb
- **DDoS LOIC**: Attack over UDP

For each of these extensions, we generate the adversarial traffic and train a model using the CIC-IDS-2017 traffic on the best feature for the respective class as found above. Afterwards, we measure the model's performance on the generated adversarial traffic using ROC AUC score, forming our baseline. Next, we prepare a mix of CIC-IDS-2017 traffic and adversarial traffic, retrain the model and measure its performance. Just like in the experiments in previous section, we balance the training and testing sets with random samples of benign traffic from Monday traffic capture of CIC-IDS-2017.

In Table II, we report the average ROC AUC Scores for the different extensions across ten runs. All models perform poorly at first, having less than 50% probability of predicting a random malicious sample as malicious. This probability increases substantially after adversarial training. Our results show that ConCap can be effectively used for robustness enhancements of ML-NIDS models.

TABLE II
AVERAGE ROC AUC SCORES FOR DIFFERENT ATTACKS

Attack class	Baseline	Adversarial	Difference
FTP no persistence	0.4982	0.59236	0.09416
GoldenEye POST	0.48923	0.84631	0.35708
LOIC UDP	0.30098	0.78868	0.4877

TABLE III
DATASET AUGMENTATION: BASELINE + ADVERSARIAL ROC AUC SCORES ON CIC-IDS-2017

Attack class	Baseline	Adversarial	Difference
FTP No Persistence	0.99577	0.99615	0.00038
GoldenEye POST	0.98522	0.84716	-0.13807
LOIC UDP	0.78831	0.78879	0.00048

Lastly, we do a sanity check experiment for the model performance on original samples after adversarial training. The adversarial training would be of little value if the model performance on the original dataset degrades significantly. This is done by using the test split of CIC-IDS-2017 traffic and having the adversarial model predict it. Table III presents our findings.

We see no significant drop in performance for the FTP No Persistence and LOIC UDP models, but we do notice a degradation for the GoldenEye POST model. We posit that this is caused by this adversarial change being more significant than the other two classes. Nevertheless, the models maintains its predictive power with ROC AUC Score above 0.8, from which we conclude that the model remains usable.

V. CONCLUSION

This article presents a case study of a new method of robustness enhancement of ML-NIDS systems through dataset augmentation.

After an analysis of CIC-IDS-2017, we use ConCap to reconstruct the attacks in this dataset. Using Decision Trees, we have experimentally verified that the produced traffic can be used as a substitute for the dataset, with models achieving high levels of accuracy.

Finally, we have constructed three variations of existing attacks and shown that the models can be adversarially trained to recognize these new attacks.

VI. FUTURE WORK

Building on this article, further research can look into supporting more elaborate network architectures and scenarios, as ConCap is quite limited in this regard. Furthermore, ConCap can be utilized as a simulation environment, providing an

excellent environment for blue team training and practice, as well as new attack modelling for red teams. Finally, real-time extensions to the ML-NIDS systems can prove useful to not only construct Intrusion Detection Systems, but also Intrusion *Prevention* Systems, stopping attacks as they are happening.

References

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

[2] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. "SoK: Pragmatic assessment of machine Learning for Network Intrusion Detection". In: (2023). eprint: 2305.00550 (cs.CR).

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

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

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

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

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

[8] *Heartbleed*. URL: https://heartbleed.com/.

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

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

[11] Sebastien Macke. https://github.com/lanjelot/patator.

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

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

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

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

[16] Martin Roesch. "Snort - Lightweight Intrusion Detection for Networks". In: *Proceedings of the 13th USENIX Conference on System Administration*. LISA '99. Seattle, Washington: USENIX Association, 1999, 229–238.

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

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

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

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

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

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

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

[24] *Suricata*. 2025. URL: https://suricata.io/.

[25] Geert-Jan (ugent)01802387 Van Nieuwenhove. *Genereren van DDoS aanvallen voor het creëren van een dataset.* und. 2023. URL: http://lib.ugent.be/catalog/rug01:003150617.

[26] Robin Wood. *digininja / DVWA*. 2025. URL: https://github.com/digininja/DVWA.

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

[28] zeek. *zeek / zeek*. 2025. URL: https://github.com/zeek/zeek.

Verbetering van de Robuustheid van Netwerkintrusiedetectie via Data-Augmentatie: een Onderzoek van CIC-IDS-2017

Jozef Jankaj, Miel Verkerken, Laurens D'hooge, Tim Wauters, Bruno Volckaert
IDLab, Department of Information Technology at Ghent University - imec, Ghent, Belgium
{jozef.jankaj, miel.verkerken, laurens.dhooge, tim.wauters, bruno.volckaert}@UGent.be

***Abstract*—Naarmate onze wereld steeds meer met elkaar verbonden raakt, worden onze netwerken bedreigd vanuit meer bronnen dan ooit tevoren. Bestaande beveiligingsmaatregelen hebben moeite om gelijke tred te houden met de snel evoluerende dreigingen in de cyberwereld, wat snelle verbeteringen noodzakelijk maakt. Een mogelijke weg naar verbetering is het gebruik van op machine learning gebaseerde netwerk-inbraakdetectiesystemen (ML-NIDS), die geautomatiseerde bescherming aan netwerken bieden. De bruikbaarheid van deze modellen neemt echter snel af naarmate nieuwe vormen van aanvallen worden ontdekt, en hun robuustheid is twijfelachtig bij het omgaan met variaties van aanvallen. In dit artikel verkennen we een nieuwe methode om de robuustheid van NIDS-modellen te vergroten door middel van datasetverrijking met behulp van ConCap.**

I. Introductie

Met de opkomst van het internet zijn fysieke bedreigingen voor onze veiligheid overgeslagen naar de virtuele wereld. Onze privacy wordt voortdurend bedreigd naarmate we er steeds meer van opgeven ten behoeve van het gemak dat de onderling verbonden wereld biedt. Tal van kwaadwillenden zijn maar al te bereid om aan te vallen, te stelen en onze systemen te vernietigen. Om deze dreiging, als een zwaard van Damocles, het hoofd te bieden, is netwerkbeveiliging een belangrijk onderwerp van onderzoek geworden. Hoewel er aanzienlijke vooruitgang is geboekt in netwerkbeveiliging dankzij verbeteringen in berichtversleuteling, de implementatie van firewalls en algemene beveiligingspraktijken, blijven aanvallen op onze netwerken frequent voorkomen. Tijdige detectie en onderbreking van deze aanvallen is van cruciaal belang om de vertrouwelijkheid, integriteit en beschikbaarheid van onze netwerken te waarborgen.

Een Netwerk Intrusie Detectie Systeem (NIDS) dient dit doel door het netwerkverkeer te monitoren en alarm te slaan bij de detectie van kwaadaardig verkeer. Bestaande systemen [16, 24, 28] worden geïmplementeerd op basis van voorgeprogrammeerde regels en kenmerken van het verkeer. De snelle evolutie van cybersecurity maakt deze systemen echter al snel ineffectief, omdat voortdurend nieuwe aanvallen en varianten van bestaande aanvallen worden ontdekt en uitgevoerd. Om deze en andere redenen hebben onderzoekers hun aandacht gericht op op Machine Learning gebaseerde NIDS systemen (ML-NIDS). Zowel klassieke, shallow-learning methoden [10, 17, 3] als deep learning-methoden [15, 23] zijn eerder in de literatuur onderzocht. Deze methoden zijn sterk afhankelijk van de kwaliteit van de data waarop deze modellen getrained worden, en daarom zijn er diverse datasets [19, 1, 13, 22] ontwikkeld en voorgesteld als benchmark voor ML-NIDS-onderzoek.

Helaas is gebleken dat deze datasets fouten bevatten [5, 6], wat hun bruikbaarheid vermindert. Problemen zoals gebrekkige documentatie, foutieve implementaties van aanvallen en onjuiste labeling zijn aan het licht gekomen, met als resultaat enige terughoudendheid van netwerkbeheerders in het implementeren van deze modellen [2].

Om deze en andere knelpunten aan te pakken, is het ConCap-framework [25] ontwikkeld en ingezet. Dit maakt gebruik van de kracht van Kubernetes-clusters om netwerken in-silico op te bouwen, aanvallen erop uit te voeren en verkeer vast te leggen ten behoeve van datasetcreatie. In dit artikel onderzoeken wij het gebruik van deze technologie om de robuustheid van bestaande ML-NIDS-datasets te verbeteren via datasetverrijking, waarbij de CIC-IDS-2017-dataset [19] als casestudy wordt gebruikt.

We doen dit als volgt: in Sectie II beschrijven we ons proces van het reconstrueren van de CIC-IDS-2017-dataset in ConCap-scenario's en de keuzes die we daarbij maken. In Sectie III voeren we experimenten uit met de dataset om aan te tonen dat onze reconstructie betrouwbaar en bruikbaar is. In Sectie IV verkennen we mogelijke verbeteringen in modelrobuustheid via adversarial training door de CIC-IDS-2017-dataset te verrijken met ConCap-verkeer.

II. CIC-IDS-2017 reconstructie

De CIC-IDS-2017-dataset bestaat uit verkeersopnames gedurende een werkweek in juli 2017, waarin een testnetwerk wordt getroffen door verschillende soorten aanvallen: Brute-force, Denial-of-Service, Heartbleed, Webaanvallen, Infiltratie, Botnet, Distributed Denial-of-Service en Portscan.

Wij reconstrueren de aanvallen in de dataset door Docker-images te verzamelen en te schrijven voor de verschillende aanvallers en doelwitten. Door deze images te gebruiken in ConCap-scenario's kunnen we de dataset effectief reconstrueren als PCAP-bestanden die de aanvallen representeren.

A. Maandag

Op maandag vinden er geen aanvallen plaats; er is uitsluitend onschadelijke verkeer aanwezig dat is gegenereerd door

het B-Profile [20]. Dit onschadelijke verkeer komt op alle dagen voor en fungeert als achtergrondactiviteit binnen het netwerk. Wij reconstrueren dit verkeer niet afzonderlijk, maar hergebruiken het verkeer van deze dag in onze experimenten.

B. Dinsdag

FTP- en SSH-bruteforceaanvallen vinden plaats op dinsdag. Deze aanvallen worden geïmplementeerd met behulp van Patator [11]. Het is onduidelijk welke woordenlijsten de auteurs gebruiken om deze aanvallen uit te voeren; daarom kiezen wij ervoor om bekende woordenlijsten uit de SecLists-repository [12] te gebruiken.

C. Woensdag

Op woensdag vinden Denial-of-Service- en Heartbleed-aanvallen plaats. De auteurs voeren deze aanvallen uit met de tools Slowhttptest en Slowloris [21], GoldenEye [18] en HULK [4], waarbij elk een ander deel van het HTTP-protocol aanvalt. Wij reconstrueren deze aanvallen met de respectieve tools, waarbij we Slowhttptest hergebruiken om zowel de Slowhttptest- als de Slowloris-aanvallen te implementeren, aangezien deze tool beide ondersteunt.

Daarnaast wordt ook misbruik gemaakt van de Heartbleed-bug [8]. Wij volgen de auteurs in het gebruik van de Heartleech-tool [7] om de aanval uit te voeren op een kwetsbare versie 1.0.1f van de OpenSSH-server.

D. Donderdag

Webaanvallen vinden plaats op donderdag en bestaan uit drie subklassen: Bruteforce, SQL-Injection en Cross-Site Scripting. Elk van deze aanvallen wordt uitgevoerd op een onderdeel van de Damn Vulnerable Web App [26], met behulp van het Selenium-framework. Aangezien we geen toegang hebben tot de oorspronkelijke code, schrijven we onze eigen Python-scripts om deze aanvallen uit te voeren.

Op deze dag vindt ook een infiltratieaanval plaats, waarbij het slachtoffer een kwaadaardig programma downloadt en uitvoert. Dit bestand creëert een omgekeerde Meterpreter-shell naar de aanvaller, via welke een Portscan-aanval wordt uitgevoerd. Wij kiezen ervoor om deze aanval niet te reconstrueren, om twee redenen: Ten eerste beperkt het gebrek aan ondersteuning voor meerfasige aanvalsexecutie binnen ConCap onze mogelijkheid om deze aanval getrouw te reproduceren. Ten tweede vindt er weinig interessant netwerkverkeer plaats: het is het specifieke bestand dat een bestandsoverdracht kwaadaardig maakt, niet de handeling van de overdracht zelf.

E. Vrijdag

Op vrijdag worden de resterende aanvalsklassen uitgevoerd. Een Botnet-aanval wordt uitgevoerd met behulp van het Aresbotnet [27]. Wij nemen deze aanval niet op in onze reconstructie vanwege technische beperkingen: Ares biedt geen mogelijkheid om het botnet aan te sturen vanaf Linux-hosts, wat onze mogelijkheden om de aanval vanuit Docker-containers uit te voeren beperkt.

Vervolgens wordt een Distributed Denial-of-Service-aanval uitgevoerd met LOIC [14]. Hoewel LOIC primair via een grafische interface wordt aangestuurd, biedt de auteur een mogelijkheid om het op afstand te aansturen via een IRC-server. Wij gebruiken de multi-target-functionaliteit van ConCap om zowel het doelwit als een IRC-server te starten. Een bot maakt verbinding met deze server en wacht tot LOIC verbindt, waarna LOIC wordt aangestuurd om de aanval uit te voeren door het kanaalonderwerp aan te passen naar het bijbehorende aanvalcommando.

Tot slot wordt een Portscan uitgevoerd. Na analyse van het netwerkverkeer hebben wij een container samengesteld met open poorten 21, 22, 80, 139 en 445, respectievelijk wijzend naar een FTP-server, SSH-server, HTTP-server en een SMB-dienst. De portscan-aanval wordt op deze container uitgevoerd. Vanwege moeilijkheden bij het opzetten van SMB hebben wij ervoor gekozen de SMB-portscans afzonderlijk uit te voeren en het verkeer tijdens de preprocessingfase samen te voegen met de overige portscans.

III. Experimentele verificatie

We willen eerst verifiëren of het verkeer dat door ConCap wordt gegenereerd vergelijkbare kenmerken vertoont met het fysieke verkeer in CIC-IDS-2017. Daartoe preprocessen we de gegenereerde PCAP-bestanden naar NetFlows met behulp van CICFlowmeter [9]. CICFlowmeter extraheert meer dan tachtig kenmerken uit de verkeersopname, waarop we ML-modellen kunnen trainen. We hanteren een tweezijdige methodologie voor onze verificatie: we trainen een model op ConCap-verkeer en meten de prestaties op CIC-IDS-2017-verkeer, en omgekeerd.

Om de betrouwbaarheid van onze experimenten te waarborgen, maken wij gebruik van de gecorrigeerde versie van CIC-IDS-2017, aangeboden door Engelen et al. [5], als vervanging voor de oorspronkelijke dataset van Sharafaldin et al., waarin diverse labelingsfouten zijn hersteld, met name verkeerd geïmplementeerde aanvallen.

Voor het trainen van het model gebruiken we een beslissingsboom met een enkele root, die een enkele feature gebruikt om de beslissing op te baseren, met het Gini-criterium. Na het preprocessen van de datasets voegen we een gelijke hoeveelheid goedaardige NetFlows toe aan de aanval-NetFlows om klassebalans te bereiken. Deze goedaardige NetFlows nemen we van het verkeer van maandag in de CIC-IDS-2017, zoals hierboven al aangehaald. Vervolgens trainen we het model op elke feature afzonderlijk en meten we de nauwkeurigheid, precisie en recall, evenals de ROC AUC-score.

We zijn op zoek naar een set features met een hoge voorspellende waarde in beide richtingen. Hoewel het wenselijk is, verwachten we niet dat de best presterende features aan beide zijden identiek zijn, omdat de onderliggende netwerken fundamenteel verschillend zijn: het ConCap-netwerk is een theoretisch perfect netwerk dat volledig in-silico is geïmplementeerd. Hoewel ConCap de mogelijkheid biedt om netwerkartefacten te configureren (pakketverlies, corruptie, vertragingen, herschikkingen...), ontbreekt deze informatie voor het CIC-IDS-2017-netwerk, waardoor reproductie onmogelijk is.

TABLE I
VERIFICATION RESULTS

Attack class	Feature	ROC-AUC Score
FTP Bruteforce	Bwd RST Flags	0.978288
	Packet Length Mean	0.976290
	Average Packet Size	0.976290
SSH Bruteforce	Fwd Seg Size Min	0.922024
	Fwd IAT Min	0.854907
	Bwd Segment Size Avg	0.817511
DoS Slowloris	Total TCP Flow Time	0.910773
	Bwd Packet Length Max	0.868125
	Total Length of Bwd Packet	0.868125
DoS Slowhttptest	Total TCP Flow Time	0.905697
	Fwd IAT Min	0.875804
	Fwd IAT Total	0.842901
DoS GoldenEye	Bwd Packet Length Std	0.939286
	Packet Length Variance	0.927954
	Packet Length Std	0.927954
DoS HULK	Bwd Packet Length Std	0.977942
	Fwd RST Flags	0.954837
	Subflow Bwd Bytes	0.951484
Heartbleed	Bwd Packet Length Std	1.000000
	Flow Bytes/s	0.931818
	Packet Length Std	0.905702
Web Attack Bruteforce	Fwd Seg Size Min	0.934932
	Fwd IAT Min	0.891781
	FIN Flag Count	0.828767
Web SQL Injection	Fwd Seg Size Min	0.884615
	FIN Flag Count	0.865385
	Bwd IAT Min	0.816719
Web XSS	Fwd Seg Size Min	0.930556
	Bwd Packet Length Std	0.869792
	Packet Length Max	0.845486
DDoS LOIC	Fwd Seg Size Min	0.934932
	Fwd IAT Min	0.860959
	FIN Flag Count	0.821918
Portscan	Fwd Packet Length Max	0.949676
	Total Length of Fwd Packet	0.948696
	Fwd Segment Size Avg	0.947596

Deze experimenten tonen de aanwezigheid van deze gemeenschappelijke kenmerken met een hoge voorspellende waarde, gemeten aan de hand van de ROC AUC-score. Uiteraard vereisen verschillende aanvallen verschillende kenmerken voor detectie. In Tabel I presenteren we de drie belangrijkste kenmerken voor elke aanvalsklasse, bepaald op basis van hun gemiddelde ROC AUC-score voor de twee experimenten.

Over het algemeen vinden we veel kenmerken die goed presteren bij beide trainingsmethoden, met regelmatig scores boven de 0,9 op de ROC AUC-score en geen enkele onder de 0,8. Het moet worden opgemerkt dat, door het geringe aantal voorbeelden van sommige klassen in CIC-IDS-2017 (Heartbleed- en Web-aanvallen), modellen van deze aanvalsklassen tekenen van overfitting vertonen. Niettemin hebben we aangetoond dat het ConCap-framework betrouwbaar kan worden gebruikt om datasets te repliceren en mogelijk uit te breiden, aangezien de modellen die zijn gebaseerd op ofwel de originele dataset ofwel het ConCap-verkeer, de andere betrouwbaar kunnen voorspellen.

TABLE II
AVERAGE ROC AUC SCORES FOR DIFFERENT ATTACKS

Attack class	Baseline	Adversarial	Difference
FTP no persistence	0.4982	0.59236	0.09416
GoldenEye POST	0.48923	0.84631	0.35708
LOIC UDP	0.30098	0.78868	0.4877

IV. DATASETAUGMENTATIE

Modelrobustheid verwijst naar het vermogen van een model om prestaties te behouden bij ongeziene voorbeelden of variaties van de eerder geziene voorbeelden. ConCap kan eenvoudig worden ingezet om de modelrobustheid te vergroten door verkeer te genereren van variaties op aanvallen.

Na het verifiëren van het nut van ConCap als vervanger voor de CIC-IDS-2017 dataset in de vorige sectie, voeren we aanvullende experimenten uit om het effect van ConCap op de robuustheid van datasets te onderzoeken. We zijn met name geïnteresseerd in het effect van datasetaugmentatie door middel van adversariële training.

We breiden de bestaande dataset uit met scenario's die verschillende opties van de bestaande tools testen. Meer specifiek:

- **FTP Bruteforce**: Persistence uitzetten, waardoor er per verbinding slechts één poging wordt gedaan
- **DoS GoldenEye**: Aanval met de POST HTTP-verb
- **DDoS LOIC**: Aanval via UDP

Voor elk van deze uitbreidingen genereren we het adversariële verkeer en trainen we een model op het beste kenmerk voor de betreffende klasse, zoals hierboven vastgesteld, gebruikmakend van het CIC-IDS-2017-verkeer. Vervolgens meten we de prestatie van het model op het gegenereerde adversariële verkeer met behulp van de ROC AUC-score, wat onze basislijn vormt. Daarna stellen we een combinatie samen van CIC-IDS-2017-verkeer en adversariële verkeer, hertrainen we het model en meten we opnieuw de prestaties. Net als in de experimenten in de vorige sectie, balanceren we de trainings- en testsets met willekeurige samples van goedaardig verkeer uit de maandagse verkeerstrace van CIC-IDS-2017.

In Tabel II rapporteren we de gemiddelde ROC AUC-scores voor de verschillende extensies over tien runs. Alle modellen presteren aanvankelijk slecht, met minder dan 50% kans om een willekeurig kwaadaardig sample correct als kwaadaardig te classificeren. Deze kans neemt aanzienlijk toe na adversariële training. Onze resultaten tonen aan dat ConCap effectief kan worden ingezet voor het vergroten van de robuustheid van ML-NIDS-modellen.

Tot slot voeren we een sanity check-experiment uit om de modelprestaties op originele samples na adversariële training te evalueren. De adversariële training zou van weinig meerwaarde zijn als de modelprestaties op de originele dataset erdoor significant verslechteren. Dit wordt getest door gebruik

TABLE III
DATASET AUGMENTATION: BASELINE + ADVERSARIAL ROC AUC SCORES ON CIC-IDS-2017

Attack class	Baseline	Adversarial	Difference
FTP No Persistence	0.99577	0.99615	0.00038
GoldenEye POST	0.98522	0.84716	-0.13807
LOIC UDP	0.78831	0.78879	0.00048

te maken van de test-split van het CIC-IDS-2017-verkeer, waarbij het adversariële model deze samples voorspelt. Tabel III toont onze bevindingen.

We zien geen significante prestatievermindering bij de modellen voor FTP No Persistence en LOIC UDP, maar we merken wel een degradatie op bij het GoldenEye POST-model. Wij vermoeden dat dit komt doordat de adversariële wijziging bij deze klasse ingrijpender is dan bij de andere twee. Desondanks behoudt het model zijn voorspellende kracht met een ROC AUC-score boven de 0,8, waaruit we concluderen dat het model bruikbaar blijft.

V. CONCLUSIE

Dit artikel presenteert een casestudy van een nieuwe methode voor het verbeteren van de robuustheid van ML-NIDS-systemen door middel van datasetaugmentatie.

Na een analyse van CIC-IDS-2017 hebben we ConCap gebruikt om de aanvallen in deze dataset te reconstrueren. Met behulp van beslissingsbomen hebben we experimenteel aangetoond dat het gegenereerde verkeer als vervanging van de dataset kan dienen, waarbij modellen hoge nauwkeurigheid bereiken.

Tot slot hebben we drie variaties van bestaande aanvallen geconstrueerd en aangetoond dat de modellen adversariëel getraind kunnen worden om deze nieuwe aanvallen te herkennen.

VI. TOEKOMSTIG ONDERZOEK

Voortbouwend op dit artikel kan vervolgonderzoek zich richten op het ondersteunen van uitgebreidere netwerkarchitecturen en scenario's, aangezien ConCap op dit vlak nog vrij beperkt is. Daarnaast kan ConCap worden ingezet als simulatieomgeving, wat een uitstekende basis biedt voor training en oefening van blue teams, evenals voor het modelleren van nieuwe aanvallen door red teams. Tot slot kunnen real-time uitbreidingen van ML-NIDS-systemen van waarde zijn, niet alleen voor het opbouwen van Intrusie Detectie Systemen, maar ook van Intrusie *Preventie* Systemen, die aanvallen in real-time kunnen stoppen.

REFERENTIES

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

[2] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. "SoK: Pragmatic assessment of machine Learning for Network Intrusion Detection". In: (2023). eprint: 2305.00550 (cs.CR).

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

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

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

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

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

[8] *Heartbleed*. URL: https://heartbleed.com/.

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

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

[11] Sebastien Macke. https://github.com/lanjelot/patator.

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

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

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

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

[16] Martin Roesch. "Snort - Lightweight Intrusion Detection for Networks". In: *Proceedings of the 13th USENIX Conference on System Administration*. LISA

'99. Seattle, Washington: USENIX Association, 1999, 229–238.

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

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

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

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

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

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

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

[24] *Suricata*. 2025. URL: https://suricata.io/.

[25] Geert-Jan (ugent)01802387 Van Nieuwenhove. *Genereren van DDoS aanvallen voor het creëren van een dataset.* und. 2023. URL: http://lib.ugent.be/catalog/rug01:003150617.

[26] Robin Wood. *digininja / DVWA*. 2025. URL: https://github.com/digininja/DVWA.

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

[28] zeek. *zeek / zeek*. 2025. URL: https://github.com/zeek/zeek.

Dankwoord

Het heeft lang geduurd, maar we zijn er geraakt. Na jarenlange inzet, doorheen corona en andere obstakels, sta ik eindelijk aan het einde van mijn opleiding. En wat voor een rit het geweest is!

Deze thesis is het resultaat van een passie die ik tijdens deze opleiding heb ontdekt, een eindeloze nieuwsgierigheid en de steun van de mensen rondom mij. Eerst en vooral wil ik mijn oprechte dank uitspreken aan prof. dr. Bruno Volckaert voor zijn vele wijze lessen, en aan Miel Verkerken en Laurens D'hooge voor hun begeleiding doorheen het voorbije jaar.

Daarnaast wil ik ook mijn medestudenten, vrienden en familie bedanken voor hun steun en betrokkenheid. De talloze keren dat ik mijn problemen met mijn ouders besprak, hielpen mij telkens weer om tot oplossingen te komen.

Tot slot wil ik mijn verloofde Aliaksandra bedanken. Haar steun en goede raad zijn doorheen de jaren van onschatbare waarde geweest.

Toelating tot bruikleen

De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef.

Jozef Jankaj
1 juni 2025

Contents

1 Introduction

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

> "Then we typed the G, and the system crashed... Yet a revolution had begun" [13]

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

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

The aim of this work is to address this complaint by studying a method of enhancing robustness of NIDS datasets through dataset augmentation. We take the CIC-2017 dataset [29] as a case study, created by the Canadian Institute for Cybersecurity.

This work is structured as follows: In Section 2, we explore the literature surrounding ML-NIDS, illustrating current methods and datasets used in the field. In Section 3, we formulate a hypothesis behind the poor generalization performance of ML-NIDS models, explore the CIC-IDS-2017 dataset in-depth and recreate it using ConCap [34]. In Section 4, we experimentally verify the usefulness of our reconstruction through ConCap by training an ML model. Next, in Section 5, we explore a robustness enhancement of the ML model through adversarial training. Finally, in Section 6, we summarize our findings and in Section 7 we reflect on further research avenues.

2 Literature study

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

One of the first implementations of a Network Intrusion Detection System (NIDS) has been NADIR (Network Anomaly Detection and Intrusion Reporter) [15] from 1990. This system was developed for the Integrated Computing Network at Los Alamos National Laboratory to automate the detection of unauthorized access to the system. NADIR utilized statistical profiles of network users to detect anomalies, notifying and printing out abnormalities in the profiles as detected by manually designed expert rules.

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

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

In general, ML-NIDS can be split into two subgroups: signature-based and anomaly-based. Both subgroups fulfill the same task (tell malicious and benign traffic apart), but approach it in fundamentally opposite ways. Signature-based ML-NIDS is trained to recognise an attack, or even the specific kind of attack in multi-class classification: the "positive" label is the malicious traffic. On the contrary, anomaly-based ML-NIDS learns to recognise what benign traffic looks like, and triggers and alarm at any traffic that is "out of the ordinary".

2.1 Machine Learning methods

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

2.1.1 Shallow Learning

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

1. K-Nearest Neighbours (KNN) [6]: Initially developed by Fix and Hodges in 1951 [10] and later improved by Cover and Hart in 1967 [6], these models can be used for multiclass classification. Predictions are formed by a majority vote, where the prediction is formed by the label of samples most similar to the input.

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

3. Decision Trees (DT) [36]: Decision trees form a tree data structure, with each internal node testing for a particular feature in the data, reaching a class label in one of the leaf nodes.

4. Ensemble methods [24]: Ensemble methods combine multiple simpler classifiers to form a larger one. This way, weaknesses of the underlying classifier (e.g. an SVM being unable to separate multiple classes) can be overcome (a separate SVM being trained for each class label, as described above).

Liao and Vemuri [18] use a KNN classifier for intrusion detection by building a profile of the input process consisting of the occurrences of system calls the process performs.

Sahu and Mehtre [27] implement the J48 Decision Tree, testing it on Kyoto 2006+ dataset [9]. Their trained model boasts 97.23% accuracy and a true positive rate of 99% for normal and attack packets.

Bao, Xu and Hou [3] build an NIDS system consisting of two SVM models: one for anomaly detection, the other for misuse detection using previously seen attack signatures. Authors do make claims about "high training rate and decision rate, insensitiveness to dimensions of input data, continuous correction

of various parameters with increase in training data which endows the system with self-learning ability, and so on.", yet they do not support this with figures, measurements or experiment setup.

2.1.2 Deep Learning methods

Deep Learning methods are a subset of machine learning methods that utilize neural networks for learning. Neural networks consist of artificial neurons: nodes that have a particular activation function. Similarly to biological systems, these nodes are connected together to form layers. Inputs propagate through the layers and result in some activation in the final layer [37]. For the purposes of classification, this final layer could, for example, consist of one neuron for each classification class.

Depending on the network topology and the layers utilized, different models arise [17]:

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

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

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

Song et al. [33] perform an analysis of autoencoders in ML-NIDS, where an autoencoder is trained to recognize normal behaviour. Any input that the autoencoder fails to recreate below a particular error margin is classified as malicious.

Pham et al. [25] propose a six-layer 1D CNN model, consisting of four convolutional layers and two fully connected layers. They compare this model with different Shallow Learning models (Gaussian Naïve Bayes, Logistic Regression, KNN, SVM, AdaBoost, Gradient Boosting, XGBoost, CatBoost, LightGBM)

and show that their CNN model outperforms them on the UNSW-NB15[21] and NSL-KDD [38] datasets, achieving 99.3% and 99.43% accuracy respectively.

2.2 Dataset quality

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

- CIC-IDS-2017/CSE-CIC-IDS2018 [29, 1] (4623 citations): CIC-IDS-2017 dataset and its cousin CSE-CIC-2018 are all-purpose NIDS datasets, comprising of seven different attack classes executed over the course of multiple days. CIC-IDS-2017 is the base, comprising 14 hosts, while CSE-CIC-2018 is much larger in scale, being implemented on Amazon Web Services as a simulation of a realistic company network: fifty attacker machines, 420 victim machines spread over five departments and thirty servers, 500 machines in total.

- ISCX IDS 2012 [32] (1558 citations): A predecessor to CIC-IDS-2017, ISCX IDS 2012 implements its attacks as overlapping "scenarios" defined as unambiguous α and β profiles, focusing on HTTP, SMTP/IMAP, FTP and SSH traffic.

- UNSW-NB15 [21] (4001 citations): UNSW-NB15 dataset is built using the IXIA attack generator on a testbed consisting of three virtual servers, two for normal traffic and one for attacks.

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

Similarly, Flood et al. [11] found issues with UNSW-NB15, concluding that "without modification, UNSW NB15 is unsuitable for evaluating the ability of classifiers to generalise between attack categories". They, too, note lacking documentation in other datasets (ToN_IoT, UNSW-NB15 and CIC-IDS-2017/CSE-CIC-18).

In their pragmatic assessment of the field, Apruzzese et al. [2] note the skepticism of professional ML-NIDS practitioners to use these "highly successful" datasets, with the most common complaint being the lack of proof of generalization performance and robustness: "It works in *your network*. But will it work equally well in *my network*, and is it *affordable* (now and in the long-term)?".

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

2.3 ConCap

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

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

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

The ConCap tool by Verkerken et al. [34] has been proposed as a new method for ML-NIDS dataset generation. ConCap generates traffic from scenario files using a Kubernetes cluster, in which the targets are specified as Docker containers. This method solves both abovementioned limitations at once: reproducibility is guaranteed, as the dataset does not only consist of raw traffic capture files, but primarily of scenario files. Furthermore, the dataset can be easily extended with new attack classes or variants of existing attack classes, down to individual options with which the attack tools are configured. This makes the dataset both self-documenting and reproducible.

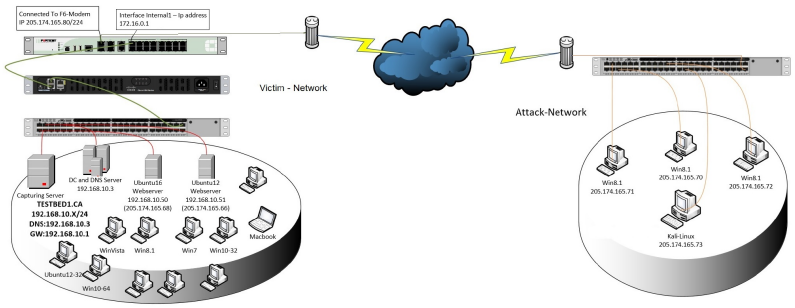

Figure 1: Architecture of CIC-IDS-2017 testbed

3 Dataset Reconstruction

In order to study enhancements to model robustness, we need to first have the dataset reconstructed down to the individual attacks. To this end, we use the ConCap framework and reconstruct the CIC-IDS-2017 dataset into ConCap scenarios. We do this as follows: In Section 3.1, we analyse the CIC-IDS-2017 and the literature surrounding it in depth. In Section 3.2, we describe the reconstruction process and any significant choices we make.

3.1 Analysis of CIC-IDS-2017

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

The dataset contains a relatively small victim network, consisting of a Windows Server, a Ubuntu 16 webserver and a Ubuntu 12 server, as well as two Ubuntu 14.04, two Ubuntu 16.04, a Windows 7, Windows 8.1-64, Windows Vista, Windows 10 Pro and a Macintosh computers, nine in total. The attacker network consists of one Kali Linux and three Windows 8.1 machines. The attackers communicate with the victim network over the internet, appearing in the PCAP files under the IP address `172.16.0.1`. There is further network infrastructure present, but it is inconsequential for our purposes. Figure 1 [29] shows a schematic of the testbed.

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

3.1.1 Monday

On Monday, no attacks take place, only background traffic is happening. The dataset authors utilise a B-Profile system [30] for benign traffic generation "to profile abstract behavior of human interactions and generates naturalistic background traffic"[29]. The traffic capture of this day consists of "abstract behaviour of 25 users based on the HTTP, HTTPS, FTP, SSH, and email protocols." [29] and is an excellent source of benign samples that we later use in our experiments.

3.1.2 Tuesday

On Tuesday, in addition to the background traffic as seen on Monday, we see the first attack class take place: Bruteforce. In the morning, an FTP Bruteforce attack is performed using Patator [19]. This attack is run against the iscxtap user, with various passwords being tried from an unknown wordlist. In the afternoon, SSH Bruteforce attack is performed using presumably the same tool. Due to inherent encryption of SSH traffic, it is unclear against which user the attack is conducted or what wordlist is used for passwords.

3.1.3 Wednesday

On Wednesday, Denial-of-Service (DoS) attacks are launched against the Ubuntu 16 webserver. The dataset consists of four DoS attacks conducted with the tools Slowhttptest [31], GoldenEye [28] and HULK [4]. In the afternoon, the vulnerable OpenSSH server falls victim to the Heartbleed bug [14] exploitation.

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

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

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

Http Unbearable Load King (HULK) attacks the server by flooding it with raw UDP packets. As UDP does not require a three-way handshake to set up a connection like TCP does, this attack can simply flood the server with packets

it must handle, overloading it. HULK works best as a distributed attack with multiple malicious clients flooding the target.

The Heartbleed bug [14] was first discovered in 2014 as a vulnerability in the implementation of the Heartbeat extension proposed by RFC6520. First introduced in December 2011, this bug allows anyone to read the memory of the SSH server due to a missing bounds check [23]. As the process memory usually contained the unencrypted secret key used to encrypt communications with, it became the primary target of Heartbleed exploits, allowing subsequent decryption of previous and ongoing encrypted communications. CIC-IDS-2017 authors implemented this attack using the Heartleech tool [12] against a vulnerable OpenSSL server version 1.0.1f.

3.1.4 Thursday

On Thursday, authors implement a Web and Infiltration attack classes.

Web attacks consist of attacking the Damn Vulnerable Web App (DVWA) hosted on one the webservers. The authors attack the application by performing a bruteforce attack on the login page of the application and by performing a SQL injection and Cross-Site-Scripting (XSS) attack.

SQL injection attack exploits careless replacements of variables in strings. Malicious strings can alter the behaviour of the query, leaking private information hidden within the database.

XSS attacks occur when the attacker injects malicious code in to the web application that subsequently gets run for another user. In this instance, SQL injection is performed against the dedicated endpoint for practicing SQL injections provided by DVWA and the XSS attack is conducted by leaking a cookie into the browser's console.

Infiltration attack is executed by having the user download and execute an infected file on the victim network, which in turn sets up a reverse shell through the Metasploit Framework [26] and allows the attacker to execute portscan of the entire victim network using NMap [39].

3.1.5 Friday

On Friday morning, the authors conducted a botnet attack through the Ares [40] tool, enslaving the computers in the victim network and getting them to take a screenshot of the desktop.

On Friday afternoon, a Distributed Denial-of-Service (DDoS) attack is conducted against the Ubuntu 16 webserver using the Low Orbit Ion Cannon (LOIC) [22]. This tool is started on the Windows 8.1 hosts in the attacker network and together, they conduct the DDoS attack.

Finally, last attack class is introduced: Portscan. While we have previously seen this attack as a part the infiltration attack, here we observe it in its raw form. All Windows machines in the victim network are portscanned for running services, using "the main NMap switches such as sS, sT, sF, sX, sN, sP, sV, sU, sO, sA, sW, sR, sL and B." [29]. In our analysis of the raw PCAP files,

we find that switches sF, sX, sP and sA are missing entirely: there is no traffic at the times specified by the dataset authors, nor is there any traffic present that would arise from these switches. Furthermore, the switch -sL performs no attack at all, but simply lists the hosts that would be scanned by the tool.

3.1.6 Troubleshooting the dataset

As alluded to before, Engelen et al. [8] have performed an extensive analysis of this dataset and found numerous errors in labeling and construction: errors in the construction of the actual flows, imprecise documentation, incorrect labeling, among others. We use their fixed version of the dataset in our experiments.

3.2 Reconstruction

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

ConCap operates on a Kubernetes cluster, spinning up pods that communicate with each other. By defining a scenario file in the YAML format, users can easily spin up containers to perform a wide variety of tasks. ConCap is designed to capture traffic between pods and analyse it using traffic analysis tools such as CICFlowmeter [16] or Rustiflow [35]. As ConCap is primarily aimed at constructing attack traffic In order to construct a scenario, the user can define the attacker and target images, the network configuration and the labelling of generated NetFlows from the traffic occuring on the network. In order to supply the attack tools with the IP addresses of targets, an attack command can be specified with environment variables which will be resolved at runtime.

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

3.2.1 Tuesday

In reconstructing the Tuesday attacks, we use Patator by Sebastien Macke [19], more specifically its version tagged 0.8. We construct a Docker image containing Patator and a list of usernames and passwords to try against the server. We build this image under the name `themessik/patator` and push it to Dockerhub. It is unclear what dictionaries authors use for the bruteforce attacks, forcing us to make an arbitrary choice. We choose for a username and password list from the SecLists [20] repository: Top Usernames Shortlist[2] and Pwdb Top 1000[3]

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

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

3.2.2 Wednesday

For the DoS attacks, we use the respective tools mentioned by the dataset authors. We use Sergey Shekyan's Slowhttptest [31] tool to reconstruct both Slowhttptest and Slowloris attacks, as this tool provides the required functionality for both. For GoldenEye and HULK, we build dedicated Docker images (`themessik/goldeneye` and `themessik/hulk` respectively) and point them at the webserver. For the implementation of the HULK attack, we use a ported version rewritten in Go. This version should maintain the semantics of the original Python tool, as there are multiple such tools named HULK available online, making it unclear which tool the dataset authors use or which one is the original.

For the Heartbleed attack, we build dedicated target image (`themessik/heartbleed`) based on Ubuntu 16.04 and install the Heartbleed-vulnerable version of OpenSSL 1.0.1f, mirroring the authors. For the attacker image (`themessik/heartleech`), we use the Robert David Graham's tool heartleech [12] to execute the attack, using the `--autopwn` option to extract the private key out of the server.

3.2.3 Thursday

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

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

3.2.4 Friday

On Friday morning, a botnet attack using Ares is executed. We choose to omit this attack in our reconstruction due to technical limitations: Ares does not

provide a way to control the botnet from Linux hosts and is reliant on Windows for its execution. Due to this, we cannot package it into a Docker container and execute attacks with it on ConCap.

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

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

4 Experimental verification

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

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

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

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

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

For every attack class, we prepare the two datasets for experiments: the fixed dataset provided by [8] (the CIC traffic) and the synthetic dataset generated by ConCap (the ConCap traffic). We treat both datasets with the same cleaning procedure as describe above.

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

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

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

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

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

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

- **Precision**: Precision describes how precise the model is in its positive predictions, calcualted as a ratio of true positives and all predicted positives:
$$\frac{TP}{TP + FP}$$
- **Area Under Receiver Operator Characteristic Curve (ROC AUC) score**: By calculating the True Positive Rate and False Positive Rate of the model at set intervals, a curve can be plot through them. Area under this curve expresses how the probability of the model predicting a random malicious sample as malicious.

The processed NetFlow files can be found on Kaggle[4]. We perform the verification experiment for each attack class separately.

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

4.1 Tuesday

4.1.1 FTP

For FTP Bruteforce, Figures 2 and 3 show a range of features performing excellenty, with many scoring above .95 ROC AUC Score. Averaing out the results, we find the top three best performing features for both ways to be Bwd RST Flags (0.978288), Packet Length Mean (0.976290) and Average Packet Size (0.976290).

4.1.2 SSH

For SSH Bruteforce, Figures 4 and 5 show a smaller range of features than FTP Bruteforce, but nonetheless these features maintain a high predictive power. For the top three best performing common features, we find Fwd Seg Size Min (0.922024), Fwd IAT Min (0.854907) and Bwd Segment Size Avg (0.817511).

[4] `https://www.kaggle.com/datasets/jozefjankaj/thesis-files`

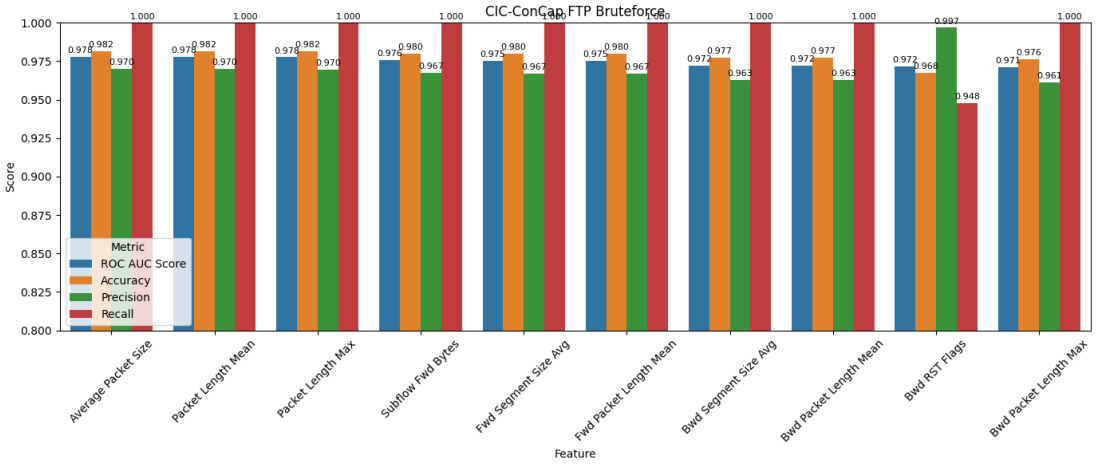

Figure 2:
Best performing features for FTP Bruteforce when trained on CIC-IDS-2017
and measured on ConCap traffic

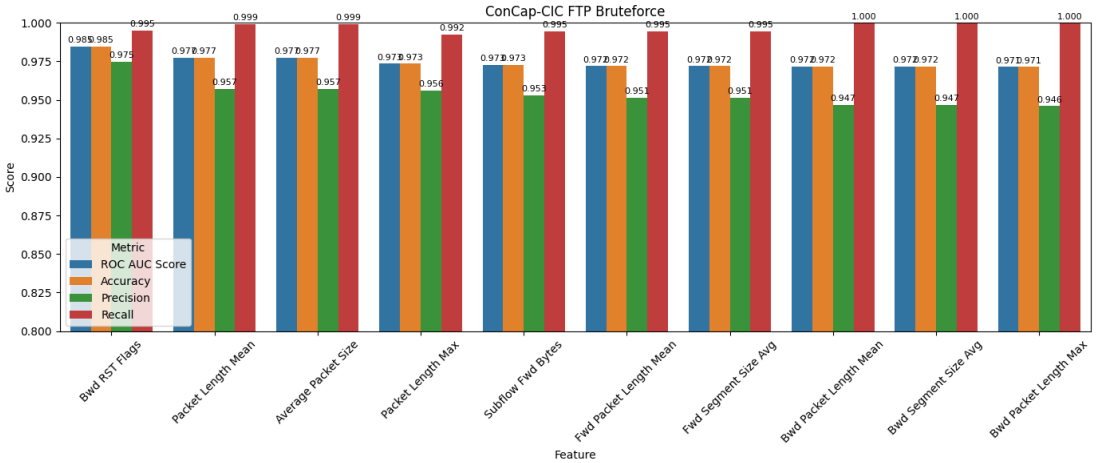

Figure 3:
Best performing features for FTP Bruteforce when trained on ConCap traffic
and measured on CIC-IDS-2017

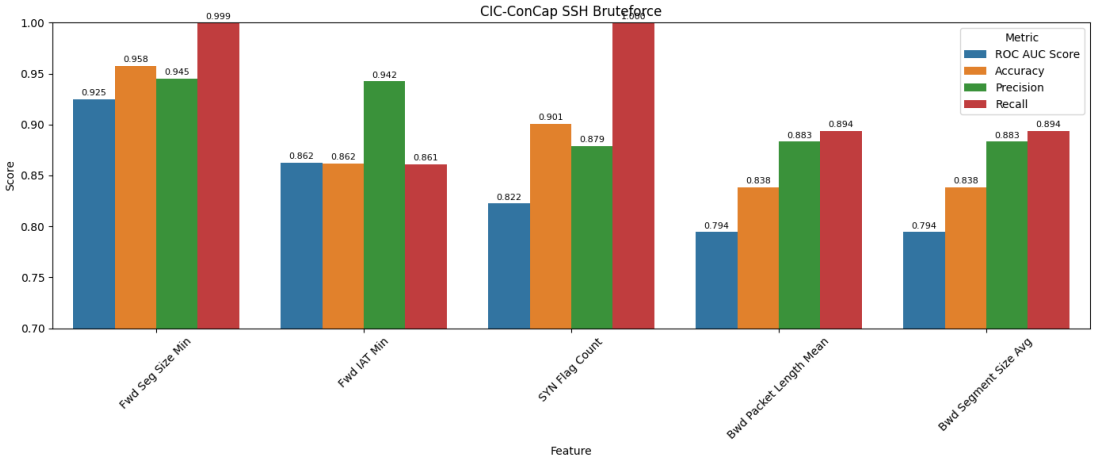

Figure 4:
Best performing features for SSH Bruteforce when trained on CIC-IDS-2017 traffic and measured on ConCap traffic

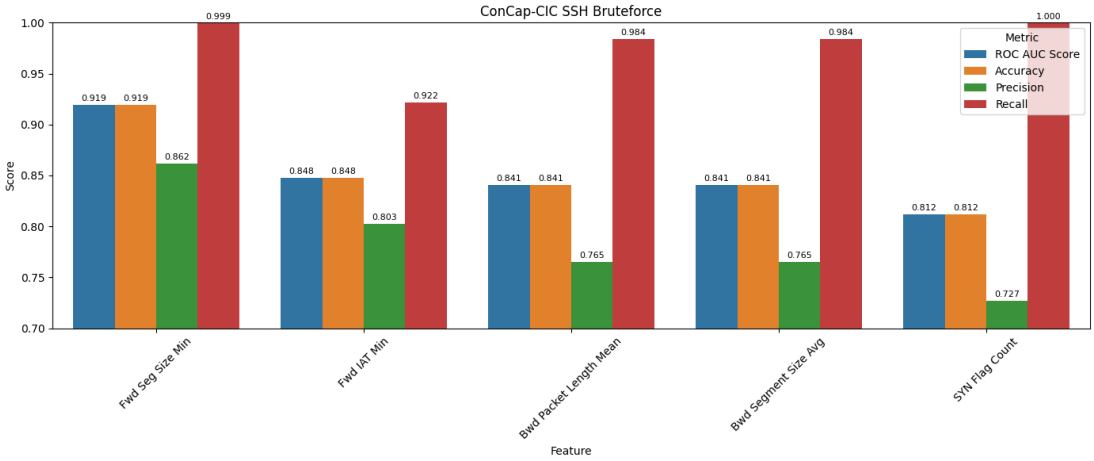

Figure 5:
Best performing features for FTP Bruteforce when trained on ConCap traffic and measured on CIC-IDS-2017

4.2 Wednesday

4.2.1 Slowloris

For Slowloris, we mainly find features that focus on flow time and size of packets, as shown by Figures 6 and 7. Interestingly, we can see that the recall metric for features focused on size plummets when trained on CIC-IDS-2017, suggesting a difference in implementation we were not able to spot. Nonetheless, given the nature of this attack, we expect the main feature with high predictive power to be related to Flow time, and Total TCP Flow Time is precisely that feature. In terms of best common performing features, we find Total TCP Flow Time (0.910773), Bwd Packet Length Max (0.868125) and Total Length of Bwd Packet (0.868125).

4.2.2 Slowhttptest

Similarly to Slowloris, Figures 8 and 9 show the Total TCP Flow Time feature to perform best, maintaining high recall and ROC AUC Score. Interestingly, we see far more features that focus on timing than rather than size that we've seen in Slowloris attacks. For the best performing features, we find Total TCP Flow Time (0.905697), Fwd IAT Min (0.875804) and Fwd IAT Total (0.842901).

4.2.3 HULK

For HULK, Figures 10 and 11 show that mostly features focusing on forward and backward packets hold the most predictive power, performing well across all metrics. For the best performing features, we find Bwd Packet Length Std (0.977942), Fwd RST Flags (0.954837) and Subflow Bwd Bytes (0.951484) to perform best for both ways of testing.

4.2.4 GoldenEye

Figures 12 and 13 show a mix of features that predicts GoldenEye attacks well, without a particular focus on any one category. The best performing features are Bwd Packet Length Std (0.939286), Packet Length Variance (0.927954) and Packet Length Std (0.927954).

4.2.5 Heartbleed

For Heartbleed, Figures 14 and 15 show only three features scoring above 0.8 ROC AUC Score, with Bwd Packet Length Std being delivering a perfect predictor in both ways of training. We do need to note here that CIC-IDS-2017 has a low amount of Heartbleed flows (11), raising questions about this model's actual performance due to these signs of overfitting. Nevertheless, the best performing features are the three shown, Bwd Packet Length Std (1.000000), Flow Bytes/s (0.931818) and Packet Length Std (0.905702).

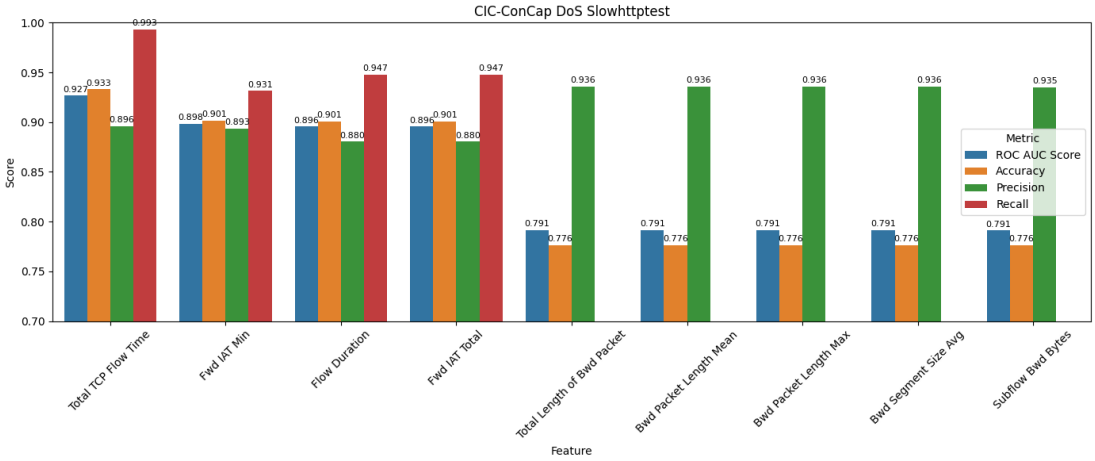

Figure 6:
Best performing features for DoS Slowloris when trained on CIC-IDS-2017 traffic and measured on ConCap

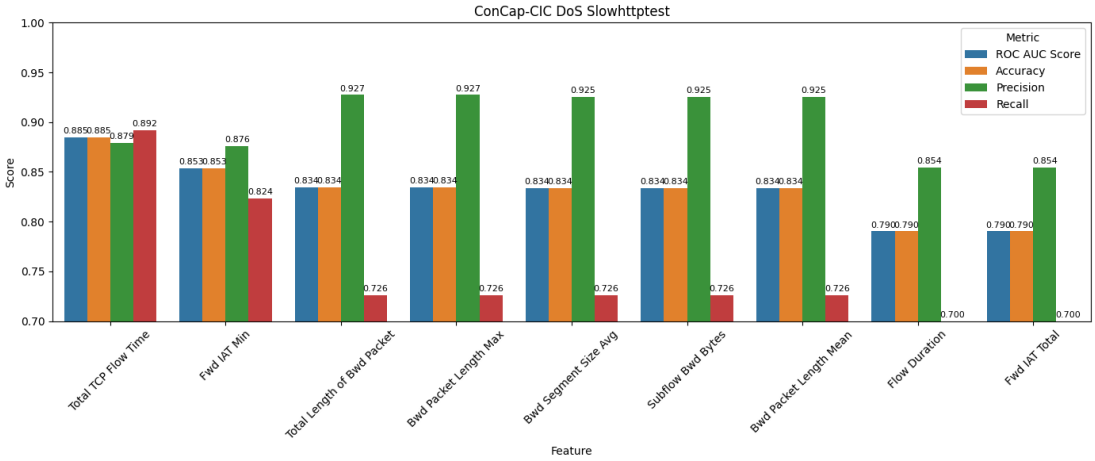

Figure 7:
Best performing features for DoS Slowloris when trained on ConCap traffic and measured on CIC-IDS-2017

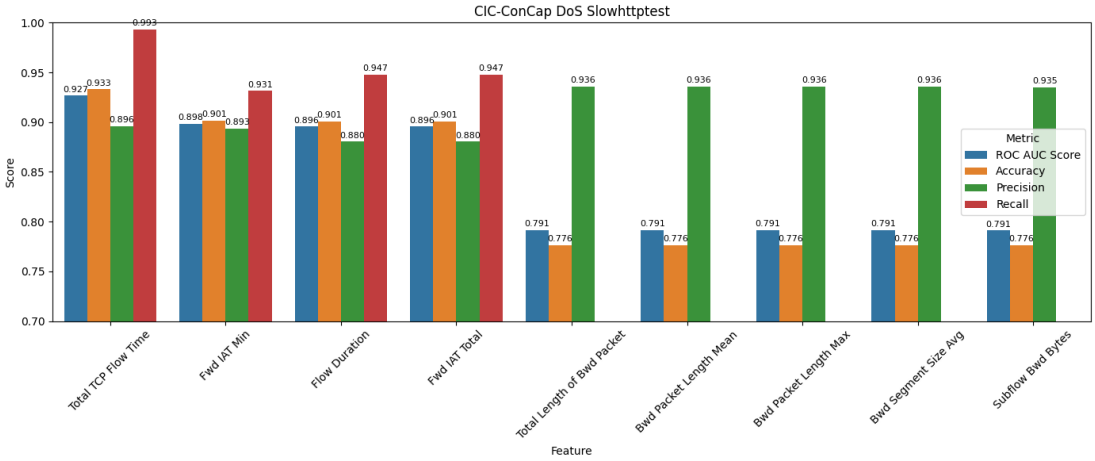

Figure 8:
Best performing features for DoS Slowhttptest when trained on ConCap traffic and measured on CIC-IDS-2017

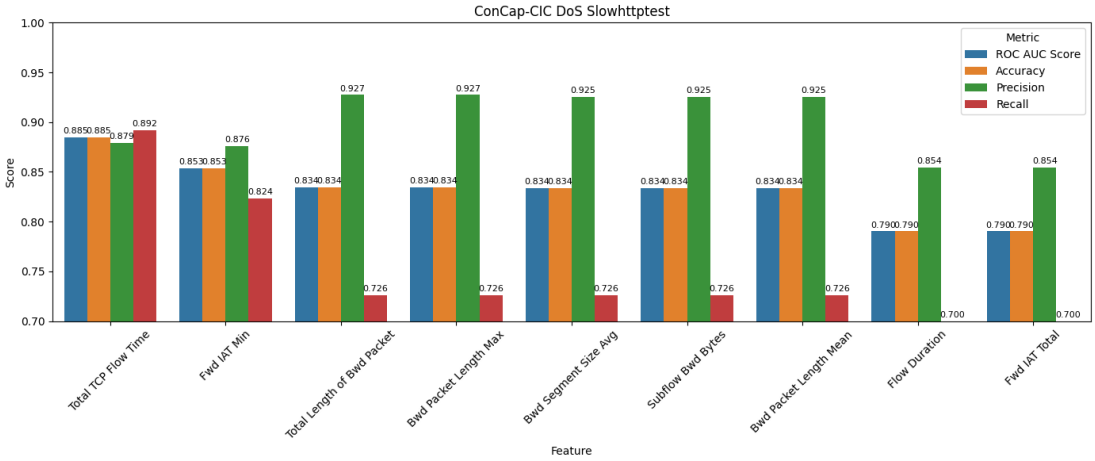

Figure 9:
Best performing features for DoS Slowhttptest when trained on ConCap traffic and measured on CIC-IDS-2017

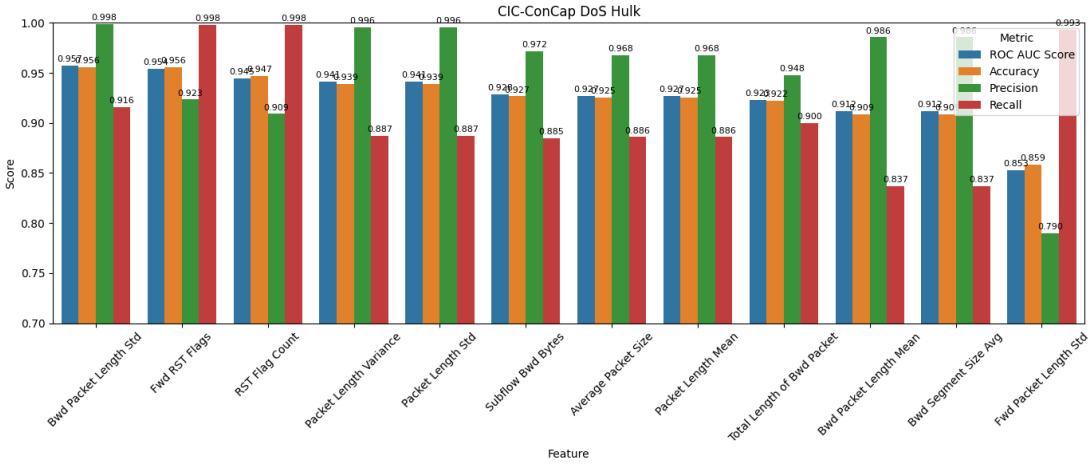

Figure 10:
Best performing features for DoS HULK when trained on CIC-IDS-2017 traffic and measured on ConCap

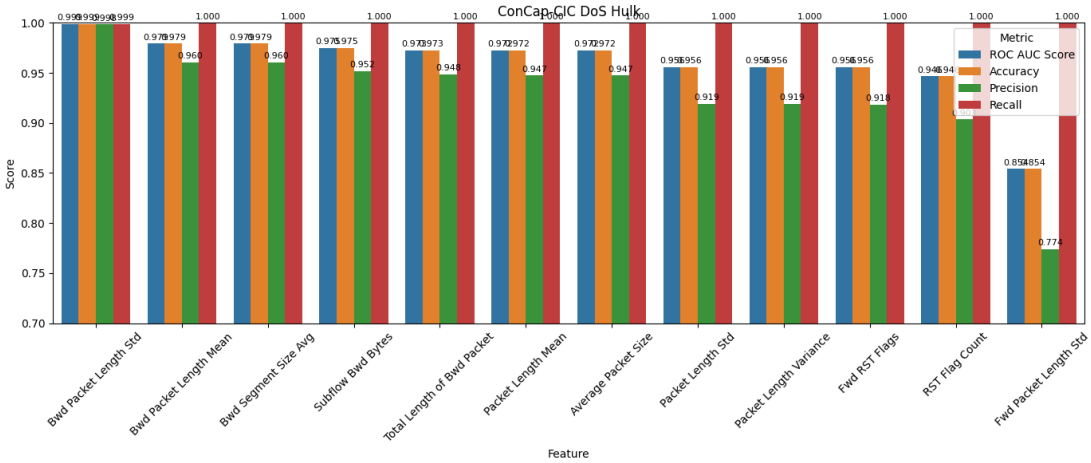

Figure 11:
Best performing features for DoS HULK when trained on ConCap traffic and measured on CIC-IDS-2017

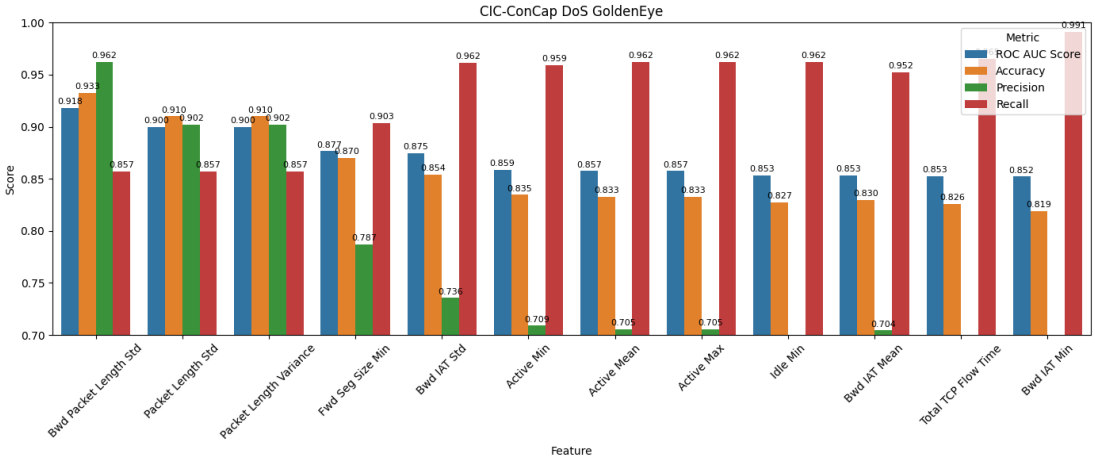

Figure 12:
Best performing features for DoS GoldenEye when trained on CIC-IDS-2017 traffic and measured on ConCap

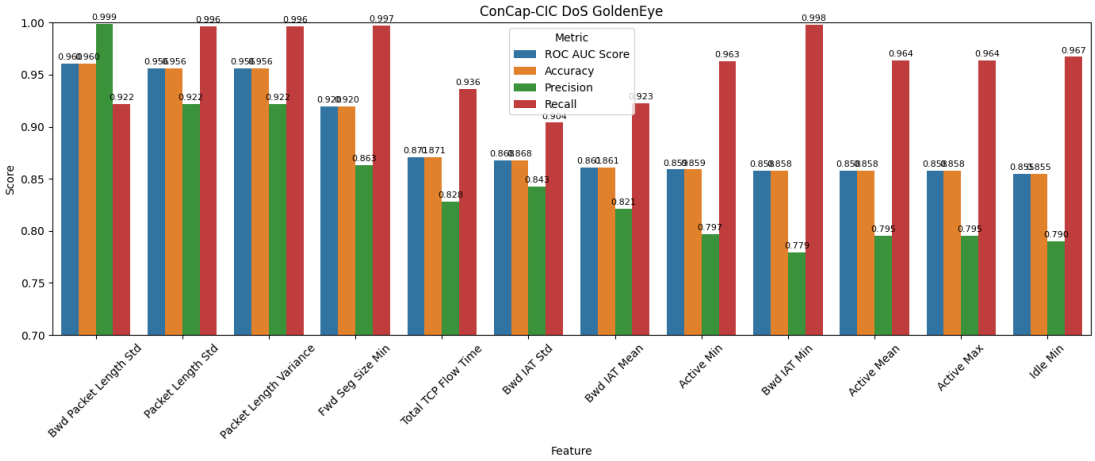

Figure 13:
Best performing features for DoS GoldenEye when trained on ConCap traffic and measured on CIC-IDS-2017

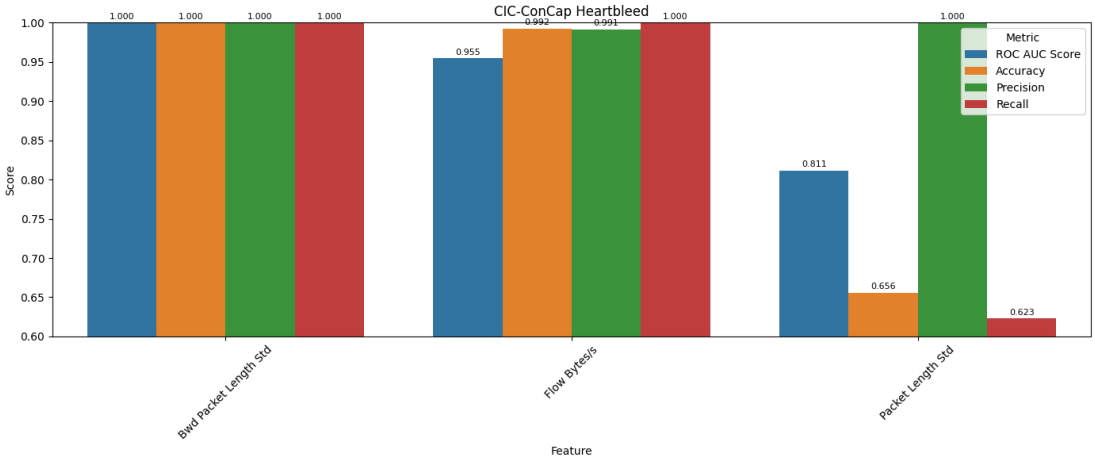

Figure 14:
Best performing features for Heartbleed bug exploit when trained on CIC-IDS-2017 traffic and measured on ConCap

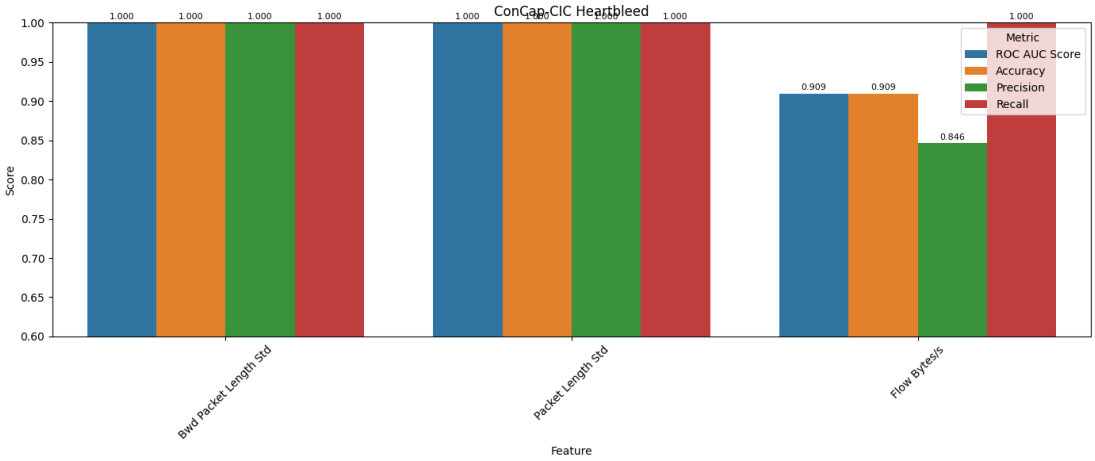

Figure 15:
Best performing features for Heartbleed bug exploit when trained on ConCap traffic and measured on CIC-IDS-2017

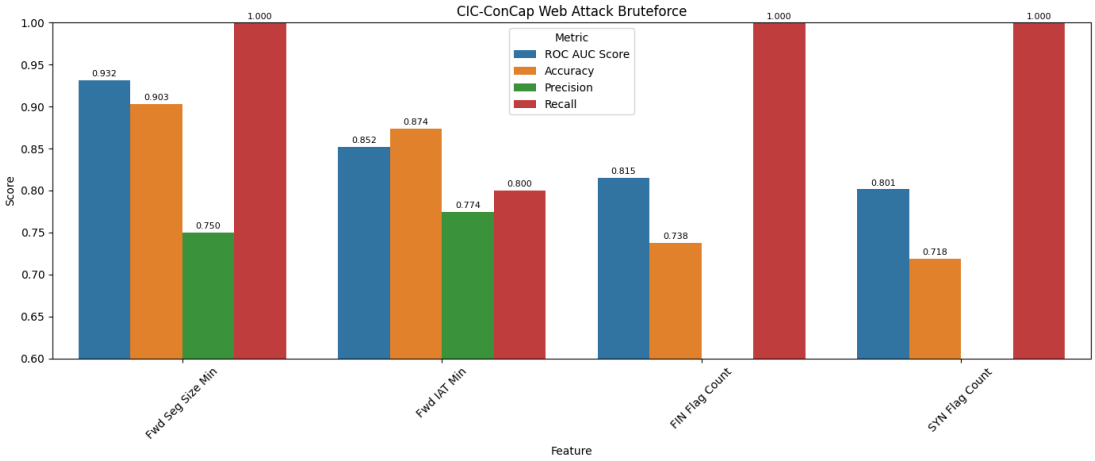

Figure 16:
Best performing features for Bruteforce Web Attack when trained on CIC-IDS-2017 traffic and measured on ConCap

4.3 Thursday

4.3.1 Web Attacks

Similarly to Heartbleed, there's only a small number of flows for each of the subcategories of Web Attacks (73, 13 and 18 for Bruteforce, SQL Injection and Cross-Site Scripting respectively). Also similarly we observe features achieving perfect recall, leading us again to suggest that these models are overfitting and not generalizing well. Nevertheless, Figures 16 and 17 show the best performing features of the Bruteforce Web Attack, with best performing features being Fwd Seg Size Min (0.934932), Fwd IAT Min (0.891781) and FIN Flag Count (0.828767).

Figures 18 and 19 show the best performing features of SQL Injection attack, with best performing features being Fwd Seg Size Min (0.884615), FIN Flag Count (0.865385) and Bwd IAT Min (0.816719).

Lastly, Figures 20 and 21 show the best performing features for Cross-Site Scripting attack, with best performing features being Fwd Seg Size Min (0.930556), Bwd Packet Length Std (0.869792) and Packet Length Max (0.845486).

It is clear that Fwd Seg Size Min is the best predictor for the simulated Web Attacks, scoring average ROC AUC score above 0.8 in each category.

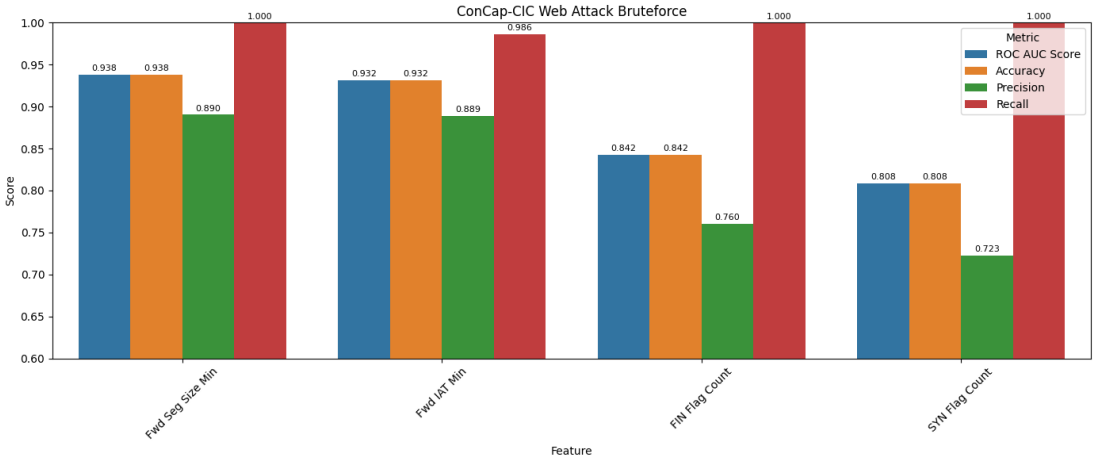

Figure 17:
Best performing features for Bruteforce Web Attack when trained on ConCap traffic and measured on CIC-IDS-2017

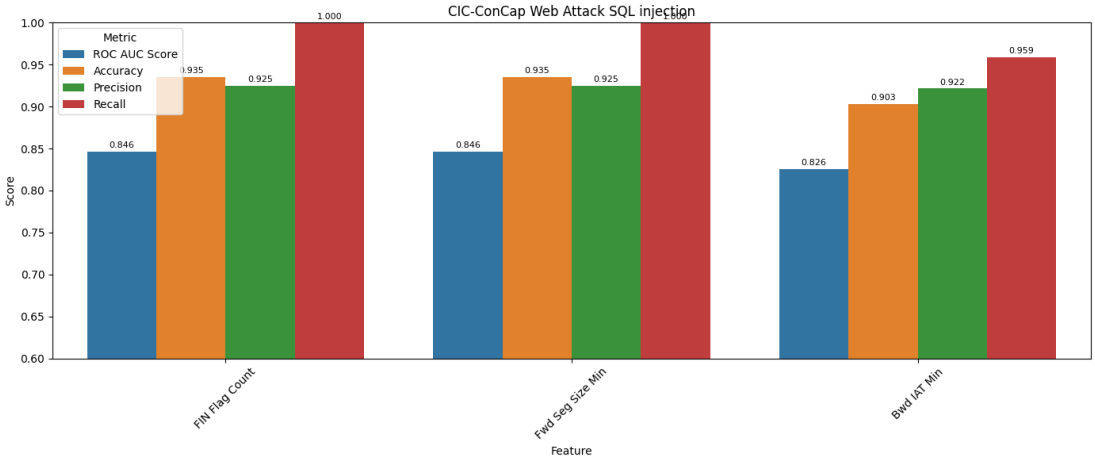

Figure 18:
Best performing features for SQL Injection Web Attack when trained on CIC-IDS-2017 traffic and measured on ConCap

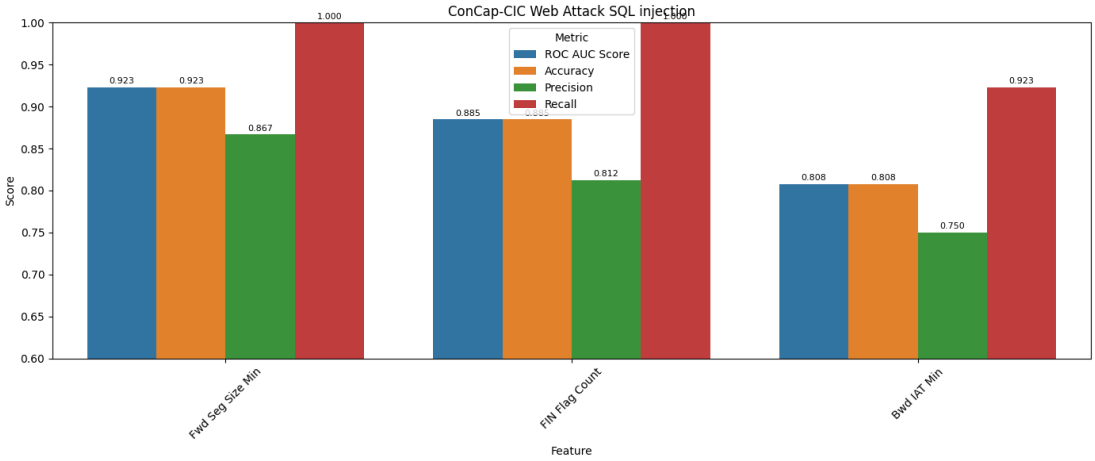

Figure 19:
Best performing features for SQL Injection Web Attack when trained on ConCap traffic and measured on CIC-IDS-2017

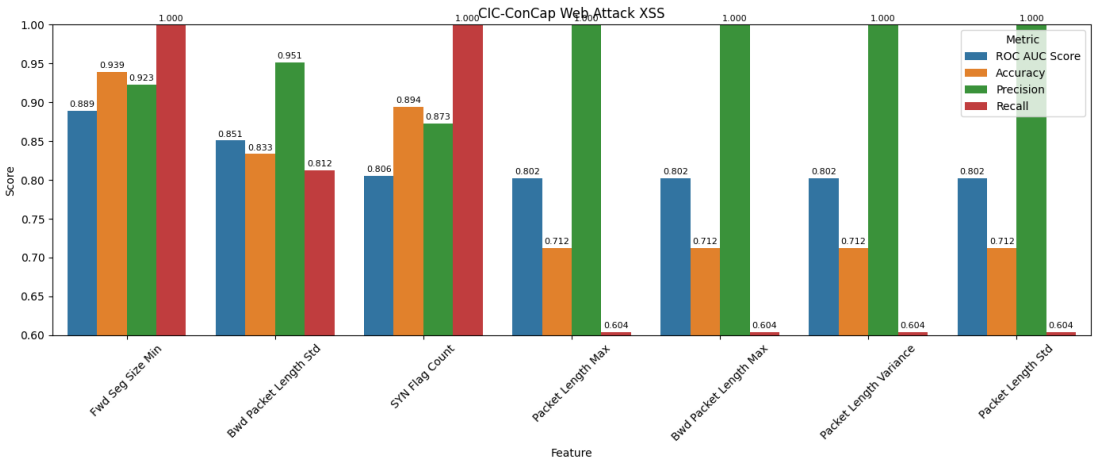

Figure 20:
Best performing features for Cross-Site Scripting Web Attack when trained on CIC-IDS-2017 traffic and measured on ConCap

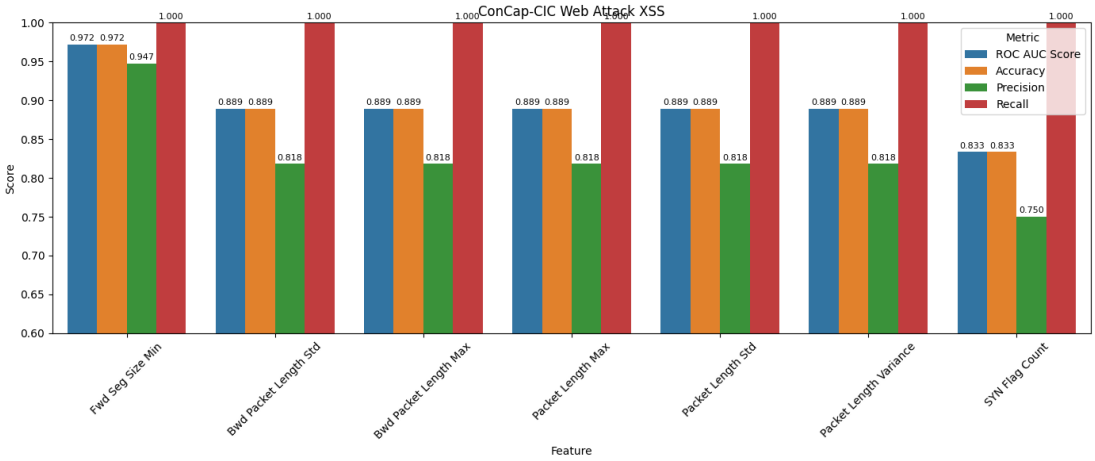

Figure 21:
Best performing features for Cross-Site Scripting Web Attack when trained on ConCap traffic and measured on CIC-IDS-2017

4.4 Friday

4.4.1 LOIC

For the DDoS LOIC attack, Figures 22 and 23 show the focus on features focusing on packet travel time (Fwd IAT Min) as well as opening connections (SYN Flag Count), as we expect given the nature of LOIC being a DDoS tool. The best performing features are Fwd Seg Size Min (0.934932), Fwd IAT Min (0.860959) and FIN Flag Count (0.821918), with SYN Flag Count (0.821918) having the same score as FIN Flag Count.

4.4.2 Portscan

Finally, Figures 24 and 25 show the best performing metrics for the Portscan attacks. Noticable is that these attacks are apparently quite easy to classify, given the large amount of metrics that achieve above 0.9 ROC AUC score. Best features appear to be Fwd Packet Length Max (0.949676), Total Length of Fwd Packet (0.948696) and Fwd Segment Size Avg (0.947596).

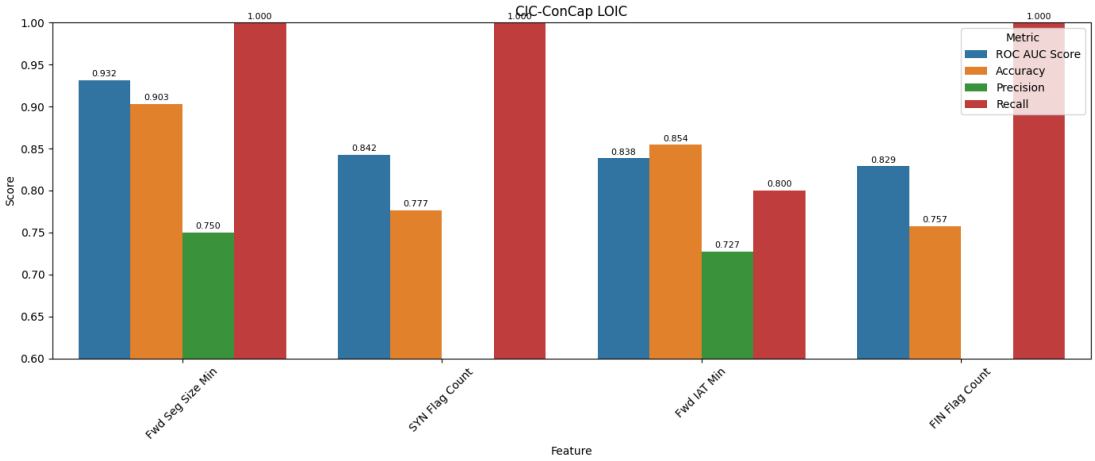

Figure 22:
Best performing features for DDoS LOIC attack when trained on CIC-IDS-2017 traffic and measured on ConCap

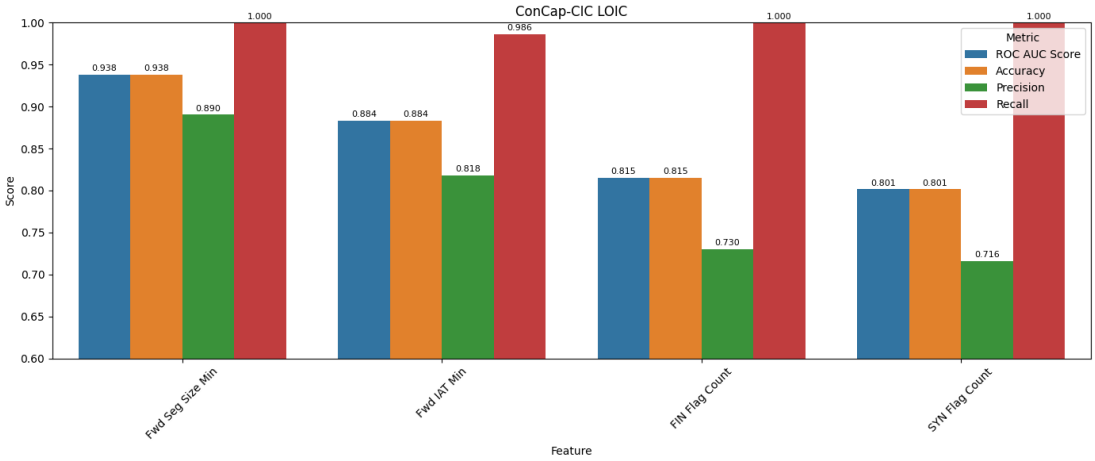

Figure 23:
Best performing features for DDoS LOIC attack when trained on ConCap traffic and measured on CIC-IDS-2017

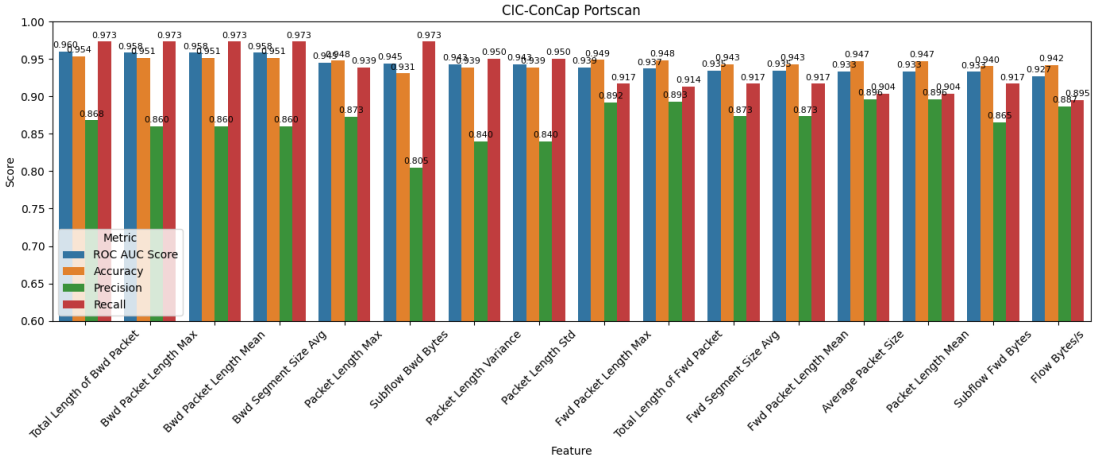

Figure 24:
Best performing features for Portscan attack when trained on CIC-IDS-2017 traffic and measured on ConCap

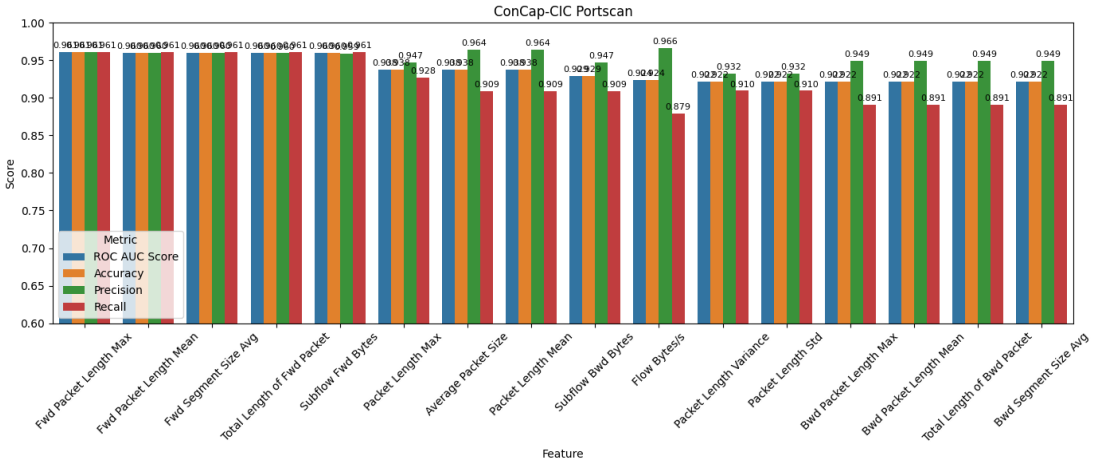

Figure 25:
Best performing features for Portscan attack when trained on ConCap traffic and measured on CIC-IDS-2017

4.5 Discussion

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

First, although ConCap allows for the configuration of network features that affect communication speed, we lack this information about the network conditions of the CIC-IDS-2017 dataset. For this reason, we opt to use the example configuration provided by ConCap authors and keep these conditions the same for all scenarios. Consequently, this could have impact on the prominance of features that are largely determined by the specific network characteristics.

Second, ConCap networks are built fully in-silico and thus do not suffer from the same artifacts as physical networks might: arbitrary delay in packet processing, faulty connections or interference, as well as processing delay. ConCap network architecture is, without introduction of artificial defects, the perfect network. As we lack information about packet drops, reordering or corruptions, we cannot recreate these faithfully, which could have an impact on feature prominance.

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

Nevertheless, the results above clearly show presence of features that hold high predictive power for both experiments, leading us to conclude that traffic generated by ConCap can be used instead of the original dataset traffic, as it posesses similar qualities.

5 Enhancing the Robustness

Model robustness refers to the model's ability to generalize to unseen samples or to variations of previously seen samples. Using ConCap, extra malicious samples and their variations can be easily generated, allowing for adversarial training of models. After having experimentally verified that ConCap can be used as a stand-in for dataset traffic, in this section, we want to investigate the effect of dataset augmentation on the robustness of models.

Our investigation starts by generating adversarial samples from three classes of attacks contained in CIC-IDS-2017:

- **FTP Bruteforce no persistence**: By default, Patator will use persistent connections, allowing it to try multiple options over single established connection. This bandwidth-saving measure is widely used on the web to increase communication speed. By turning connection persistence off, we force Patator to only try a single username-password combination before closing the connection.
- **DoS GoldenEye POST**: The attacks with GoldenEye in CIC-IDS-2017 dataset are conducted using the HTTP verb GET. For this adversarial class, we use the HTTP POST verb instead.
- **DDoS LOIC UDP**: By default, LOIC performs DoS attacks using TCP traffic, but provides an option to conduct an attack with UDP as well, which we utilise in this adversarial sample.

With the adversarial samples ready, we first establish a baseline in order to quantify the impact of adversarial training on model performance. Our methodology is similar to the methodology described in Section 4: we use the CIC-IDS-2017 traffic as a training set and the adversarial samples as a testing set, balancing both with benign flows as previously described.

We train a model on the training set and record its performance on the testing set, using the best performing feature as found in Section 4. Table 1 shows the recorded baselines for each of the adversarial samples.

We can see that the model performs very poorly on these adversarial samples, having at worst 30% probability of predicting a LOIC UDP attack correctly; the probability of predicting either non-persistent FPT bruteforce or GoldenEye POST DoS attack is no better than a coinflip.

Having established a baseline, we prepare a training set that consists of a mix between CIC-IDS-2017 traffic and adversarial traffic, using a 50% train-test

Table 1: Dataset augmentation baseline ROC AUC scores

Attack class	Baseline
FTP No Persistence	0.4982
GoldenEye POST	0.48923
LOIC UDP	0.30098

Table 2: Dataset augmentation: Baseline + Adversarial ROC AUC scores

Attack class	Baseline	Adversarial	Difference
FTP No Persistence	0.49822	0.59236	0.09414
GoldenEye POST	0.4887	0.8454	0.35671
LOIC UDP	0.2951	0.78802	0.49292

Table 3: Dataset augmentation: Baseline + Adversarial ROC AUC Scores on CIC-IDS-2017

Attack class	Baseline	Adversarial	Difference
FTP No Persistence	0.99577	0.99615	0.00038
GoldenEye POST	0.98522	0.84716	-0.13807
LOIC UDP	0.78831	0.78879	0.00048

split of adversarial samples. We retrain the model and measure its performance again. Table 2 extends Table 1 with the performance of this adversarially-trained model and the difference in the two models. We see clear improvements in model performance, showing that ConCap can indeed be used for adversarial training of models to increase their robustness.

However, these results would be worthless if the model's ability to predict original traffic would be diminished. As a last test, we also verify that this is not the case, by having both baseline and adversarial models also predict the testing portion of the CIC-IDS-2017 dataset, without adversarial flows mixed in. The results of this experiment are presented in Table 3.

It is interesting to see that the change of protocol did cause GoldenEye model to reduce its predictive power by 13.8% on average. However, this is to be expected, as this variant of the attack has more drastic implications for the underlying traffic than the other adversarial samples. It is positive to see that the other two adversarially trained models maintain their performance. From all this, we can conclude that ConCap can be feasibly used for adversarial training of models, increasing their robustness against variants of attacks.

6 Conclusion

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

In this thesis, we have experimentally verified a new method of constructing ML-NIDS datasets through the ConCap framework by reconstructing the CIC-IDS-2017 dataset into ConCap scenarios and showing that the resulting dataset is representative of the original. We find features that hold high predictive power for model training and conclude that ConCap can be used for dataset reconstruction.

Furthermore, we have further shown that the robustness of ML-NIDS models can be increased by augmenting their training with ConCap traffic, which improves their generalization performance to unseen samples from other datasets. For this, we have generated three variants of attacks already contained in CIC-IDS-2017 and performed adversarial training. Our results show that through adversarial training, ML models can be taught to recognise variants of the attacks without large reductions to their predictive power on the original dataset.

All of this further cements ConCap (and other such frameworks) as the next step in ML-NIDS dataset generation and augmentation.

7 Future Work

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

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

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

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

8 References

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

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

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

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

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

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

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

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

[9] External Data Source. *Kyoto 2006+ Dataset (11/01/2006 to 12/31/2015)*. 2018.

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

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

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

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

[14] *Heartbleed*. URL: https://heartbleed.com/.

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

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

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

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

[19] Sebastien Macke. https://github.com/lanjelot/patator.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[34] Geert-Jan (ugent)01802387 Van Nieuwenhove. *Genereren van DDoS aanvallen voor het creëren van een dataset*. und. 2023. URL: http://lib.ugent.be/catalog/rug01:003150617.

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

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

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

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

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

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

A Artifacts

The scenario files, the Docker images and Jupyter notebooks used in this thesis and raw LaTeXfiles for this text can be found on `https://github.com/TheMessik/thesis`. Should this link not function, do not hesitate to contact the author and request a copy.

The generated traffic files can be found on `https://www.kaggle.com/datasets/jozefjankaj/thesis-files`. Follow the procedures in the Jupyter notebooks for an example how to use them.