FACULTY OF ENGINEERING AND ARCHITECTURE

Towards Secure and Generalizable Machine Learning for Network Intrusion Detection

Miel Verkerken

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Information Engineering Technology

Supervisors
Prof. Filip De Turck, PhD - Prof. Bruno Volckaert, PhD
Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

July 2025

GHENT UNIVERSITY

ISBN 978-94-93464-04-9
NUR 986, 984
Wettelijk depot: D/2025/10.500/64

Members of the Examination Board

Chair

Prof. Em. Hendrik Van Landeghem, PhD, Ghent University

Other members entitled to vote

Prof. Giovanni Apruzzese, PhD, University of Liechtenstein, Liechtenstein
Prof. Wouter Joosen, PhD, KU Leuven
Prof. Eric Laermans, PhD, Ghent University
Prof. Sam Leroux, PhD, Ghent University
Prof. Didik Sudyana, PhD, National Cheng Kung University, Taiwan

Supervisors

Prof. Filip De Turck, PhD, Ghent University
Prof. Bruno Volckaert, PhD, Ghent University

Acknowledgements

This dissertation marks the conclusion of a long and fulfilling journey. I chose the scenic route, completing a six-year teaching assistant mandate that combined research with teaching. Serving as a teaching assistant was not only an opportunity to deepen my research expertise but also a rewarding experience in sharing knowledge with students. I consider this dual role a privilege that has profoundly shaped both my academic development and personal growth.

I could never have completed this journey alone, it was made possible by the support of many people behind me.

First and foremost, I would like to thank my promoters, Prof. Filip De Turck and Prof. Bruno Volckaert, for believing in me and offering me this opportunity. Your guidance and support created a safe and motivating environment where I could freely explore ideas and take ownership of my research. Filip, your diplomatic communication style and your calm perspective taught me the value of clarity and empathy. Bruno, your strong work ethic inspired me to keep pushing forward, even during the most challenging moments.

I would also like to thank all my colleagues in the Information Engineering Technology program for their collaboration and support. In particular, I am especially grateful to Prof. Veerle Ongenae for her guidance throughout the educational side of my mandate. From the very beginning, she believed in me and placed her trust in my abilities, which gave me the confidence to grow into my role as a teacher.

I've also been fortunate to work with great colleagues at IDLab, and I would like to thank all of them, not only for the engaging discussions about research, but also for the many conversations about life beyond academia. In particular, Laurens, whose research aligned closely with mine. Our many discussions helped shape both our work, and validating and challenging each other's ideas made the research process all the more enriching. José, thank you for introducing me to the networking research community and for helping me broaden my expertise into cloud computing. Your guidance has had a lasting impact on my growth as a researcher. I also wish to thank Ghent University, imec, and the entire IDLab research group for providing the resources and community that made this work possible.

I am grateful to the members of my doctoral jury for their time and constructive feedback. Their critical review has undoubtedly contributed to the improved quality of this dissertation. I would like to express my special thanks to Prof. Giovanni Apruzzese for the close collaboration during the final year of my PhD, which began with a research stay at the University of Liechtenstein. "It takes two to think", and through our (often late-night) discussions, I experienced the true satisfaction

that comes from scientific research. Thanks to your mentorship, the final stretch of my PhD felt like a high point, and I look back on it with pride. Prof. Didik Sudyana, our story has been a unique one: from over a year of virtual collaboration, to your visit as an exchange PhD student in 2024, and now to your role as a jury member at my defense. I'm truly grateful that our paths have crossed in so many meaningful ways.

On a more personal note, I want to thank my girlfriend, Mia. Your patience, care, and perseverance were often the glue that held everything together, especially during those long, sleep-deprived nights before another deadline. Thank you for managing our life when I was fully absorbed in my work, for celebrating every win—no matter how small—and for being proud of me even when I wasn't. This dissertation is as much a reflection of your support as it is of my effort.

To my parents, Bart and Carine, thank you for giving me every opportunity in life and supporting my choices without hesitation. To my grandparents, Herman, Rita, and Nelly, thank you for always being there, helping in countless ways, and cheering me on with pride. To my sisters, Leie and Kaat, and to my wider family, who helped me in visible and invisible ways throughout this journey.

To my friends, whether near or far, thank you for keeping me grounded, for your friendship, and for reminding me of everything outside academic life. Your presence in my life has brought laughter, balance, and perspective throughout this journey.

Lastly, I am proud of the work I present here, not because it answers all questions, but because it reflects honest curiosity, effort, and a desire to contribute something meaningful. I look forward to seeing where the road leads next.

Ghent, June 2025
Miel Verkerken

Table of Contents

List of Figures

List of Tables

Samenvatting
– Summary in Dutch –

In onze moderne samenleving zijn digitale systemen diep geïntegreerd in vrijwel elke sector, van gezondheidszorg en financiën tot energie en transport. Hoewel deze onderling verbonden digitale systemen veel maatschappelijke voordelen bieden, vergroten ze ook aanzienlijk het aanvalsoppervlak voor indringers. Netwerk-intrusiedetectiesystemen (NIDS'en) dienen al lange tijd als een belangrijke bescherming tegen mogelijke bedreigingen. Traditionele IDS'en ondervinden echter steeds meer moeilijkheden bij het omgaan met nieuwe dreigingen, geëncrypteerd verkeer en snel evoluerende aanvalspatronen. Vooral IDS'en die gebaseerd zijn op handmatige signaturen—oftewel patronen van bekende aanvallen, die vergeleken worden met de inhoud van het netwerkverkeer—schieten hierin tekort. Om deze tekortkomingen aan te pakken, richt de cybersecurity onderzoeksgemeenschap zich in toenemende mate op machinaal leren gebaseerde NIDS'en (ML-NIDS'en). Ondanks aanzienlijke academische interesse sinds de jaren '80, blijft de praktische adoptie van ML-NIDS'en beperkt.

Dit proefschrift onderzoekt de uitdagingen die de overgang van academisch onderzoek naar praktische operationele toepassingen van ML-NIDS'en belemmeren. Deze uitdagingen worden onderzocht aan de hand van vijf onderzoeksvragen die (1) de classificatieprestaties van *unsupervised* ML-modellen, (2) de generalisatiekracht van ML-NIDS'en, (3) het ontwerp van gedistribueerde ML-NIDS voor cloud-omgevingen, (4) de praktische obstakels voor gebruik in productie omgevingen, en (5) de haalbaarheid om een ML-NIDS te omzeilen met realistische bedreigingsmodellen behandelen. Om structuur te geven aan het beantwoorden van de onderzoeksvragen, is het proefschrift opgebouwd rond drie thematische pijlers, waarbij elk hoofdstuk bijdraagt aan een of meerdere pijlers. Samen ondersteunen deze pijlers het overkoepelende doel: het bouwen van veiligere, praktisch toepasbare en beter generaliseerbare ML-NIDS'en.

De eerste pijler, *AI for Cybersecurity*, richt zich op het toepassen van ML binnen het domein van intrusiedetectie. Aan het begin van mijn doctoraat werden verschillende zogenaamde moderne academische benchmarkdatasets gepubliceerd. De eerste studies rapporteerden bijna perfecte classificatieresultaten op deze datasets, voornamelijk behaald met *supervised* ML-technieken. Latere validatie-experimenten, waarbij de veelbelovende modellen getest werden op data niet behorende tot de trainingsdataset, toonden echter aanzienlijke prestatieverliezen aan. Dit roept twijfels op over de eerder veronderstelde generalisatiekracht van deze modellen. Hoofdstuk 2 onderzoekt of *unsupervised* en *self-supervised* ML-modellen vergelijkbaar hoge classificatieprestaties kunnen behalen als *supervised* methoden op deze moderne academische datasets, en of ze evenzeer lijden onder een gebrek aan generalisatie. Om de generalisatiekracht te testen, wordt

een nieuwe *inter-dataset* evaluatiestrategie geïntroduceerd, waarbij modellen worden getraind en gevalideerd op één dataset, en getest op een tweede, voorheen ongebruikte dataset. Hoewel de modellen bijna perfecte prestaties behalen op data uit de trainingsdataset, slagen ze er niet in om vergelijkbare resultaten te behalen op de tweede dataset. De AUROC-scores dalen tot wel 37%, wat het generalisatieprobleem van ML-NIDS bevestigt. Om de prestaties te verbeteren, stelt Hoofdstuk 3 een meerlagige ML-NIDS-architectuur voor die anomaliedetectie, *multiclass-classificatie* en detectie van *zero-day* aanvallen combineert. Deze aanpak benut de sterktes van zowel *supervised* als *unsupervised* leren, maakt een modulaire implementatie mogelijk en is in staat om zowel gekende als ongekende aanvallen te detecteren. In vergelijking met een enkelvoudig ML-NIDS-model biedt de architectuur verbeterde classificatieprestaties, alsook extra voordelen voor hiërarchische installaties, zoals een lager bandbreedtegebruik en verbeterde privacy bescherming. Een belangrijke bevinding binnen deze eerste pijler is dat het generalisatieprobleem niet alleen voortkomt uit het modelontwerp, maar vooral uit de kwaliteit van de trainingsdata. Deze realisatie leidde tot de ontwikkeling van ConCap, een framework dat het mogelijk maakt om precies en herhaalbaar netwerkverkeer te genereren en automatisch het verkeer aggregeert in network connecties met een overeenkomstig label. Hoofdstuk 6 toont aan dat ML-modellen die getraind zijn op netwerkverkeer dat met ConCap is gegenereerd, kunnen generaliseren naar zowel academische benchmarkdatasets als echte netwerkomgevingen. Hiermee is het generalisatieverhaal rond—beginnend met twijfels over de optimistische resultaten op academische datasets, en eindigend met een praktisch framework om de oorzaak van het probleem aan te pakken, de kwaliteit en diversiteit van de data.

De tweede pijler, *Research to Practice*, behandelt de kloof tussen academische ML-NIDS-prototypes en hun inzet in reële omgevingen. Ondanks veelbelovende resultaten in de literatuur blijven experts in de industrie vaak sceptisch, mede door het gebrek aan praktische toepasbaarheid. Bovendien vereist een reële implementatie schaalbare en efficiënte systemen die de complexiteit van productieomgevingen aankunnen. Hoofdstuk 4 introduceert ChronosGuard, een hiërarchisch, cloud-native ML-NIDS gebaseerd op de meerlagige architectuur dat voorgesteld werd in Hoofdstuk 3. Uitgebreide experimenten met Kubernetes onderzoeken systematisch hoe verschillende uitrolstrategieën, orkestratiealgoritmes en netwerktopologieën invloed hebben op de prestaties en het gebruik van computerbronnen, zoals CPU en geheugen. De resultaten bieden experimenteel inzicht in de praktische afwegingen en tonen aan dat een hiërarchische implementatie en een netwerkbewuste orkestratiealgoritme de operationele prestaties van ML-NIDS aanzienlijk kunnen verbeteren. Een ander cruciaal onderdeel van een praktische ML-NIDS implementatie is de *preprocessing* pijplijn, met name de extractie van kenmerken op netwerkconnectie niveau uit netwerkpakketten. Hoofdstuk 5 presenteert een systematische literatuurstudie en empirisch vergelijkende studie van de meest gebruikte tools in ML-NIDS onderzoek. De resultaten tonen een gefragmenteerd ecosysteem, dat sterk leunt op tools waarbij het vaak aan standaardisatie, prestaties en reproduceerbaarheid ontbreekt. Om deze beperkingen te overwinnen, wordt RustiFlow geïntroduceerd: een snelle, op Rust gebaseerd netwerk *feature-extractor* gebouwd met eBPF. RustiFlow ondersteunt zowel realtime als offline verkeersanalyse, meerdere featuresets en aanpasbare configuraties. Het is ontworpen om te voldoen aan zowel de academische als operationele vereisten en brengt de noden samen voor academische experimenten en praktische toepassingen in de echte wereld.

De derde pijler, *Cybersecurity for AI*, onderzoekt de vijandige context waarin IDS'en opereren. Hoe-

wel *adversarial* ML uitvoerig is bestudeerd in domeinen zoals computervisie, werken ML-NIDS'en onder een strenger bedreigingsmodel. Een realistische aanname is dat aanvallers de ingangsvectoren van het ML-model niet direct kunnen manipuleren, en in plaats daarvan ontwijkingsstrategieën moeten ontwikkelen door de netwerkpakketten aan te passen die hun systemen genereren. Hoofdstuk 6 formaliseert een nieuwe klasse van *adversarial* aanvallen, genaamd *host-space perturbations*. Dit zijn kleine, realistische aanpassingen aan de handelingen van een aanvaller die hetzelfde doel bereiken, maar mogelijks variaties veroorzaken in de gegenereerde netwerkpakketten die detectie door ML-NIDS kunnen omzeilen. Een praktische evaluatie van deze *host-space perturbations* toont aan dat zelfs minimale aanpassingen, zoals het wijzigen van één karakter in een aanvalscommando, state-of-the-art ML modellen consequent kunnen omzeilen. Deze bevindingen benadrukken dat praktische ontwijking door realistische aanvallers mogelijk is, en dat ML-NIDS geëvalueerd moeten worden aan de hand van meer realistische bedreigingsmodellen.

Vooruitkijkend naar de toekomst, komen er uit dit werk verschillende veelbelovende onderzoeksrichtingen naar voren. Het verbeteren van de kwaliteit van datasets blijft een fundamentele opdracht, en ConCap biedt een manier om divers, reproduceerbaar en correct gelabeld netwerkverkeer te genereren die de tekortkomingen van bestaande benchmarks aanpakt. Verder kan de robuustheid tegen *adversarial* aanvallen verder worden onderzocht door het *inverse feature-mapping* probleem te analyseren en de impact van verschillende *feature extractors* op de veiligheid van ML modellen te evalueren. Het verbeteren van generalisatie vereist zowel datagestuurde augmentatie strategieën als architecturale vernieuwing, zoals het leren van netwerk- en besturingssysteemonafhankelijke representaties van netwerkverkeer. Tot slot kan het gebruik van complexere ML-modellen met grotere leercapaciteit verder worden onderzocht zodra er voldoende publieke datasets met kwalitatieve netwerkdata beschikbaar zijn.

Tot slot draagt dit proefschrift niet alleen bij aan de ontwikkeling van meer generaliseerbare, praktische en veilige ML-NIDS—onder meer via tools zoals RustiFlow en ConCap—maar pleit het ook voor een mentaliteitsverandering in hoe ML-NIDS geëvalueerd, geïmplementeerd en beveiligd worden. Door fundamentele problemen aan te pakken op het vlak van generalisatie, toepasbaarheid en veiligheid, legt het de basis voor de volgende generatie ML-NIDS die klaar zijn voor gebruik in de echte wereld.

Summary

In today's modern society, digital systems are deeply integrated into nearly every sector, from healthcare and finance to energy and transportation. While these interconnected digital systems offer many societal benefits, they also create a significantly expanded attack surface for malicious actors. Network intrusion detection systems (NIDSs) have long served as an essential line of defense against such threats. However, traditional NIDSs face growing limitations when dealing with novel threats, encrypted traffic, and rapidly evolving attack patterns—especially those that rely on manually created signatures and detect known attack behaviors through pattern matching on packet payloads. To address these shortcomings, the cybersecurity community has increasingly explored machine learning–based NIDSs (ML-NIDSs). Despite considerable academic interest since the 1980s, the adoption of ML-NIDSs remains limited.

This dissertation investigates the challenges that prevent ML-NIDSs from transitioning from academic research into practical operational adoption. These challenges are explored through five research questions, which examine (1) the classification performance of unsupervised ML models, (2) the generalization strength of ML-NIDSs, (3) the design of distributed ML-NIDS for cloud environments, (4) the practical issues preventing real-world deployment, and (5) the feasibility to evade a ML-NIDS using realistic threat models. To structure the exploration of the research questions, the dissertation is organized around three thematic pillars, with each chapter contributing to one or more pillars. Together, these pillars support the overarching goal to build more secure, practical, and generalizable ML-NIDS.

The first pillar, *AI for Cybersecurity*, focuses on applying ML to the domain of intrusion detection. At the beginning of my PhD, several new so-called modern academic benchmark datasets were released. Initial studies reported near-perfect classification performance on these datasets, mainly obtained using supervised ML techniques. However, subsequent validation experiments testing the promising models outside the training dataset revealed significant performance drops, questioning the ML models' prior assumed generalization. Chapter 2 evaluates if unsupervised and self-supervised ML models can obtain similar high classification performance as supervised methods on these modern academic datasets and test if they suffer from the same lack of generalization. To validate the generalization strength, a novel *inter-dataset* evaluation strategy is introduced, in which models are trained and validated on one dataset and tested on a second, previously unseen dataset. While the models achieve near-perfect classification performance within the training dataset, they fail to generalize to the second dataset, with AUROC scores dropping up to 37%, confirming the significant generalization gap of ML-NIDS. To improve performance, Chapter 3 proposes a multi-stage ML-NIDS architecture that combines anomaly detection, multi-class classification, and zero-day attack detection. This approach leverages the strengths of supervised

and unsupervised learning, enables modular deployment, and is capable of detecting both known and unknown, zero-day attacks. Compared to a single-model ML-NIDS, the architecture achieves improved classification performance while offering additional benefits for hierarchical deployments, such as reduced bandwidth usage and improved privacy preservation. A key insight from this pillar is that the generalization problem may stem not only from model design but mainly from the quality of the training data. This realization motivated the development of ConCap, a containerized framework that enables fine-grained, reproducible network traffic generation with automated network flow labeling. Chapter 6 demonstrates that ML models trained on network traffic generated with ConCap can generalize to both academic benchmark datasets and real-world network environments. This brings the generalization story full circle—starting from doubts about overly optimistic results on academic datasets and concluding with a practical, reproducible framework to address the root cause.

The second pillar, *Research to Practice*, addresses the gap between academic ML-NIDS prototypes and deployment in real-world environments. Despite promising results in the literature, industry practitioners often remain skeptical due to the lack of practical considerations. Furthermore, real-world deployment demands scalable and efficient systems capable of handling the complexity of production environments. Chapter 4 introduces ChronosGuard, a hierarchical, cloud-native ML-NIDS based on the multi-stage architecture proposed in Chapter 3. Extensive experiments using Kubernetes systematically explore how different deployment strategies, scheduling algorithms, and network configurations affect performance and resource utilization. The results provide empirical insights into practical trade-offs and demonstrate that hierarchical deployments and network-aware schedulers can significantly improve the operational performance of ML-NIDS. Another critical component of practical ML-NIDS deployment is the preprocessing pipeline—particularly the extraction of flow-level features from raw network traffic. Chapter 5 presents a systematic literature review and empirical benchmarking study of the most commonly used tools in ML-NIDS research. The results reveal a fragmented ecosystem, reliant on custom tools that lack standardization, performance, and reproducibility. To overcome these limitations, RustiFlow is introduced, a high-performance, Rust-based network flow feature extractor built using eBPF. RustiFlow supports both real-time and offline traffic analysis, multiple feature sets, and custom configuration. It is designed to meet academic and operational requirements, bridging the gap between academic experimentation and practical, real-world deployments.

The third pillar, *Cybersecurity for AI*, explores the adversarial context in which IDSs operate. While adversarial machine learning has been extensively studied in domains such as computer vision, ML-NIDSs operate under a more constrained threat model. A realistic assumption is that attackers can not directly manipulate the input vectors of the ML model and must instead create evasion strategies by altering the raw network traffic generated by the systems under their control. To reflect this constraint, Chapter 6 formalizes a new class of adversarial attacks called *host-space perturbations*: small, realistic modifications to an attacker's actions achieving the same goal but potentially causing variations in the resulting network traffic that may evade ML-NIDS detection. A practical evaluation of host-space perturbations demonstrates that even minor modifications, such as changing a single character in the attack command, can evade state-of-the-art classifiers consistently. These findings highlight that practical evasion is possible by a realistic attacker and the need for ML-NIDS to be evaluated using more realistic threat models.

Looking forward, several promising research directions emerge from this work. Improving dataset quality remains a fundamental task, and ConCap offers a path to generating diverse, reproducible, and well-labeled traffic that addresses many flaws of existing benchmarks. Similarly, adversarial robustness can be further explored by investigating the inverse feature-mapping problem and evaluating the impact of different feature extractors on model security. Enhancing model generalization requires both data-driven augmentation strategies and architectural innovation, such as learning network and operating system invariant traffic representations. Finally, the adoption of more complex ML models with more learning power can be explored when sufficient high-quality network data becomes publicly available.

In conclusion, this dissertation contributes not only with advancements toward more generalizable, practical, and secure ML-NIDS—through the development of tools such as RustiFlow and ConCap—but also by advocating a shift in mindset for evaluating, deploying, and securing ML-NIDS. By addressing fundamental issues in generalization, practicality, and security, it lays the foundations for the next generation of ML-NIDS that are ready for real-world environments.

1 Introduction

This chapter situates the conducted research work within the broader scientific context, presents the main research questions, and outlines the structure of this dissertation. It also provides an overview of the research publications and software tools that were authored during this period. Positioned at the intersection of artificial intelligence (AI) and cybersecurity, this dissertation focuses on the application of machine learning (ML) techniques to network intrusion detection (NID). In particular, it explores the key challenges of ML-based intrusion detection systems from three complementary perspectives, with the overarching goal of advancing towards more generalizable, practical, and secure machine learning-based network intrusion detection systems (ML-NIDS).

1.1 The Evolving Cybersecurity Landscape

The modern world is becoming increasingly dependent on interconnected digital systems, spanning every sector—from healthcare and finance to transportation and energy. This digital transformation is driven by technological advances in computer networks, the exponential growth in the number of connected devices, and the expanding diversity of hardware, software, and networking protocols. As a result, today's networks are vast, heterogeneous, and more complex than ever before [1, 2]. The rapid digitalization of society brings many benefits, such as increased availability and efficiency, but it also raises the cybersecurity risk on two fronts: by expanding the attack surface and by negatively affecting the potential impact of successful attacks [3, 4].

In this digitized world, cybersecurity has become a fundamental concern. Attacks can compromise

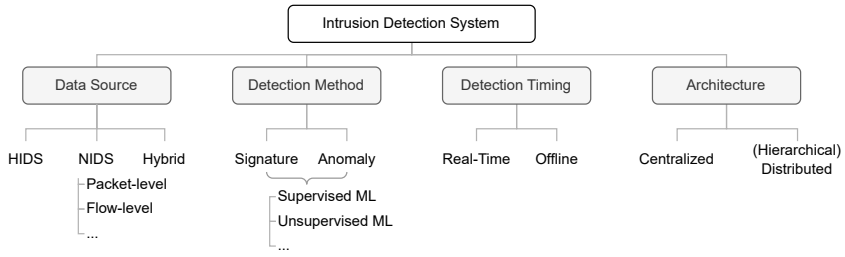


Figure 1.1: Taxonomy of intrusion detection systems.

the confidentiality, integrity, and availability of digital assets, including sensitive data, systems, and communication networks [5, 6, 7]. Common examples include distributed denial-of-service (DDoS) attacks that target system availability, brute-force login attempts that compromise confidentiality, and man-in-the-middle (MITM) attacks that violate both integrity and confidentiality by hijacking secure communications.

To protect against these threats, organizations employ a layered defense strategy encompassing a wide range of tools and practices aimed at identifying, protecting, detecting, responding to, and recovering from cyber incidents. It has long been acknowledged within the computer security community that full and provable protection of information and communication technology (ICT) infrastructure is practically infeasible—if not impossible [8, 9]. This realization has driven the development of detection tools, which are designed to identify ongoing or past intrusions rather than merely prevent them.

Among these tools, one of the most essential is the intrusion detection system (IDS). An IDS plays a critical role in monitoring network or system activities for signs of malicious behavior, offering organizations a way to detect and respond to threats as early as possible [10, 11].

1.2 Intrusion Detection Systems (IDSs)

An IDS is a security mechanism designed to identify signs of potential misuse or malicious activity within a system or network. When suspicious behavior is detected, the IDS raises an alert, which can be propagated to security analysts or integrated into a Security Information and Event Management (SIEM) system [12, 13]. SIEM platforms are typically used in larger environments to aggregate and correlate alerts from diverse sources—such as IDSs, firewalls, antivirus software, server logs, and network traffic data—providing a centralized view of ongoing threats and enabling incident response teams to act with improved situational awareness [14, 15].

IDSs—see Figure 1.1 for a taxonomy—can be categorized based on their data sources and deployment location. Host-based IDS (HIDS) monitor data local to a specific device, such as system logs, resource utilization, system calls, and the host's network traffic [16, 17, 18]. In contrast, Network-based IDS (NIDS) analyze network traffic [19], typically at strategic points such as routers

or switches, to detect suspicious patterns. Hybrid approaches also exist, combining host-level insights with broader network visibility [20]. The effectiveness of an IDS depends on its placement and the type of input data it processes. HIDS are limited to protecting the host on which they are deployed but can access fine-grained telemetry. NIDS, on the other hand, offer broader visibility over entire network segments but may lack detailed host-level context. Regardless of the data source, IDSs can be deployed in a centralized architecture—where data is forwarded to a central analysis point—or distributed across multiple locations, possibly in a hierarchical structure.

IDSs can operate in real-time, flagging threats as they occur, or in offline mode, analyzing stored data retrospectively [21]. Real-time systems provide faster responses but face higher performance constraints, while offline systems can perform deeper analysis with relaxed time constraints but delay detection of potential intrusions.

At a high level, IDSs rely on two detection methods: signature-based and anomaly-based detection. Traditional signature-based IDSs [22] identify threats by comparing observed data to a knowledge base of known attack signatures. One of the earliest open-source signature-based detector is Snort [23], and later Suricata [24]. This approach can detect known attacks with high confidence and demonstrates low false-positive rates. However, they struggle against zero-day attacks or obfuscated threats that do not match existing patterns [25]. Additionally, the manual effort required to develop and maintain up-to-date signatures is labor-intensive and increasingly unsustainable [26]. The widespread adoption of encryption protocols such as HTTPS renders packet payloads opaque, making deep packet inspection (DPI) infeasible and significantly reducing the visibility of network-based detectors [27, 28]. In contrast, anomaly-based IDSs[29] establish a baseline of normal behavior and flag deviations as potential threats. One example tool is Zeek [30], formerly known as Bro [31]. While this approach is capable of detecting novel and unforeseen attacks, it is prone to high false-positive rates, especially in dynamic environments where the definition of “normal” can shift over time. As highlighted by Sommer and Paxson[32], the trade-off between detection coverage and precision remains a continuous challenge.

To address these limitations, the cybersecurity community has increasingly turned to data-driven methods, particularly those based on machine learning (ML) and artificial intelligence (AI).

1.3 The Shift Towards Data-Driven Intrusion Detection

The success of ML in domains like computer vision and natural language processing has inspired its application across cybersecurity tasks, including malware detection [33], phishing recognition [34], threat intelligence [35], and traffic classification [36]. Intrusion detection, in particular, was one of the earliest areas in cybersecurity to explore ML-based approaches, dating back to the seminal work of Denning in the 1980s [21, 37]. Since then, it has remained a highly active research field, with thousands of scientific papers published on the topic.

This dissertation focuses on intrusion detection systems that analyze network traffic as the primary data source for detecting threats. NIDSs are particularly well-suited for this task, as the

network is the most common way for an intruder to enter the target infrastructure or system and is used by attackers to lateral move from one system to another.

A machine learning–based NIDS uses data-driven techniques to learn patterns and characteristics from previously observed network behavior. For misuse detection, ML models can be trained on labeled datasets of malicious and benign traffic to recognize known threats or close variants. For anomaly detection, models learn a representation of "normal" behavior and raise alerts when deviations are observed. This flexibility allows anomaly-based Network IDSs (NIDSs) to operate without needing predefined attack signatures. ML methods used in IDSs range from classical algorithms like Decision Trees, Support Vector Machines (SVM), and Isolation Forests to more recent deep learning models. The learning process can be supervised [38], requiring labeled examples of attacks, or unsupervised/self-supervised [39], which relies on modeling normal behavior from unlabeled or partially labeled data. Each approach comes with trade-offs in terms of data requirements, detection capabilities, and real-world applicability [40].

A NIDS may inspect traffic in various forms: at the packet level, flow level, or using multi-flow contexts [8]. While packet-level inspection allows detailed analysis, it is often impractical in high-speed networks due to the overwhelming volume of data, storage limitations, and privacy concerns [41]. Moreover, DPI becomes challenging when payloads are encrypted [42]. Instead, NIDSs often rely on flow-level analysis, which aggregates packets into flows based on the five-tuple (source and destination IP addresses, ports, and protocol). Each flow summarizes the interaction between two hosts over time and captures features such as start and end timestamps, packet and byte counts, inter-arrival times, and selected header attributes [43]. This high-level metadata provides rich information about communication patterns without exposing packet contents, significantly reducing data size and preserving privacy. Flow-based NIDSs are thus a scalable and efficient solution for detecting intrusions in encrypted, high-throughput networks [44].

Despite strong academic interest, the practical adoption of ML-based IDSs remains limited. A significant gap persists between research and practice, driven by issues such as limited generalizability, lack of high-quality data, and unrealistic ML evaluation. [19, 45, 46, 47, 48, 49].

1.4 Research Questions

At the beginning of this PhD in late 2019, the ML-based intrusion detection research landscape was largely dominated by studies focusing on model-centric approaches. Most published papers aimed to marginally improve classification performance within narrow experimental setups, often relying on decade-old benchmark datasets with well-documented flaws. While concerns about data quality and the real-world applicability of these models had already been raised in the community, their impact was not yet well understood [32]. As a result, many of these works continued to report promising results without addressing the underlying limitations. One hypothesis was that unsupervised ML approaches might be less susceptible than supervised approaches, but studies mostly focused on the latter. This led to the formulation of my first research question:

RQ 1. *Can unsupervised ML-NIDS models achieve near-perfect classification performance on par with supervised methods when trained and evaluated on academic benchmark datasets?*

Early in this dissertation, we started questioning the impact of the promising results on benchmark ML-NIDS datasets published by researchers [50], suspecting that the reported high classification performance was overly optimistic and unlikely to hold in real-world scenarios. At the same time, a new generation of so-called modern NIDS datasets was released [51], offering an opportunity to revisit the generalization problem using a more realistic evaluation strategy. This led to the formulation of the next research question, which was revisited multiple times during this PhD.

RQ 2. *Do ML-based NIDS generalize across heterogeneous network environments?*

Simultaneous to our numerous (unsuccessful) attempts to address the generalization challenge using the available benchmark datasets, such as preventing shortcut learning by removing metadata [52] and non-metadata features [53], training data-constrained models [54], and lowering model complexity [55]. We already started focusing on the practical deployment challenges of ML-NIDS in real-world, large-scale distributed networks, such as cloud native architectures. Leveraging our lab's expertise in cloud-native systems in collaboration with Taiwan's high-speed networking lab at National Yang Ming Chiao Tung University (NYCU) led us to the third research question:

RQ 3. *How can we design and deploy a distributed ML-NIDS suitable for large-scale, real-world deployment in modern cloud infrastructures?*

Through collaborations with industry partners and the deployment of ML-NIDS in practice, we found additional challenges to practical adoption beyond the previously identified generalization challenge. These include the lack of high-performance, flexible tooling for feature extraction, incompatibilities between research and industry workflows, and best practices regarding cloud-native deployments. This motivated a broader investigation into the practical gaps between research prototypes and production-ready ML-NIDS systems.

RQ 4. *What are some challenges, besides the lack of generalization, that may impair the integration of ML-NIDS research prototypes into practice?*

Finally, since ML-NIDS operate in inherently adversarial environments, it is important to evaluate their robustness against adversarial ML attacks, which aim to manipulate models to produce incorrect predictions or decisions by introducing malicious inputs or modifying training data. Unlike domains such as computer vision—where attackers can directly manipulate input vectors—ML-NIDS face unique constraints due to network semantics and the limited access to the internal representation used by the models [56]. In this context, traditional gradient-based attacks are not possible, as attackers cannot directly modify the input features derived from raw network traffic [57]. Therefore, we focus on more realistic threat models and attack strategies, which motivates our final research question:

Table 1.1: Mapping of Research Questions (RQs) to Dissertation Chapters

	RQ1	RQ2	RQ3	RQ4	RQ5
Chapter 2	●	●			
Chapter 3		●	●		
Chapter 4			●	●	
Chapter 5			●	●	
Chapter 6		●			●

RQ 5. *How feasible is it for a realistic attacker to evade an ML-NIDS?*

Table 1.1 presents the mapping between the research questions and the corresponding chapters in which they are investigated in this dissertation.

1.5 Dissertation Outline

The remainder of this dissertation is organized into five chapters, each contributing to one or more of the three thematic pillars illustrated in Figure 1.2. These pillars collectively support the overarching goal of this work: to build secure, practical, and generalizable machine learning-based intrusion detection systems (ML-NIDS). Each of the five chapters is based on a peer-reviewed publication, with the exception of Chapter 6, which is currently under review. For all five chapters, I contributed to all roles as defined by the CRediT author statement–including conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing (original draft and review & editing), and visualization–except for supervision, project administration, and funding acquisition. A final chapter presents the overall conclusions and outlines directions for future work.

1.5.1 AI for Cybersecurity

The first pillar, *AI for Cybersecurity*, explores the application of machine learning techniques to network intrusion detection. At the beginning of this PhD in 2019, several new academic benchmark datasets had been released, such as the now very popular CIC-IDS-2017 and CSE-CIC-IDS-2018 datasets from the Canadian Institute for Cybersecurity (CIC) [58]. Early research using these benchmark datasets mainly focused on supervised approaches and reported promising near-perfect classification accuracy [59]. However, from early on, we suspected these results were too optimistic. Our initial cross-dataset validation experiments confirmed this hypothesis, revealing significant performance drops when models were evaluated beyond their training datasets [50].

Chapter 2 begins by benchmarking unsupervised and self-supervised methods on these benchmark datasets (RQ1). These baselines achieve similar high classification performance, creating the hypothesis that these models are also susceptible to the lack of generalization strength. This

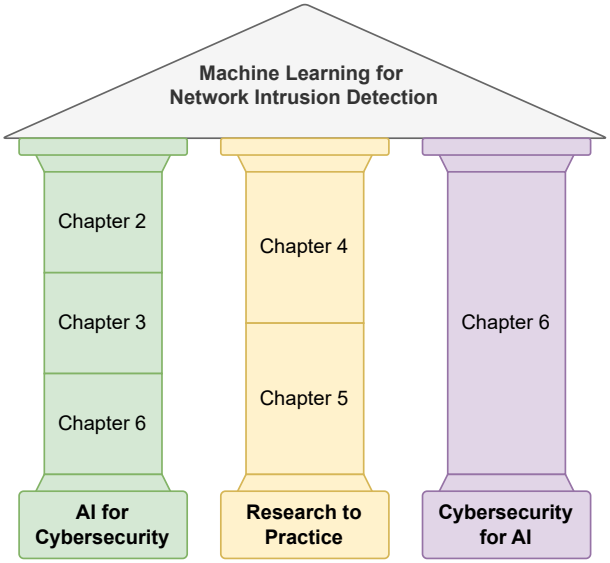


Figure 1.2: Conceptual structure of this dissertation, organized into three thematic pillars. Together, these pillars support the overarching goal of building secure, practical, and generalizable ML-NIDS.

hypothesis is validated using an *inter-dataset evaluation strategy*. Models trained on CIC-IDS-2017 are tested on CSE-CIC-IDS-2018, and vice-versa, assessing the generalization strength of ML-NIDS on samples generated by the same traffic generation methodology but different underlying network infrastructure (RQ2). The results confirm the hypothesis by showing a significant drop in classification performance when tested outside the trained dataset. Unsupervised and self-supervised models are interesting for their ability to learn from unlabeled or benign-only data and detect anomalies like zero-day attacks as long as they deviate sufficiently from benign. Since it is difficult to work with real zero-day attacks—by definition, once discovered, they are no longer zero-day—previously unseen attack classes are used as a proxy. These samples belong to entirely different attack classes from those encountered during training. However, these models miss the fine-grained classification capabilities to assign more granular labels to the input, e.g., classifying the malicious network traffic as a port scanning or denial-of-service (DoS) attack.

Chapter 3 builds on these insights by designing a multi-stage ML-NIDS that combines the strengths of unsupervised and supervised learning. The proposed system consists of three sequential stages: an anomaly detector to filter out benign traffic, a multi-class classifier to identify known threats, and an extension stage that re-evaluates uncertain samples to detect potential zero-day attacks. We evaluated whether this multi-stage architecture offers performance improvements over a supervised single-model ML-NIDS, evaluated the generalization strength of the zero-day detection capabilities (RQ2), and explored its suitability for hierarchical deployments in privacy-sensitive, bandwidth-constrained environments (RQ3).

Chapter 6, the last conceptual chapter of this dissertation, revisits the generalization challenge of ML-NIDS, this time addressing it from a data quality perspective rather than model design (RQ2). Closer to the end of my PhD, multiple independent studies performed thorough analysis of the academic benchmark datasets for ML-NIDS [46, 47, 48, 49] and identified multiple issues or bad design smells as defined by Flood et al [49], such as wrong labels, poor data diversity, and traffic collapse. Moreover, Engelen et al [46] performed a thorough analysis of the CIC-IDS-2017 and 2018 datasets used in earlier chapters and fixed multiple mistakes in both the used flow feature extractor, CICFlowMeter [60], and the labeling logic. To address these dataset challenges, we propose ConCap, a framework for realistic network traffic generation with fine-grained control and automated labeling. It allows researchers to specify attacker-target interactions using containerized environments, configure resource constraints, and set up networking characteristics in a structured, reusable format. As a result, experiments can be shared and rerun with minimal setup, supporting reproducibility and enabling detailed analysis of the traffic generation process. Results show that ML models trained on network traffic generated with ConCap generalize perfectly to both real-world and academic target networks—provided that the network characteristics simulated in ConCap resemble those of the target environment.

1.5.2 Research to Practice

The second pillar, *Research to Practice*, explores the persistent gap between academic research and real-world adoption of ML-NIDS. A recent SoK [61] highlights practitioners' skepticism towards ML-based NIDS, citing challenges such as the inability to transfer the optimistic research results into operational environments and ignoring the pragmatic aspects of operational deployment. We already explore the first cause in the first pillar of this dissertation, namely the optimistic performance claims, by questioning the reported near-perfect classification performance and evaluating the generalization strength of ML-NIDS using our proposed *inter-dataset evaluation strategy*. This pillar focuses on the second cause, the practical aspects of real-world deployment of ML-NIDS, not only by making our work reproducible by making the research artifacts available (see Section 1.7) but also by looking into how to create large-scale practical deployments of the proposed research approaches in modern clouds.

Chapter 4 packages the multi-stage ML-NIDS introduced in Chapter 3 into a scalable, containerized system called ChronosGuard, and evaluates its performance in modern cloud environments. Through extensive empirical experimentation, we systematically analyze how various factors, such as the network topology, workload orchestrator, and deployment strategies, affect classification performance and resource efficiency.

Chapter 5 analyses and proposes a solution to an additional hurdle in the practical adoption we identified from our interactions with industry partners, namely the preprocessing stage—specifically, the extraction of statistical network flow features. Existing research tools lack the performance required to monitor high-speed network connections in real-time or to process large network traffic captures. On the contrary, industry tools were originally designed for monitoring

use cases and lack the flexibility required for research purposes. First, the literature review in this chapter confirmed this practical experience and revealed that over half of research papers use a custom network flow feature extractor. This lack of a standardized approach makes adoption more difficult. Moreover, each network flow exporter provides a unique feature set, causing ML models trained using one flow exporter to be incompatible with data exported by other tools [62]. Secondly, the performance of the most used operational and research tools used by researchers is benchmarked for both real-time and offline feature extraction. Lastly, RustiFlow is presented, a high-performance, flexible flow exporter capable of both real-time and offline processing. Additionally, RustiFlow supports multiple feature sets and allows custom feature definitions, providing a tool aimed at both researchers and practitioners.

1.5.3 Cybersecurity for AI

The final pillar, *Cybersecurity for AI*, explores the security of ML-NIDS themselves. This is important to consider since cybersecurity applications inherently operate in an adversarial environment and, therefore must be robust against any kind of attack, such as evasion attacks or data poisoning. Historically, adversarial attacks against ML systems originate from the computer vision domain [63], NID presents fundamentally different challenges. Unlike in image recognition, where attackers can manipulate input pixels directly, attackers in network security typically cannot access or alter the feature vectors used by ML models. For instance, flow-based ML-NIDS, as studied in the two previous pillars, preprocesses raw network traffic into network flows by computing statistical features from the data exchanged between source and destination. When using a realistic threat model, an attacker cannot access the internal system computing these statistical features or the ML-NIDS system itself to adjust the input vectors before classification [56]. Moreover, since this transformation is a one-way function, it becomes impossible to first craft adversarial examples in feature space using gradient-based methods like FGSM and then transform them back to network traffic. A more realistic approach is to craft the adversarial examples in the problem space by perturbing the raw network packets themselves, such as adding padding, fragmenting packets, or changing the timing between packets. However, there are multiple challenges associated with perturbing the packets directly, such as maintaining the attack semantics, constraints of the network protocols, and not all network traffic transformation will result in a change to the corresponding statistical features of a network flow [57].

Chapter 6 investigates evasion attacks from the perspective of a realistic attacker (RQ5). We introduce and formalize a new category of adversarial ML attacks simulated through *host-space perturbations*–changes to attacker actions that may lead to significant deviations in the generated network traffic. These attacks are both practical and effective, requiring the attacker to modify as little as a single parameter to evade detection by state-of-the-art ML-NIDS.

1.6 Publications

The results of the research during this PhD have been published in scientific journals and presented at different international conferences. This section provides an overview of these publications.

1.6.1 Publications in International Journals

1. L. D'hooge, **M. Verkerken**, T. Wauters, B. Volckaert, and F. De Turck. **Hierarchical feature block ranking for data-efficient intrusion detection modeling.** *Published in Computer Networks*, Article 108613, 2021.

2. **M. Verkerken**, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. **Towards model generalization for intrusion detection: Unsupervised machine learning techniques.** *Published in Journal of Network and Systems Management (JNSM)*, pages 1-25. 2022.

3. Y. Lai, D. Sudyana, Y. Lin, **M. Verkerken**, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. **Task assignment and capacity allocation for ml-based intrusion detection as a service in a multi-tier architecture.** *Published in IEEE Transactions on Network and Service Management (TNSM)*, pages 672-683. 2022.

4. L. D'hooge, **M. Verkerken**, T. Wauters, F. De Turck, and B. Volckaert. **Investigating generalized performance of data-constrained supervised machine learning models on novel, related samples in intrusion detection.** *Published in Sensors*, Article 1846, 2023.

5. **M. Verkerken**, L. D'hooge, D. Sudyana, Y. Lin, T. Wauters, B. Volckaert, and F. De Turck. **A novel multi-stage approach for hierarchical intrusion detection.** *Published in IEEE Transactions on Network and Service Management (TNSM)*, pages 3915-3929. 2023.

6. L. D'hooge, **M. Verkerken**, T. Wauters, F. De Turck, and B. Volckaert. **Characterizing the impact of data-damaged models on generalization strength in intrusion detection.** *Published in Journal of Cybersecurity and Privacy (JCP)*, pages 118-144. 2023.

7. D. Sudyana, Y. Lin, **M. Verkerken**, R. Hwang, Y. Lai, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. **Improving Generalization of ML-Based IDS With Lifecycle-Based Dataset, Auto-Learning Features, and Deep Learning.** *Published in IEEE Transactions on Machine Learning in Communications and Networking (TMLCN)*, pages 645-662. 2024.

8. T. Goethals, M. Sebrechts, **M. Verkerken**, F. De Turck, and B. Volckaert. **Mixed-runtime Pod Networking for Kubernetes-based Edge Computing.** *Submitted for review, 2025.*

1.6.2 Publications in Proceedings of International Conferences

1. **M. Verkerken**, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. **Unsupervised machine learning techniques for network intrusion detection on modern data.** *Published in Proceedings of the 4th Cyber Security in Networking Conference (CSNet)*, pages 1-8. 2020, IEEE.

2. Y. Lai, D. Sudyana, Y. Lin, **M. Verkerken**, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. **Machine learning based intrusion detection as a service: Task assignment and capacity allocation in a multi-tier architecture.** *Published in Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, 2021.

3. L. D'hooge, **M. Verkerken**, B. Volckaert, T. Wauters, and F. De Turck. **Establishing the contaminating effect of metadata feature inclusion in machine-learned network intrusion detection models.** *Published in Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Springer, 2022.

4. L. D'hooge, **M. Verkerken**, T. Wauters, B. Volckaert, and F. De Turck. **Discovering non-metadata contaminant features in intrusion detection datasets.** *Published in Proceedings of the 19th Annual International Conference on Privacy, Security & Trust (PST)*, 2022, pages 1-11. IEEE.

5. **M. Verkerken**, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. **Poster: Fixing the Foundations: Towards Generalization for Machine Learning Intrusion Detection Systems** *Presented at Faculty of Engineering and Architecture Research Symposium (FEARS 2022) and Network and Distributed System Security (NDSS) Symposium 2023.*

6. L. D'hooge, **M. Verkerken**, T. Wauters, F. De Turck, and B. Volckaert. **Castles built on sand: Observations from classifying academic cybersecurity datasets with minimalist methods.** *Published in Proceedings of the 8th International Conference on Internet of Things, Big Data and Security (IoTBDS)*, 2023, SCITEPRESS.

7. J. Santos[*], **M. Verkerken**[*], L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. **Performance Impact of Queue Sorting in Container-Based Application Scheduling.** *Published in Proceedings of the 19th International Conference on Network and Service Management (CNSM)*, 2023, IEEE.

8. D. Sudyana[*], **M. Verkerken**[*], L. D'hooge, Y. Lin, R. Hwang, Y. Lai, F. Yudha, T. Wauters, B. Volckaert, and F. De Turck. **Quality Analysis in IDS Dataset: Impact on Model Generalization.** *Published in Proceedings of the IEEE Conference on Communications and Network Security (CNS)*, 2024, IEEE.

9. **M. Verkerken**[*], J. Santos[*], L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. ChronosGuard**: A Hierarchical Machine Learning Intrusion Detection System for Modern Clouds.** *Published in Proceedings of the 20th International Conference on Network and Service Management (CNSM)*, 2024, IEEE.

10. **M. Verkerken**, M. Callewaert, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck. RustiFlow**: Bridging the Gap Between Security Research and Practice using eBPF-based Network Flow Extraction.** *Accepted at IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2025*

11. S. Bettaieb, L. D'hooge, C. Bertrand Van Ouytsel, A. Legay, E. Rivière, **M. Verkerken**, and B. Vol-

ckaert. **Simplicity Performs, But Should It? Examining Malware Detection Benchmark Datasets.** *Submitted for review, 2025.*

12. **M. Verkerken**, L. D'hooge, B. Volckaert, F. De Turck, and G. Apruzzese. ConCap**: Enabling Fine-Grained Network Traffic Generation for Security Assessments of Flow-based Intrusion Detection Systems.** *Submitted for review, 2025.*

* Authors contributed equally.

1.7 Available and Reproducible Research

Research artifacts developed as part of this PhD have been published using open source licenses (i.e. MIT). This includes source code, scripts, datasets, models, test suites, benchmarks, and any other material underlying the paper's contributions. The following list provides an overview of all public code repositories created as part of this PhD.

In addition to making research artifacts openly available, I contributed to the broader reproducibility movement in cybersecurity research by serving on the Artifact Evaluation Committee for NDSS 2024. In this role, I helped assess the quality, completeness, and reproducibility of submitted research artifacts, promoting the reproducibility of experimental results and the dissemination of artifacts to benefit the community as a whole.

1. Code to train ML models, reproduce results, and recreate visualizations from Chapter 2.
   `https://gitlab.ilabt.imec.be/mverkerk/cic-ids-2018`.
2. Multi-stage hierarchical IDS pipeline implementation, including models, training scripts, test data, and evaluation code for Chapter 3.
   `https://gitlab.ilabt.imec.be/mverkerk/multi-stage-hierarchical-ids`.
3. Code to reproduce experiments and analysis from Chapter 4, including analysis scripts, containers, and deployment files for K8s.
   `https://github.com/idlab-discover/ChronosGuard`.
4. RustiFlow: high-performance flow exporter built with Rust and eBPF, used in Chapter 5.
   `https://github.com/idlab-discover/RustiFlow`.
5. ConCap framework for generating realistic, labeled network traffic using controlled, containerized environments for NIDS research, as used in Chapter 6.
   `https://github.com/idlab-discover/ConCap`.
6. Container images and Dockerfiles for attacker and target environments for ConCap used in Chapter 6.
   `https://github.com/idlab-discover/image-concap`.

1.8 Awards and Grants

Parts of the research conducted during this PhD have received awards and grants:

1. **Best Paper Award IoTBDS 2023**: The paper "Castles Built on Sand: Observations from Classifying Academic Cybersecurity Datasets with Minimalist Methods" [55] received this award at the 8th International Conference on Internet of Things, Big Data and Security.
2. **Student Travel Grant CNSM 2024**: Miel Verkerken was awarded this grant for the paper "ChronosGuard: A Hierarchical Machine Learning Intrusion Detection System for Modern Clouds" [64] at the 20th International Conference on Network and Service Management.
3. **FWO Grant for Long Stay Abroad (V450224N)**: Miel Verkerken was awarded this grant to support his research stay in the summer of 2024 at the University of Liechtenstein, where he worked under the supervision of Assistant Professor Giovanni Apruzzese at the Hilti Chair of Data and Application Security.

1.9 Educational Involvement

Throughout the course of this PhD, I have been actively involved in teaching and student supervision, contributing to the development and teaching of educational content in information engineering technology and computer science programs.

Teaching Activities

I was responsible for preparing, teaching, and evaluating computer lab sessions for the following Bachelor and Master courses:

- **Network Security (E008710)** 2024–2025
- **Server-Side Application Frameworks (E761038)** 2020–2025
- **Object-Oriented Programming (E702050)** 2019–2025
- **Software Engineering (E761035)** 2019–2025
- **Web Technologies (E761026A)** 2019-2020 (discontinued)

In addition, I served as a member of the Education Committee for Information Engineering Technology, where I contributed to discussions on curriculum development and course evaluations.

Master Thesis Supervision

I supervised a total of 15 Master's theses on topics related to intrusion detection and machine learning. Ten of these have been successfully defended, while five are currently ongoing in the

2024–2025 academic year.

- Niels Hauttekeete: *Evaluation of open source big data processing frameworks for intrusion detection* (2020–2021)
- Dylan De Roe: *Evaluation of machine learning frameworks for near real-time intrusion detection* (2021–2022)
- Othello Clemens: *Capturing realistic brute-force and DoS attacks with high variability for dataset creation in intrusion detection using controlled, containerized environments* (2022–2023)
- Arthur Isaac: *Investigating the impact of federated learning on generalization of ML-based intrusion detection* (2022–2023)
- Raman Talwar: *Exploring Explainable AI techniques for detecting contamination features in ML-based intrusion detection* (2022–2023)
- Geert-Jan Van Nieuwenhove: *Generating DDoS attack data for dataset creation* (2022–2023)
- Matisse Callewaert: *Real-time adaptive feature extraction for ML-based network intrusion detection* (2023–2024)
- Charly Moerdyk: *Capturing realistic DDoS attacks in containerized environments for dataset creation* (2023–2024)
- Niels Savvides: *Heterogeneous data generalization in distributed intrusion detection systems: a federated learning approach* (2023–2024)
- Ilkay Yüksel: *Adversarial robustness of federated learning: defenses against poisoning attacks* (2023–2024)

Ongoing thesis supervision (2024–2025):

- Maarten De Meyere: *Federated learning on Kubernetes: ML-based distributed intrusion detection*
- Simon Hondekyn: *Reproducible network datasets in containerized environments: a CIC-IDS-2017 case study*
- Maxime Tisseghem: *Enhancing network intrusion detection robustness via dataset augmentation: a CSE-CIC-IDS2018 case study*
- Jozef Jankaj: *Enhancing network intrusion detection robustness via dataset augmentation: a CIC-IDS-2017 case study*
- Seppe Van Rijsselberghe: *Towards robust network intrusion detection using traffic augmentation and embedding techniques*

Bibliography

[1] A. Jurcut, T. Niculcea, P. Ranaweera, and N.-A. Le-Khac, "Security considerations for internet of things: A survey," *SN Computer Science*, vol. 1, pp. 1–19, 2020.

[2] A. Singh and K. Chatterjee, "Cloud security issues and challenges: A survey," *Journal of Network and Computer Applications*, 2017.

[3] T. Alam, "A reliable communication framework and its use in internet of things (iot)," vol. 3, 05 2018.

[4] T. Moore, "The economics of cybersecurity: Principles and policy options," *International Journal of Critical Infrastructure Protection*, vol. 3, no. 3-4, pp. 103–117, 2010.

[5] N. Virvilis and D. Gritzalis, "The big four-what we did wrong in advanced persistent threat detection?" in *2013 international conference on availability, reliability and security.* IEEE, 2013, pp. 248–254.

[6] F. Sierra-Arriaga, R. Branco, and B. Lee, "Security issues and challenges for virtualization technologies," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–37, 2020.

[7] S. Lata and D. Singh, "Intrusion detection system in cloud environment: Literature survey & future research directions," *International Journal of Information Management Data Insights*, vol. 2, no. 2, p. 100134, 2022.

[8] V. Rimmer, A. Nadeem, S. Verwer, D. Preuveneers, and W. Joosen, "Open-World Network Intrusion Detection," in *Security and Artificial Intelligence: A Crossdisciplinary Approach*, L. Batina, T. Bäck, I. Buhan, and S. Picek, Eds. Cham: Springer International Publishing, 2022, pp. 254–283. [Online]. Available: https://doi.org/10.1007/978-3-030-98795-4_11

[9] N. Carlini, "Poisoning the unlabeled dataset of {Semi-Supervised} learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1577–1592.

[10] C. Crowley, "Sans 2024 soc survey: Facing top challenges in security operations," SANS Research Program, Tech. Rep., 2024. [Online]. Available: https://newsletter.radensa.ru/wp-content/uploads/2024/07/SANS-2024-SOC-Survey.pdf

[11] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, 2021.

[12] G. González-Granadillo, S. González-Zarzosa, and R. Diaz, "Security information and event management (siem): Analysis, trends, and usage in critical infrastructures," *Sensors*, vol. 21, no. 14, p. 4759, 2021.

[13] A. R. Muhammad, P. Sukarno, and A. A. Wardana, "Integrated security information and event management (siem) with intrusion detection system (ids) for live analysis based on machine learning," *Procedia Computer Science*, vol. 217, pp. 1406–1415, 2023.

[14] S. Bhatt, P. K. Manadhata, and L. Zomlot, "The operational role of security information and event management systems," *IEEE security & Privacy*, vol. 12, no. 5, pp. 35–41, 2014.

[15] T. Van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna, "Deepcase: Semi-supervised contextual analysis of security events," in *IEEE Symposium on Security and Privacy (SP)*, 2022.

[16] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, and Q. Chen, "A survey of intrusion detection systems leveraging host data," *ACM computing surveys (CSUR)*, vol. 52, no. 6, pp. 1–35, 2019.

[17] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos:: Practical intrusion detection and investigation using whole-system provenance," in *IEEE Symposium on Security and Privacy*, 2024.

[18] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls: Review and future trends," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[19] G. Apruzzese, P. Laskov, and J. Schneider, "Sok: Pragmatic assessment of machine learning for network intrusion detection," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2023.

[20] M. A. Aydın, A. H. Zaim, and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Computers & Electrical Engineering*, vol. 35, no. 3, pp. 517–526, 2009.

[21] D. E. Denning and P. G. Neumann, "Requirements and model for ides: a real-time intrusion detection system," 1985.

[22] M. Masdari and H. Khezri, "A survey and taxonomy of the fuzzy signature-based intrusion detection systems," *Applied Soft Computing*, 2020.

[23] M. Roesch, "Snort – Lightweight Intrusion Detection for Networks," p. 11, 1999.

[24] S. IDS, "Suricata IDS," 2010. [Online]. Available: https://suricata-ids.org/

[25] R. Ahmad, I. Alsmadi, W. Alhamdani, and L. Tawalbeh, "Zero-day attack detection: a systematic literature review," *Artificial Intelligence Review*, vol. 56, no. 10, pp. 10 733–10 811, 2023.

[26] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of {SOC} analysts' perspectives on security alarms," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2783–2800.

[27] S. Canard, A. Diop, N. Kheir, M. Paindavoine, and M. Sabt, "Blindids: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 561–574. [Online]. Available: https://doi.org/10.1145/3052973.3053013

[28] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[29] Z. Yang, X. Liu, T. Li, D. Wu, J. Wang, Y. Zhao, and H. Han, "A systematic literature review of methods and datasets for anomaly-based network intrusion detection," *Computers & Security*, 2022.

[30] Z. IDS, "An Open Source Network Security Monitoring Tool," 2018. [Online]. Available: https://zeek.org/

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

[32] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316, iSSN: 2375-1207.

[33] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When malware is packin'heat; limits of machine learning classifiers based on static analysis features," in *Network and Distributed Systems Security (NDSS) Symposium 2020*.

[34] Y. Yuan, Q. Hao, G. Apruzzese, M. Conti, and G. Wang, ""Are Adversarial Phishing Webpages a Threat in Reality?" Understanding the Users' Perception of Adversarial Webpages," Apr. 2024, arXiv:2404.02832 [cs]. [Online]. Available: http://arxiv.org/abs/2404.02832

[35] R. Mills, A. K. Marnerides, M. Broadbent, and N. Race, "Practical Intrusion Detection of Emerging Threats," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 582–600, Mar. 2022, conference Name: IEEE Transactions on Network and Service Management.

[36] B. Cebere and C. Rossow, "Understanding web fingerprinting with a protocol-centric approach," in *RAID*, 2024.

[37] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, pp. 222–232, 1987.

[38] J. Piet, A. Sharma, V. Paxson, and D. Wagner, "Network detection of interactive $\{$SSH$\}$ impostors using deep learning," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4283–4300.

[39] F. Falcão, T. Zoppi, C. B. V. Silva, A. Santos, B. Fonseca, A. Ceccarelli, and A. Bondavalli, "Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection," in *ACM/SIGAPP symposium on applied computing*, 2019.

[40] S. Zanero and S. M. Savaresi, "Unsupervised learning techniques for an intrusion detection system," in *Proceedings of the 2004 ACM symposium on Applied computing*, 2004, pp. 412–419.

[41] I. Arnaldo and K. Veeramachaneni, "The holy grail of "systems for machine learning" teaming humans and machine learning for detecting cyber threats," *ACM SIGKDD Explorations Newsletter*, 2019.

[42] J. Kim, S. Camtepe, J. Baek, W. Susilo, J. Pieprzyk, and S. Nepal, "P2dpi: practical and privacy-preserving deep packet inspection," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 135–146.

[43] G. Vormayr, J. Fabini, and T. Zseby, "Why are My Flows Different? A Tutorial on Flow Exporters," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2064–2103, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9076310/

[44] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2005, pp. 50–60.

[45] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *USENIX Security*, 2022.

[46] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study," in *2021 IEEE Security and Privacy Workshops (SPW)*, May 2021, pp. 7–12.

[47] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, "Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018," in *IEEE Conference on Communications and Network Security*. IEEE, 2022.

[48] M. Catillo, A. Pecchia, and U. Villano, "Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS?" in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume*, 2023.

[49] R. Flood, G. Engelen, D. Aspinall, and L. Desmet, "Bad design smells in benchmark nids datasets," in *IEEE 9th European Symposium on Security and Privacy (EuroSP)*, 2024.

[50] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Inter-dataset generalization strength of supervised machine learning methods for intrusion detection," *Journal of Information Security and Applications*, vol. 54, p. 102564, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214212619310415

[51] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, 2018.

[52] L. D'hooge, M. Verkerken, B. Volckaert, T. Wauters, and F. De Turck, "Establishing the Contaminating Effect of Metadata Feature Inclusion in Machine-Learned Network Intrusion Detection Models," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, L. Cavallaro, D. Gruss, G. Pellegrino, and G. Giacinto, Eds. Cham: Springer International Publishing, 2022, pp. 23–41.

[53] L. D'hooge, M. Verkerken, T. Wauters, B. Volckaert, and F. De Turck, "Discovering non-metadata contaminant features in intrusion detection datasets," in *2022 19th Annual International Conference on Privacy, Security & Trust (PST)*. IEEE, 2022, pp. 1–11.

[54] L. D'hooge, M. Verkerken, T. Wauters, F. De Turck, and B. Volckaert, "Investigating generalized performance of data-constrained supervised machine learning models on novel, related samples in intrusion detection," *Sensors*, vol. 23, no. 4, p. 1846, 2023.

[55] L. D'hooge, M. Verkerken, T. Wauters, F. De Turck, and B. Volckaert, "Castles built on sand: Observations from classifying academic cybersecurity datasets with minimalist methods," in *8th International Conference on Internet of Things, Big Data and Security (IoTBDS)*. SciTePress, 2023, pp. 61–72.

[56] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, and K. Roundy, ""Real Attackers Don't Compute Gradients": Bridging the Gap Between Adversarial ML Research and Practice," in *SaTML*, 2023.

[57] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *IEEE Symposium on security and privacy (SP)*, 2020.

[58] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006639801080116

[59] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Classification hardness for supervised learners on 20 years of intrusion detection data," *IEEE Access*, vol. 7, pp. 167 455–167 469, 2019.

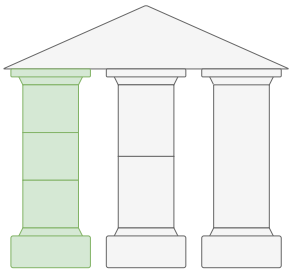
[60] "Fixed version of the cicflowmeter tool," https://github.com/GintsEngelen/CICFlowMeter.

[61] G. Apruzzese, P. Laskov, and J. Schneider, "SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection," Apr. 2023, arXiv:2305.00550 [cs]. [Online]. Available: http://arxiv.org/abs/2305.00550

[62] M. Sarhan, S. Layeghy, and M. Portmann, "Towards a standard feature set for network intrusion detection system datasets," *Mobile networks and applications*, pp. 1–14, 2022.

[63] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[64] M. Verkerken, J. Santos, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Chronosguards: A hierarchical machine learning intrusion detection system for modern clouds," in *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 2024, pp. 1–9.



2

Towards Model Generalization for Intrusion Detection: Unsupervised Machine Learning Techniques

This chapter is situated within the first pillar of this dissertation, AI for Cybersecurity, and addresses the first two research questions: whether unsupervised ML models can achieve classification performance comparable to supervised approaches on academic benchmark datasets (RQ1), and whether these models generalize across heterogeneous network environments (RQ2).

This chapter builds on our earlier work [1] by extending the evaluation of four widely used unsupervised machine learning techniques on CIC-IDS-2017 to include a second modern flow-based NIDS dataset, CSE-CIC-IDS-2018 (RQ1). To move beyond the until now default intra-dataset validation, these compatible datasets provide a unique opportunity to evaluate the generalization capacity of ML models across heterogeneous network environments (RQ2). This chapter proposes an inter-dataset evaluation strategy, where models trained and validated on one dataset are tested on a second, previously unseen dataset. Although all models demonstrate strong performance in intra-dataset evaluations, their classification accuracy drops significantly—by up to 37% AUROC—under the inter-dataset strategy. These results highlight the generalization challenge in current ML-based NIDS and serve as an important step toward more realistic evaluation and development of more robust, generalizable ML-NIDS.

⋆ ⋆ ⋆

M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck

Published in the Springer Journal of Network and Systems Management (JNSM), October 2021

Abstract Through the ongoing digitization of the world, the number of connected devices is continuously growing without any foreseen decline in the near future. In particular, these devices increasingly include critical systems such as power grids and medical institutions, possibly causing tremendous consequences in the case of a successful cybersecurity attack. A network intrusion detection system (NIDS) is one of the main components to detect ongoing attacks by differentiating normal from malicious traffic. Anomaly-based NIDS, more specifically unsupervised methods previously proved promising for their ability to detect known as well as zero-day attacks without the need for a labeled dataset. Despite decades of development by researchers, anomaly-based NIDS are only rarely employed in real-world applications, most possibly due to the lack of generalization power of the proposed models. This chapter first evaluates four unsupervised machine learning methods on two recent datasets and then defines their generalization strength using a novel inter-dataset evaluation strategy estimating their adaptability. Results show that all models can present high classification scores on an individual dataset but fail to directly transfer those to a second unseen but related dataset. Specifically, the accuracy dropped on average 25.63% in an inter-dataset setting compared to the conventional evaluation approach. This generalization challenge can be observed and tackled in future research with the help of the proposed evaluation strategy in this chapter.

2.1 Introduction

The rapid increase of connected devices to the internet and the accompanying amount of data sent over these interconnected networks does not only greatly expand the attack surface for people with malicious intents, but it also enlarges the potential impact in case of a successful cybersecurity breach. Over the last years, this resulted in a continuously growing risk that will most likely persist in the future. Even the global covid-19 pandemic could not stop this trend. A recent survey conducted by Gartner regarding future investments in the internet of things (IoT) proved that the majority of companies will increase their investments with the goal of reducing costs in the future [2]. This results in a further acceleration in the number of deployed devices and consequently enlarging the attack vector surface by a multitude. One component that can be used to mitigate the risk of a successful cyberattack is an intrusion detection system (IDS). The goal of an IDS is to detect any malicious activity on a network or system. Differentiation can be made based on the input source the IDS uses to detect an attack. A host-based intrusion detection system (HIDS) will be deployed on a particular system where it monitors system logs and resource usages, while a network-based intrusion detection system (NIDS) is placed on a node in the network and monitors traffic that passes through the network. While both types can be implemented in hardware or

software and have the ability to detect an attack before, during, or after it has occurred, an NIDS has the advantage to simultaneously cover an entire (sub)network rather than a single system for HIDS.

Traditionally, a misuse-based NIDS is used, which usually computes a signature over each incoming and outgoing packet, which in turn is compared against a database with signatures of known attacks. When a matching signature is found, then the corresponding packet is flagged as malicious. This approach can detect known attacks with high confidence but comes with a few drawbacks. First, this approach is limited by design to detect known attacks. Furthermore, there exist multiple evasion techniques to adapt the signature of a packet in such a way it becomes infeasible to match [3]. Similarly, the recent wide adoption of encryption in network traffic, in particular of HTTPS, makes it impossible to inspect the encrypted body of the packet [4]. Recently, the inspection of encrypted traffic became an active field within the research community with promising results supported by advances in the field of deep learning [5][6][7]. At last, it is a labor-intensive task to continuously maintain and update a database with known attacks [8].

In the last decade, research shifted its focus towards anomaly-based intrusion detection empowered by the developments within the field of machine learning. Machine learning models can take advantage of big amounts of data to learn certain patterns and apply these to future inputs. Two main learning techniques exist: supervised and unsupervised. Contrary to supervised, unsupervised techniques do not require labeled data for training, eliminating the need for an expensive human expert, as they mostly try to model benign behavior, and everything that differs too much from this model gets flagged as malicious. Often these techniques are not applied to the raw network packets, but instead, the traffic is aggregated into bidirectional flows of packets between the same source and destination. Over these bidirectional flows, multiple statistical features are computed in both forward as backward directions. For example, the number of packets, duration, and the average length of a packet. There are multiple advantages of transforming the raw network traffic into flows. For example, it enables a more high-level overview of the current situation as it is not limited to a single packet, besides, it also greatly reduces the amount of data to transmit in the case of a centralized fraud detector, limiting the bandwidth requirements [9].

Recent literature [10][11] advocates the adoption of flow-based NIDS relying on unsupervised machine learning techniques in the real world because of their potential to detect zero-day attacks, as long as the attack substantially differs from normal behavior, in combination with the efficient usage of high volumes of data. Despite the extensive research efforts and promising reported results, no or limited adoption in real-world applications is found until now. This can have multiple possible causes. One of them is a generalization problem, indicating that a trained model excels in an experimental setup but does not deliver the same observed classification power on new, unseen samples. An alternative approach to evaluate the generalization power of a model is proposed in this chapter that goes one step further than the commonly accepted strategy by using two different but related datasets instead of only a single dataset. Using this inter-dataset strategy, the first dataset is employed for training and validation, and the second for the final unbiased evaluation. This way, a model overfitting on a specific dataset will yield inferior results on

the second dataset and urge researchers to develop well generalizing models that most probably perform better in a real-world environment.

The main contributions of this chapter are three-fold. First, the evaluation and analysis of four promising unsupervised models (autoencoder, one-class SVM, isolation forest, and principal components analysis) on two recent and realistic datasets (CIC-IDS-2017 and CSE-CIC-IDS-2018). Second, a novel inter-dataset evaluation strategy is introduced to evaluate the generalization strength of newly proposed models. Third, this novel strategy is employed to estimate the generalization strength of the four unsupervised models without prior adaptation and compared with the conventional intra-dataset baseline.

The remainder of this chapter is structured as follows. Section 2.2 starts with giving an overview of related work in the field of NIDS and available datasets to use for the evaluation of the proposed models. Next, the methodology is presented to enable reproducible results together with the inter-dataset evaluation strategy in section 2.3. Section 2.4 presents the experimental results which are then extensively discussed in section 2.5. Before drawing a conclusion in section 2.7, possible future work is listed in section 2.6.

2.2 Related Work

This section starts by describing the recent literature conducted within the field of anomaly-based NIDS, highlighting the many techniques and algorithms together with their sometimes remarkable classification scores. Next, these results reported by researchers in an experimental setup are questioned for their real-world relevancy from both the broad perspective of artificial intelligence (AI) to the specific field of network intrusion detection. This section is concluded by a brief overview of the available datasets suitable for the evaluation of NIDS.

The first IDS has already been proposed in 1980 [12], but since the rapid advances in the field of machine and deep learning of the last decade, research interest has surged with researchers applying these techniques to develop novel IDS. Multiple surveys give a good review of the developments in the IDS field [13][14][15] [16]. They advocate the use of unsupervised techniques to overcome the limitations of available realistic datasets and for their ability to detect unseen, zero-day attacks. More specifically, the following studies confirmed the use of autoencoders [17] [18], isolation forest [19], one-class SVM [20] and principal components analysis [21] for anomaly detection with promising results but often request real-world validation in their future work. Otoum et al. [22] recently proposed a hybrid IDS that takes advantage of a traditional signature-based IDS for known attack detection combined with a anomaly-based detection for unknown attacks. Most of the proposed machine learning IDS use a flow-based classification approach. Because these flows are only constructed from the packet headers, they are not sensitive to encryption protocols preventing inspection or classification [23]. Khatouni et al. [24] uses four open-source traffic flow analyzers to extract and/or construct traffic flow level features from packet traces. Using these flow features they successfully identified known services from multiple encrypted service

channels.

All these studies have in common that they achieve excellent classification performance and promote their models for real-world usage, yet limited adoption of these techniques is found in operational applications. Already 10 years ago, this problem was raised by the research community [25] but with little effect. Recently, a paper by a collaborative effort from Google recognized this as a global challenge and named it *underspecification*. Additionally, more specifically within the field of network intrusion detection, the remarkable high results published in the literature have been openly questioned by Leevy et al. [26]. Furthermore, the survey stresses the importance of documenting all the taken steps for reproduction purposes. The survey by Ahmad et al. [27] also questions the low performance of machine learning IDS in the real-world and requests an effective method to validate the generalization performance in their future work. A recent study by Al-Omare et al. [28] tries to limit the risk of overfitting by ranking the used features and only selecting the top-performing ones with as additional benefit of reducing the computational complexity because of the lower input dimension. Similarly, Aloqaily et al. [29] proposed a hybrid intrusion detection system, named D2H-IDS, which uses a deep belief network for dimensionality reduction and a decision tree for the attack classification. The work conducted in this study tries to close the gap in classification performance between academic prototypes and operational applications by proposing an alternative evaluation strategy to estimate the generalization strength of suggested models in the recent literature and initiate future research on this topic.

The training and evaluation of machine learning-based IDS require a lot of data. Preferably, this data should be sampled from the real-world and representative for future inputs. This often proves to be a challenge as network traffic contains sensitive data and has obvious privacy concerns. One way to overcome this barrier is by anonymizing the data to prevent any information from being traced back to an individual person. This often includes operations like data aggregation or removing and modifying certain features. History has proven that this is a tedious task. The Netflix Prize is a popular example of where it went wrong. The world's largest streaming service publicly released a dataset in 2006 containing movie ratings of 500,000 subscribers. Only after a few weeks, the anonymization algorithm was already broken and sensitive information of individual users was exposed [30]. Another way to obtain realistic network traffic is by generating a synthetic dataset. Instead of collecting data from a network and its actual users with their related challenges, an experiment can be set up using a wide range of techniques to imitate them. This approach only works if the used techniques approximate benign behavior close enough.

Limited realistic datasets exist for intrusion detection. The most widely used dataset is KDDcup99 [31], created by the Defense Advanced Research Projects Agency (DARPA) for the fifth Conference on Knowledge Discovery and Data Mining in 1999. Ten years after its release, Tavallaee et al. [32] published an updated version, NSL-KDD, together with a comprehensive analysis specifying the various issues in the original dataset. The revised dataset solved many of the demonstrated shortcomings [33], but now, more than 20 years later, the used network protocols and attacks are no longer representable for modern communication networks. In the last decade, the Canadian Institute for Cybersecurity (CIC) became the front runner regarding the generation of realistic net-

work intrusion detection datasets. Sharafaldin et al [34] proposed 11 criteria needed to create a reliable intrusion detection dataset for benchmarking. The first dataset satisfying all 11 criteria is CIC-IDS-2017 and collects real network traffic using multiple internet protocols such as HTTP(s), FTP, SSH, IMAP, and POP3 for a duration of 5 days [35]. Unique about this dataset is that all benign traffic is generated by a B-Profile system imitating human interaction. One year later, CSE-CIC-IDS-2018 was published by the CIC in collaboration with the Communications Security Establishment (CSE). This dataset took the same principles of CIC-IDS-2017 but was executed on a larger scale network in the cloud of Amazon Web Services (AWS). Both datasets are distributed as raw network packets (PCAP) or bidirectional flows aggregated from the PCAP files by CICFlowMeter [36][37]. The latter enables easy use in IDS employing machine learning techniques.

2.3 Methodology

This section is divided into subsections regarding the multiple aspects to reproduce the reported results in section 2.4. First, subsection 2.3.1 discusses the used datasets together with the necessary steps from data cleaning to feature reduction required to prepare the raw data. Next, the evaluated algorithms and their optimized hyper-parameters are described in subsection 2.3.2. Subsection 2.3.3 describes the evaluation strategy, including the alternative inter-dataset evaluation strategy proposed in this chapter to estimate the generalization strength of a model. The used metrics to report the performance of the models are documented in subsection 2.3.4. Finally, the used hardware for the execution of the experiments is reported in subsection 2.3.5.

2.3.1 Datasets

For the novel inter-dataset evaluation strategy, a minimum of two datasets are needed. It is important that these datasets contain identical features and are related, more specifically for unsupervised binary classification, the benign traffic (X, y=0) should be sampled from the same distribution (D), see equation 2.1. This assumption allows to evaluate a model trained on a first dataset, on a second related dataset as the learned normal behavior theoretically should still be applicable and transferable. Ideally, a well-generalized model would classify the benign samples of both datasets with a similar classification performance. Key is that all data goes through the same preprocessing pipeline, preserving the identical set of features.

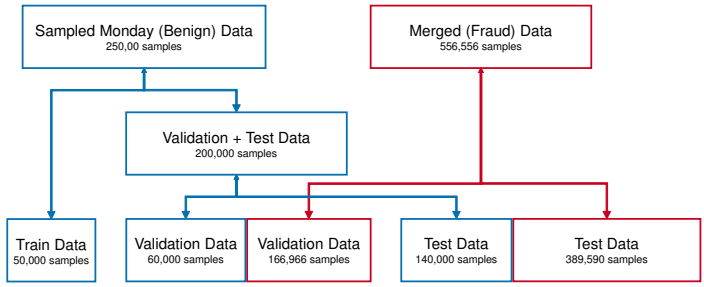
$$\begin{aligned} X, y &\sim D_1 \\ X, y &\sim D_2 \\ D_1 = D_2 \quad \forall X \text{ if } y &= 0 \end{aligned} \tag{2.1}$$

The Canadian Institute for Cybersecurity (CIC) developed a system called B-Profile to produce benign background traffic. These profiles derived abstract behavior from a group of real users using machine learning and statistical analysis techniques. In 2017, the CIC published the first dataset,

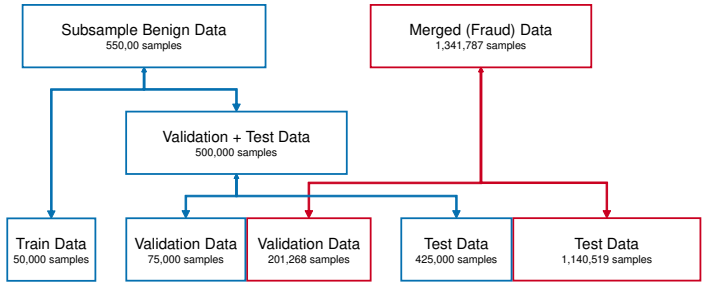
Table 2.1: Attack classes and their number of occurrences in CIC-IDS-2017 and CSE-CIC-IDS-2018

Attack Class	Details	CIC-IDS-2017		CSE-CIC-IDS-2018	
		Count	Pct (%)	Count	Pct (%)
Benign	-	2,271,320	80.32	7,364,941	84.59
(D)DOS	Hulk	230,124	13.43	145,199	11.17
	DDOS	128,025		775,955	
	GoldenEye	10,293		41,406	
	DoS slowloris	5,796		9,908	
	Slowhttptest	5,499		43	
Port scan	-	158,804	5.62	-	-
Brute Force	FTP-Patator	7,935	0.49	53	1.08
	SSH-Patator	5,897		94,041	
Web-Attack	Brute Force	1,507	0.08	555	0.01
	XSS	652		227	
	SQL Injection	21		79	
Botnet	-	1,956	0.07	144,535	1.66
Infiltration	-	36	<0.01	129,786	1.49
Heartbleed	-	11	<0.01	-	-

CIC-IDS-2017, using this technique. One year later, they published a second dataset in collaboration with the Communication Security Establishment (CSE), CSE-CIC-IDS-2018, which used the same B-Profiles to generate benign traffic. The second experiment took the generation process to a larger scale by bringing the whole network setup to the Amazon Web Services (AWS) cloud computing platform and using a total of 500 machines instead of only 14 in the first iteration. Because both datasets are generated using the same tools and processes while only differing the deployment environment, they satisfy all the requirements to be used in the novel intra-dataset evaluation strategy. The datasets are distributed as raw network packets (PCAP) as well as machine learning-friendly CSV files containing bidirectional flows (biflows) extracted from the PCAPs by CICFlowMeter [37]. These biflows contain next to a few flow identification features such as source and destination IP-address, also a timestamp, 80 statistical network features, and a label. This label contains "Benign" for the background traffic or the name of the attack class for malicious traffic and is only used during hyper-parameter selection and the final evaluation of the classification performance. Table 2.1 gives an overview of the number of occurrences for each attack class in both datasets. Important to highlight is the substantial class imbalance between the benign and multiple attack classes. Furthermore, not all classes are as present in the 2018 dataset as in the 2017 one or vice versa. Besides, even two classes, *heartbleed* and *port scan*, entirely disappeared and are not explicitly present anymore. Nevertheless, the benign class stays the most present class in both datasets with 80.32 and 84.59 percent, respectively. The class imbalance does not affect the training of the models because this chapter employs a semi-supervised



(a) CIC-IDS-2017



(b) CSE-CIC-IDS-2018

Figure 2.1: Train, validation and test split strategy for the datasets.

learning approach with only benign traffic, but can substantially influence the validation and test metrics if not carefully defined.

Figure 2.1 visualizes the applied train, validation, and test split strategy for both datasets. First, benign and malicious traffic are independently collected. Important is that these sets are clear of rows with missing values and do not contain any duplicates. Next, the benign dataset is subsampled without replacement to 250,000 flows for CIC-IDS-2017 and 550,000 flows for CSE-CIC-IDS-2018, from which 50,000 are used as the training set. The remaining benign together with all malicious flows are stratified sampled over validation and test set with a 30/70 for the 2017 and a 15/85 ratio, to retain a similarly sized validation set, for the 2018 dataset. As a result, a train, validation, and test set are obtained that are practical to use and for which the i.i.d. requirement is fulfilled.

2.3.1.1 Preprocessing Pipeline

The preprocessing pipeline transforms the raw data into the right format to be consumed by the machine learning model. Figure 2.2 visualizes the five steps of the pipeline. The first three steps clean the data by removing columns with redundant information, dropping rows with missing or infinity values, and eventually, the resulting dataset is filtered from duplicates. More specifically,

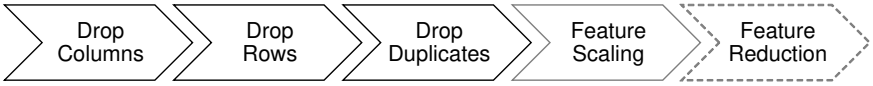


Figure 2.2: The preprocessing pipeline consisting of five steps of which the last one is not employed for the algorithms relying on a reconstruction error as anomaly score.

step two removed a duplicate column found in the 2017 dataset, removed ten features without any variance, and six features that could lead to overfitting due to the dataset generation setup such as IP addresses and timestamps. Only very limited rows contained missing or infinity values, therefor these rows are being dropped rather than filled with techniques such as imputation. This way the information loss is minimized and guarantees original and high-quality in the next phases. After the cleaning phase, the resulting collection of flows contains 67 features, see appendix 2.A for an overview. In the following feature scaling step, the input features are normalized. Because many methods exist for the normalization of the range of the data, the used method is added as a hyper-parameter to the optimization problem. Following, four techniques with their implementations from the scikit learn library [38] are included: *StandardScaler, RobustScaler, QuantileTransformer, and MinMaxScaler*. At last, the final feature reduction step is only applied for the *isolation forest* and *one-class svm* since both *principal components analysis* and *autoencoder* algorithms rely on a reconstruction error as anomaly score and already apply a feature reduction technique internally. Because this chapter evaluates unsupervised techniques, a PCA transformation is employed to reduce the high-dimensional to a lower-dimensional feature space without the need for any labels. The number of principal components of the resulting transformation is also added as a hyper-parameter to the optimization problem. To summarise, the first three steps cleaning the data can be applied globally while the last two with their corresponding hyper-parameters are included in the optimization phase of a particular algorithm.

2.3.2 Algorithms

In this section, the used anomaly detection algorithms are presented, together with the used anomaly score, and the hyper-parameters that are optimized. For all anomaly scores discussed in this section applies, the higher the score, the higher the probability for a sample to be malicious. For each algorithm, a work in the literature is referred to as a starting point for a more thorough explanation of their internal workings.

2.3.2.1 Principal Component Analysis

Principal components are the sequence of vectors that are linearly uncorrelated after an orthogonal transformation of an original, possibly correlated feature set. The process of calculating this transformation and executing it is called *Principal Component Analysis (PCA)* [39] [40]. This technique is often used for feature reduction by only retaining the components that explain the most variance of the original data. In this chapter, PCA is also employed for anomaly detection

by first performing a PCA transformation followed by the inverse transformation, to reconstruct the original data. As an anomaly score, the reconstruction error is used by computing the sum of squared errors (SSE) between the input and output vector. Without removing the components that explain the least variance, this would be a loss-less operation with an SSE equal to zero. However, when the PCA transformation is fitted on only benign data in combination with a reduction in principal components, then the malicious samples will yield a higher reconstruction error and thus anomaly score as long as the assumption that malicious traffic differentiates from normal behavior is satisfied. The number of principal components is the only hyper-parameter that needs to be optimized. For the implementation, the scikit learn library is used [38].

2.3.2.2 Isolation Forest

An *isolation forest* attempts to isolate anomalies in high-dimensional data instead of trying to model normal behavior and does this in linear time with low memory demands [41]. Therefore, it builds an ensemble of binary trees with each tree constructed from a random subsample of the training data. While the algorithm tries to isolate anomalies, it does not require them to be present during the training phase. For each sample, the average depth over all trees can be computed, which is equivalent to the number of splits that are needed to isolate that sample. As anomalies are assumed to be different from the benign data on which the trees are built, they will reside higher in the tree and thus have a lower average depth. As anomaly score, this average depth is negated so that a higher score results in a higher probability to be malicious. The number of trees, subsample rate, and the number of features to construct an individual tree are the hyper-parameters tuned during optimization. The scikit learn implementation is used [38].

2.3.2.3 Autoencoder

An *autoencoder* (AE) is an artificial neural network coming from the field of deep learning, consisting of an encoder and decoder. The encoder transforms an input vector to a lower-dimensional or latent space, after which the decoder will reconstruct the original input vector as close as possible [42]. An AE with a single hidden layer and a linear activation function results in a linear transformation which is equivalent to PCA with the number of principal components equal to the number of nodes in the single hidden layer [43]. On the other hand, when multiple hidden layers or a non-linear activation function are used, the transformation becomes non-linear. As the anomaly score, the reconstruction error is used between the input and output vector computed as SSE. The number of hidden layers, number of neurons per layer, activation function, and regularisation terms are the optimized hyperparameters. For the implementation the easy-of-use yet powerful *Keras* [44] framework is used which itself is built on top of *Tensorflow* [45].

2.3.2.4 One-Class SVM

One-class SVM works similar to standard support vector machines, but instead of separating two classes with a hyper-plane, it encloses a single class with a hyper-sphere. As the standard SVM

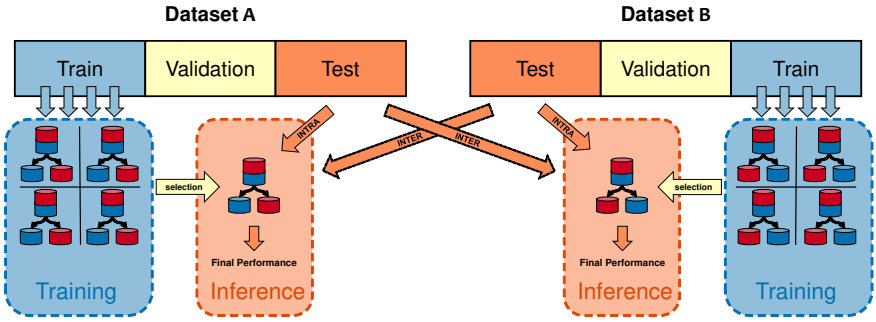


Figure 2.3: Graphical representation of the common intra- and novel inter-dataset evaluation strategy proposed in this chapter to estimate the generalization strength of machine learning models.

normally tries to find this hyper-plane with the largest possible separation margin, this translates to the smallest possible hyper-sphere for the unsupervised variant [46]. Some instances may be violating this separation to account for noise in the data and by using a kernel function a complex, non-linear boundary can be created for the sphere. The following kernel functions are examined: linear, polynomial, radial basis function (rbf), and sigmoid. The hyper-parameters to optimize are the kernel function, the kernel coëfficient and a regularization parameter ν. The parameter ν controls the number of margin errors by putting an upper bound on it. For the implementation, the scikit learn library is used [38].

2.3.3 Evaluation Setup

This chapter evaluates the generalization strength of all proposed models on basis of two different strategies. First, the commonly accepted approach to prevent overfitting and train models with high generalization power splits a single dataset in a train, validation, and test set in such a way that the independent and identically distributed (i.i.d.) assumption is satisfied. Afterwards, the internal- and hyper-parameters of the model are obtained by, respectively, using the training and validation split, often combined in a cross-validation manner. Finally, the unseen test data is used to provide an unbiased evaluation of the final model's performance. This generally results in a low risk of overfitting and a model with a good generalization strength but is still strongly dependant on the quality of the used dataset [47]. For the remainder of this chapter, this strategy is called *intra-dataset* and is visualised in figure 2.3.

Secondly, this chapter proposes an alternative evaluation strategy employing two different but related datasets. Each dataset still gets split into three parts (train, validation, and test) according to the same rules as mentioned before. Afterwards, multiple instances of the same model with different hyperparameters are trained from the train set. The validation set is used to select the best performing model with its corresponding hyperparameters of these instances. At this point, the exact same approach is used as in the intra-dataset strategy, but instead of defining the final performance of the model on the test set of the same dataset, a test set of a second

related dataset is used. With two datasets, the inter-dataset strategy can be executed twice, once train/validate on dataset A, testing on dataset B, and once the inverse. This strategy is especially valuable when using synthetically generated datasets, as often the case for problems containing sensitive information such as IDS. Models trained on a dataset containing (hidden) features specific to the generation process and highly correlated with the label will result in excellent results on that particular dataset but will fail to generalize those to a second dataset. This strategy brings the estimation of the generalization strength of a model one step closer to real-world evaluation. For the remainder of this chapter, this strategy is called *inter-dataset* and is accordingly visualized in figure 2.3.

For both strategies, it is important that the hyperparameters are fully optimized to select the best model. The open-source optimization framework *Optuna* [48] is used for the implementation. The tree-structured parzen estimator (TPE) is used to search efficiently through large spaces of possible values by selecting the next combination of hyperparameters based on the highest expected improvement. All the code can be retrieved from the GitLab repository [49].

2.3.4 Metrics

The models in this chapter are evaluated in terms of classification performance and computational complexity. The latter is simply documented as the time needed to train the model and to evaluate the test set in seconds. To discuss the used metrics for the evaluation of the classification performance, first, four terms need to be introduced: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). The positive class resembles the fraudulent flows while the negative class represents benign flows, more specifically TP are the malicious flows flagged as such, FP are benign flows flagged as fraud, TN are benign flows classified as normal and FN are malicious flows that stay undetected. A good anomaly detector maximizes the TP while having as few FN as possible. Many metrics are derived from these four situations.

2.3.4.1 Accuracy

The accuracy is defined as the proportion of correctly classified samples out of the total number of predicted samples. In the use-case of IDS this translates to the sum of the correctly detected malicious and benign flows, divided by the total number of classified flows, see equation 2.2.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

2.3.4.2 Recall

The recall or true positive rate (tpr) is defined as the fraction of positive samples that are successfully detected. Equation 2.3 defines how to compute the recall in function of TP, FP and FN. Intuitively, the recall can be interpreted as the fraction of attacks the IDS effectively detects.

$$recall = \frac{TP}{TP + FN} \tag{2.3}$$

2.3.4.3 Precision

Precision is defined as the fraction of the actual positive samples out of all the predicted positive samples. Equation 2.4 defines how to compute the precision in function of TP and FP. In the case of IDS, the precision can be interpreted as the fraction of actual attacks among all the raised alarms.

$$precision = \frac{TP}{TP + FP} \tag{2.4}$$

2.3.4.4 F1 Score

The F1 score combines both the recall and precision into a single measurement and is computed by taking the harmonic mean of the two, see equation 2.5. This single value enables easy comparison between different evaluated techniques and related work. The reported F1 score in this chapter is defined by the threshold on the anomaly score that yields the maximum score. This threshold is required to classify a sample either as benign or fraud if its anomaly score is, respectively, lower or higher.

$$F_1\ score = \frac{2 * recall * precision}{recall + precision} \tag{2.5}$$

2.3.4.5 Area Under Receiver Operating Characteristic Curve

The receiver operating characteristic curve plots the recall (or tpr) in function of the false positive rate (fpr) and visualizes the discrimination ability of a binary classifier with a varying threshold on the anomaly score. The area under the receiver operating characteristic curve (AUROC) summarizes this to a single metric. The maximum score of 1 is equivalent to a perfect classifier while a completely random classifier would result in a score of 0.5. The AUROC can intuitively be interpreted as the probability that a higher anomaly score will correctly be assigned to a randomly selected positive sample than to a randomly selected negative sample. This metric is suggested to be used for imbalanced problems such as fraud detection [50] because both the tpr and fpr used to construct the curve are fractions and thus independent of possible class imbalances in the dataset.

2.3.4.6 Area Under Precision-Recall Curve

The precision-recall curve plots the precision in function of the recall and shows the trade-off between them for different thresholds on the anomaly score. Similar to the AUROC, the area under the precision-recall curve (AUPR) summarizes the area under the curve to a single value. A high

score is achieved when the model classifies most of the frauds (i.e. high recall) while making few false alarms (i.e. high precision).

2.3.5 Hardware Setup

Each experiment for a combination of an algorithm and dataset is submitted to a job-based distributed platform. Each job starts in an isolated container built upon a basic Python 3.7 Docker image with all the needed libraries preinstalled and receives dedicated resources for maximum performance and objective benchmarking. Each job was assigned 4 CPUs, Intel(R) Xeon(R) Silver 4108 CPU @ 1.80GHz, for parallel hyperparameter optimization and 16 GB of RAM. All jobs running an AE on the Tensorflow platform were also assigned a GPU, GeForce GTX 1080 Ti, with 11 GB dedicated RAM.

2.4 Results

This section presents the results of both the intra- and inter-dataset evaluation strategy on CIC-IDS-2017 and CSE-CIC-IDS-2018 for all four unsupervised algorithms in order of increasing the average classification strength on the individual datasets. The AUROC score is used as the main evaluation metric during analysis. For completeness and in order to allow easy comparison with related work in the literature, the area under precision-recall (AUPR), accuracy, F1 score, and its corresponding precision and recall are also reported. How these metrics are computed is documented in section 2.3.4. An overview of the classification scores is given in table 2.2. The dataset column notes the training dataset followed by the test dataset. In the case of the inter-dataset evaluation, these datasets will differ while they will be equal for the intra-dataset evaluation strategy. Next to the classification performance also the computational complexity for both training and inference is highlighted, see table 2.3 for an overview.

2.4.1 Principal Component Analysis

The PCA model only had a single hyper-parameter to train, its number of principal components, together with selecting the most suited normalization technique to rescale the features. The model trained on the 2017 dataset employing a quantile scaler in conjunction with 32 components resulted in the highest AUROC of 0.9373. A maximum F1 score of 0.9390 with a corresponding recall and precision of 0.9435 and 0.9346, respectively, is achieved with a threshold of 0.700 on the SSE. On the 2018 dataset, the robust scaler in combination with 51 principal components proved best with an AUROC of 0.8494. A cutoff on the SSE of 0.006 resulted in the maximum F1 score of 0.9102 with a corresponding recall of 0.9481 and precision of 0.8752. These best performing models on each dataset are subsequently used to evaluate the other dataset without any prior adaptation in inter-dataset evaluation strategy. The model trained on 2017 and evaluated on 2018 was able to still achieve an AUROC of 0.6661, the highest absolute AUROC of all four analyzed algorithms. On the other hand, the model trained on 2018 and evaluated on 2017 achieved an AUROC of 0.6343.

Table 2.2: Overview of the binary classification performance on both individual and inter-dataset evaluation.

Algorithm	Dataset	Recall	Precision	F1	Accuracy	AUPR	AUROC ± std
PCA	2017 - 2017	0.9435	0.9346	0.9390	0.9098	0.9677	0.9373 ± 0.0004
	2018 - 2018	0.9481	0.8752	0.9102	0.8637	0.9041	0.8494 ± 0.0004
	2017 - 2018	0.7780	0.7762	0.7771	0.6748	0.8021	**0.6661 ± 0.0005**
	2018 - 2017	0.9951	0.7513	0.8562	0.7541	0.8670	0.6343 ± 0.0008
Isolation Forest	2017 - 2017	0.9314	0.9470	0.9391	0.9111	0.9831	0.9584 ± 0.0003
	2018 - 2018	0.9107	0.9223	0.9165	0.8790	0.9477	0.9055 ± 0.0003
	2017 - 2018	0.3562	0.7573	0.4845	0.4479	0.7688	0.6429 ± 0.0006
	2018 - 2017	0.8218	0.7204	0.7678	0.6343	0.8471	0.5883 ± 0.0008
Autoencoder	**2017 - 2017**	0.9778	0.9459	0.9616	0.9426	0.9911	**0.9775 ± 0.0002**
	2018 - 2018	0.9164	0.9144	0.9154	0.8766	0.9638	0.9200 ± 0.0002
	2017 - 2018	0.9025	0.7499	0.8191	0.7097	0.7580	0.6434 ± 0.0006
	2018 - 2017	0.8307	0.7184	0.7705	0.6360	0.8476	0.5751 ± 0.0008
One-Class SVM	2017 - 2017	0.9920	0.9104	0.9495	0.9223	0.9890	0.9705 ± 0.0002
	2018 - 2018	0.9323	0.9268	0.9296	0.8898	0.9741	**0.9420 ± 0.0004**
	2017 - 2018	0.9305	0.7748	0.8455	0.7523	0.8010	0.6412 ± 0.0005
	2018 - 2017	0.9993	0.7363	0.8479	0.7363	0.9245	**0.7739 ± 0.0007**

Table 2.3: Overview of the computational complexity of training and inference for CIC-IDS-2017 and CSE-CIC-IDS-2018

Algorithm	Dataset	Training ± std (s)	Inference ± std (s)
PCA	2017	0.62 ± 0.02	7.84 ± 0.02
	2018	1.16 ± 0.18	13.83 ± 2.02
Isolation Forest	2017	2.59 ± 0.011	25.3 ± 0.058
	2018	3.62 ± 0.16	79 ± 3
Autoencoder	2017	77.0 ± 29.1	16.11 ± 0.155
	2018	103 ± 31	18.62 ± 0.07
One-Class SVM	2017	86 ± 0.086	32 ± 0.078
	2018	73.0 ± 0.1	1192 ± 104

This results in an average decrease of 27.13% in AUROC while the accuracy drops 19.26% between the intra- en inter-dataset evaluation.

2.4.2 Isolation Forest

During optimization, the number of principal components in the last step of the preprocessing pipeline, the number of trees with their sample rate, and the fraction of the features used to construct a single tree are defined. On the 2017 dataset rescaled by the quantile scaler with 15 components, an isolation forest consisting of 58 trees with a sample rate of 0.9612% but only consisting of 6 features per tree, yielded the highest AUROC of 0.9584. With a threshold of -0.0305 on the anomaly score, a maximum F1 score of 0.9391 is obtained, compounded by a recall and precision of, respectively, 0.9314 and 0.9470. The model trained on the 2018 dataset rescaled with the min-max scaler followed by a PCA transformation with 5 principal components yielded an AUROC of 0.9055 for an isolation forest with 62 trees, a sample rate of 86.47% and only 3 features per tree. The highest F1 score of 0.9165 is obtained with a threshold of -0.2837. The corresponding recall and precision are 0.9107 and 0.9223. Without any prior adaptation, the model trained on the 2017 dataset still achieves an AUROC of 0.6429, while the model trained on 2018 yields 0.5883 as AUROC. This is an average decrease of 33.98% of AUROC and 39.34% drop in accuracy between intra- and inter-dataset evaluation.

2.4.3 Autoencoder

Aside from the architecture of the autoencoder also the hidden activation function, l2 regularisation term, and scaler are defined during the optimization phase. An autoencoder constructed with relu as hidden activation function, an l2 regularisation term of 4.945e-5, and an architecture with 5 hidden layers of 56, 54, 51, 54, and 56 nodes, combined with the quantile scaler yielded the maximum AUROC of 0.9775, the highest score of all four algorithms. An F1 score of 0.9616 with a corresponding recall of 0.9778 and precision of 0.9459 is obtained with a threshold of 11.58 for the reconstruction error. The min-max scaler together with an artificial neural network with a relu hidden activation function, 1.66e-4 as l2 regularisation term, 9 hidden layers, and a consecutive number of neurons per layer of 57, 51, 42, 40, 13, 40, 42, 51 and 57 yielded the maximum AUROC of 0.9200 on the 2018 dataset. The maximum F1 score is 0.9154, composed of a recall of 0.9164 and a precision of 0.9144 when using 0.0094 as the threshold for the reconstruction error. In the inter-dataset setting, the model trained on the 2017 dataset decreased with 34.18% to an AUROC of 0.6434. Similarly, the model trained on the 2018 dataset decreased with 37.49% to an AUROC of 0.5751 when evaluated on the 2017 dataset. As a result, the AUROC score decreased on average by 35.84% while the accuracy only dropped 26.08%.

2.4.4 One-Class SVM

A one-class SVM model has next to its hyper-parameters the used kernel function with its coëfficient and regularization parameter ν, also the used scaler to normalize the features and the

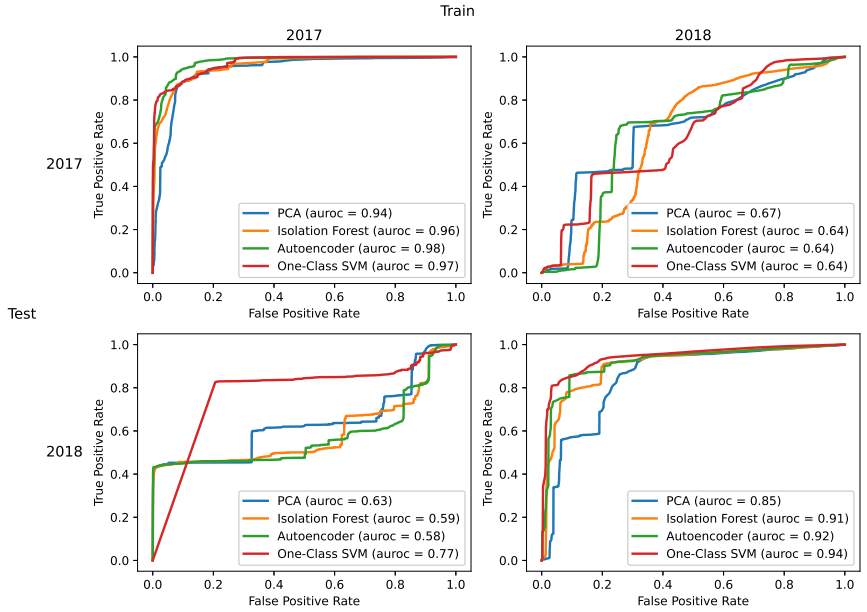
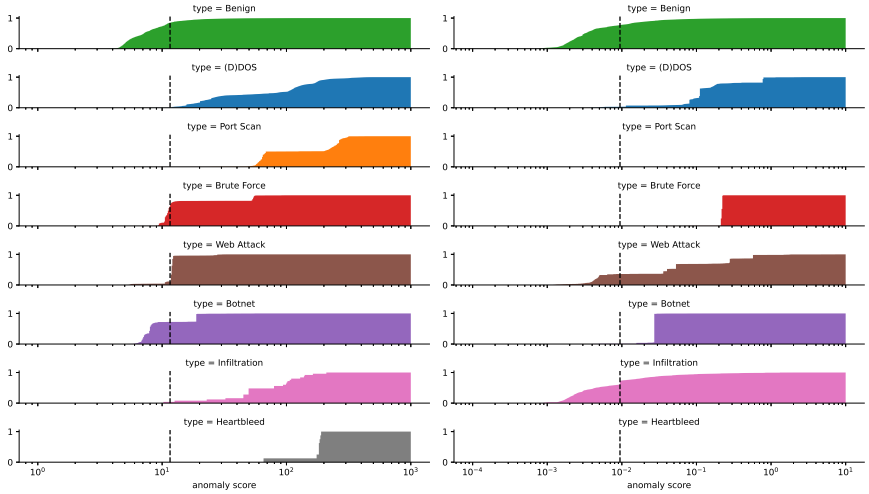


Figure 2.4: The receiver operating characteristics (ROC) curves plotted for evaluated algorithms in a grid with on the x-axis and y-axis the used dataset respectively for training and testing.

number of components for feature reduction to optimize. The best parameters found on the 2017 dataset are a rbf kernel with 0.1318 as coëfficient, ν equal to 0.0358, quantile scaler, and 19 principal components. These parameters result in an AUROC of 0.9705 and 0.9495 as the maximum F1 score compounded by a recall and precision of 0.9920 and 0.9104 when a threshold on the anomaly score of -0.1807 is used. For the model trained on the 2018 dataset, an AUROC of 0.9420 is achieved when using the quantile scaler, followed by a feature reduction with 34 components and a model with an rbf as the kernel function, 0.7496 as its coëfficient and ν of 0.0035 as a regularisation term. This is the highest score achieved on the 2018 dataset. With a threshold of 8.284e-4 on the anomaly score, the maximum F1 score is reached of 0.9296 with a corresponding 0.9323 as recall and 0.9268 as precision. On the contrary, the one-class SVM has up to two orders of magnitude lower inference throughput than the other algorithms with a similar training time as the autoencoder. A decrease of 33.93% is obtained when evaluating the model trained on the 2017 dataset without prior adaptation on the 2018 dataset, resulting in an AUROC of 0.6412. On the contrary, the model trained on the 2018 dataset only decreased 17.85% in the inter-dataset evaluation to an AUROC of 0.7739, this is the best result of all four algorithms on the inter-dataset evaluation strategy. On average, the AUROC and accuracy still decreased respectively 25.89% and 17.84%.



(a) CIC-IDS-2017 (b) CSE-CIC-IDS-2018

Figure 2.5: Overview of the cumulative anomaly score distribution for the different attack classes for the autoencoder in intra-dataset evaluation.

2.5 Discussion

Results have been presented in section 2.4 but are focused on documenting the most important and interesting ones. This section analyses those results in more depth and discusses multiple findings. First, the individual or baseline results are discussed before moving forward to the analysis of the inter-dataset evaluation strategy.

2.5.1 Intra-Dataset Evaluation

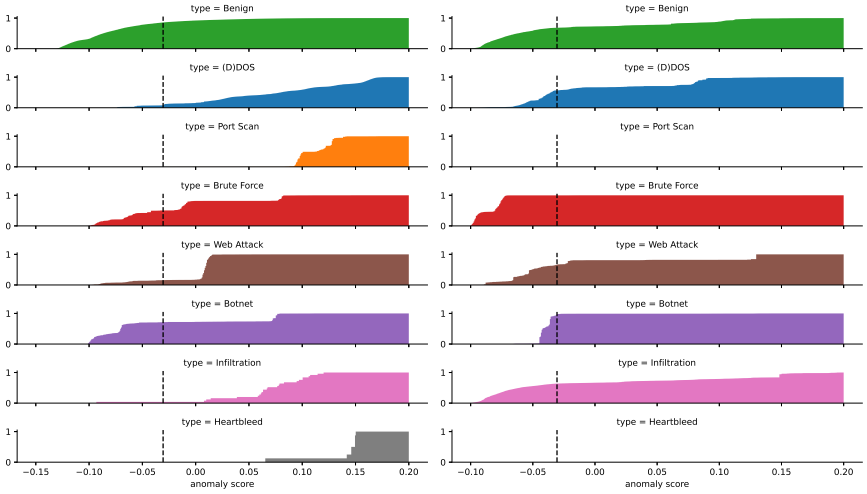
A summary of the intra-dataset classification performance is given in the first two rows for each algorithm in table 2.2. The AUROC scores on CIC-IDS-2017 range from 0.94 to 0.98 with the best performing algorithm being the autoencoder, closely followed by the one-class SVM. Even a slight improvement in the AUROC will result in a significant decrease in false positives or an increase in true positives which in result creates a better classifier. In the left-top corner of figure 2.4 the ROC curves are plotted of the best performing model for each of the four algorithms on the 2017 dataset. There are 2 scenarios visible in which the model outperforms the rest. First, when a low fpr is key, at the cost of missing a few attacks, the one-class SVM achieves the highest score. On the contrary, when the priority is detecting all attacks at the cost of more false alarms, the autoencoder prevails. Furthermore, the few samples of heartbleed, which can be used as a proxy for zero-day attacks, are reliably classified as fraud in CIC-IDS-2017, see the bottom graph in figure 2.5a. Therefore, unsupervised NIDS proved their ability to detect unknown attacks as long as the malicious traffic differs from benign traffic.

The classification performance for all algorithms is significantly lower on the CSE-CIC-IDS-2018 dataset with AUROC scores between 0.85 and 0.94 with as best performing algorithm the one-class SVM followed by the autoencoder. Figure 2.5 plots the cumulative distribution of the anomaly score for each of the attack classes and the benign traffic side-by-side for both datasets for easy comparison between the attack classes as well as between the datasets. The dashed line represents the employed threshold on the anomaly score used to obtain the maximum F1 score. All flows with an anomaly score smaller than the threshold left of the dashed line are marked as normal, while flows with a higher anomaly score than the threshold are classified as fraud. The graphs are plotted using the data of the autoencoder, similar graphs for the other algorithms are accessible online at https://gitlab.ilabt.imec.be/mverkerk/cic-ids-2018. Analysis of figure 2.5 shows two root causes for the decline in classification performance between the 2017 and 2018 dataset. First, the port scan attack class, which proved easily detectable in the 2017 dataset, disappeared entirely in the 2018 dataset. Secondly, the results demonstrate that the fairly easy detectable infiltration attack becomes more challenging in the 2018 dataset, and because of its larger share, it also weighs more through on the end result. Together they account for the decrease in AUROC. Contrarily to the 2017 results, the one-class SVM outperforms the others over the whole line, both in terms of fpr as tpr, see figure 2.4 in the right-bottom corner.

Table 2.3 gives an overview of the computational complexity as measured in the execution time of training and inference of the model. As expected, the time needed to train the model on the 2018 dataset is a bit longer due to the larger validation set. The time the models need to classify the test set is more diverse. PCA and autoencoder even increased their throughput because the inference time did not triple consistently with the size test set. While the isolation forest inference time changed accordingly to the test set size, the execution time of the one-class SVM exploded most likely due to different hyperparameters and the double as many principal components kept in the preprocessing pipeline. For use in operational applications, the throughput during inference is most important to serve as a real-time IDS. The used hardware allowed to classify up to 100.000s of flows per second, proving suitable for high-speed networks.

2.5.2 Inter-Dataset Evaluation

This chapter aims to evaluate the generalization strength of promising algorithms for anomaly-based NIDS, more specifically unsupervised techniques. Therefore a novel inter-dataset evaluation strategy is used to train a model on the first dataset and evaluate it on the second dataset. Ideally, the classification performances should be similar, however, on average the AUROC decreased by 30.45%. In the best case, only a decrease of 17.85% was observed for the one-class SVM model trained on the 2018 dataset but a decrease of 37.49% in the worst case for the autoencoder also trained on the 2018 dataset. Similarly, the accuracy consistently dropped on average 25.63% between the intra- and inter-dataset evaluation strategy. While this is a vast drop in classification performance, all the final models still perform significantly better than a completely random classifier and are thus able to generalize to a certain degree. Figure 2.4 plots the roc curves of the four evaluated algorithms trained on the 2017 dataset and evaluated on the 2018



(a) CIC-IDS-2017 (b) CSE-CIC-IDS-2018

Figure 2.6: The cumulative anomaly score distribution for the different attack classes for the same isolation forest model trained on CIC-IDS-2017 and evaluated in intra- and inter-dataset setup.

dataset in the bottom-left corner and for the models trained on the 2018 dataset and evaluated on the 2017 dataset in the top-right corner. All curves follow a similar trend except for the best generalizing one-class SVM model that outperforms the others. Analogous to the decline in classification performance in the intra-dataset setup, the decrease in the inter-dataset setup can be explained. Similar to figure 2.5, figure 2.6 plots the cumulative distribution of the anomaly score for different attack classes and benign traffic but using the data from the isolation forest trained on the 2017 dataset and evaluated on the 2017 and 2018 dataset. This figure clearly visualizes the disappearance of the previously easily detectable port scan attack and the more challenging detection of infiltration attacks. The heartbleed attack also disappeared, but because there are only a few samples present in the 2017 dataset, this barely influences the results. A combination of a shift in the distribution of attack classes between the 2017 and 2018 test set together with the model hyper-parameters selected on a validation set not representable for the distribution of attack classes in both datasets, explains the decrease in AUROC. The last observation that can be made from the plot is that the learned threshold for classification is not directly transferable to the second dataset. If the dashed line would be shifted to the left, thus taking a lower threshold, more malicious flows would be correctly flagged and result in a better F1 score than the current 0.4845.

In the inter-dataset setting, the computational complexity is not separately discussed again because the same model as trained in the intra-dataset setup without any adaptation is used, causing the same results.

2.6 Future Work

Current state-of-the-art algorithms for unsupervised anomaly-based NIDS are losing in the best case over 25% of their classification performance when fed with unseen but related data without any prior measures taken. This demands further research with the goal of improving the generalization performance of the proposed models. In the last decade, research has been focusing too much on achieving marginal gains on synthetic datasets and reporting them as advances while they have little effect for real-world applications. The inter-dataset evaluation strategy proposed in this chapter is a strong candidate for adoption in future developments and, if necessary, further adapted to estimate the generalization strength of a model. By providing the research community with new tools to evaluate the generalization strength of proposed models, this work does not only try to close the gap in classification performance between academic prototypes and real-world applications but also raises awareness among researchers active in the field for this open challenge. Two main approaches exist to tackle this issue. First, machine learning models can only perform as well as the data they are trained on. The continuous development of new, high-quality academic datasets consisting of realistic attacks is required. Second, the used techniques can be further improved to be less prone to overfitting and actually learning a high-level representation of complex intrusions. For example, a more sophisticated feature engineering approach could be used to reduce the risk of overfitting. Another possibility is to validate that fine-tuning the hyper-parameters of the pre-trained model with a very small number of flows out of the unseen dataset, improves the generalization strength. Furthermore, it would be favorable to have a similar study focusing on the generalization strength of supervised algorithms and comparing them with the results presented in this chapter. We expect that unsupervised algorithms are overall more resilient to overfitting and thus achieve a higher generalization power. This needs to be validated by future work.

2.7 Conclusion

This chapter started with evaluating four unsupervised algorithms on two realistic and recent datasets. Then these results served as a baseline for estimating the generalization strength of the models with a novel proposed inter-dataset evaluation strategy. The unsupervised algorithms were able to achieve high classification scores on the individual datasets with high throughput rates up to 100.000s of flows per second. On average, the one-class SVM yielded the highest AUROC scores of 0.9705 on CIC-IDS-2017 and 0.9420 on CSE-CIC-IDS-2018, closely followed by the autoencoder. Even the most lightweight algorithm tested, PCA achieved a good classification performance with the lowest recorded AUROC of 0.8494 on CSE-CIC-IDS-2018. Moreover, unsupervised NIDS proved their capability of detecting unknown zero-day attacks as long as the malicious traffic differs from normal traffic.

The second part of this chapter validated if these promising results also withstand when classifying a second related dataset. This increasingly difficult inter-dataset evaluation setup decreased

on average the AUROC and accuracy scores respectively by 30.45% and 25,63%, indicating that models trained on the current state-of-the-art intrusion datasets have a low generalization strength without any measures or adaptations in place. While there is a significant drop in classification performance, all four algorithms still perform better than a completely random classifier and thus are able to generalize the learned patterns to a certain degree. Again, the one-class SVM proved the best with *only* an average 25.89% decrease in AUROC. Above all, the acknowledgment of this generalization challenge can shift the current research focus of obtaining marginal gains on individual datasets in the direction of techniques improving the generalization strength. The proposed inter-dataset evaluation strategy in this chapter is a strong candidate for adaption in future research to estimate the generalization strength of newly developed models.

Bibliography

[1] M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Unsupervised machine learning techniques for network intrusion detection on modern data," in *2020 4th Cyber Security in Networking Conference (CSNet)*. IEEE, 2020, pp. 1–8.

[2] L. Goasduff, "Gartner survey reveals 47% of organizations will increase investments in iot despite the impact of covid-19," https://www.gartner.com/en/newsroom/press-releases/2020-10-29-gartner-survey-reveals-47-percent-of-organizations-will-increase-investments-in-iot-despite-the-impact-of-covid-19-, 2020.

[3] T. Cheng, Y. Lin, Y. Lai, and P. Lin, "Evasion techniques: Sneaking through your intrusion detection/prevention systems," *IEEE Communications Surveys Tutorials*, vol. 14, no. 4, pp. 1011–1020, 2012.

[4] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in https," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 133–140. [Online]. Available: https://doi.org/10.1145/2674005.2674991

[5] Y. Zeng, H. Gu, W. Wei, and Y. Guo, "$deep - full - range$: A deep learning based network encrypted traffic classification and intrusion detection framework," *IEEE Access*, vol. 7, pp. 45 182–45 190, 2019.

[6] S. Canard, A. Diop, N. Kheir, M. Paindavoine, and M. Sabt, "Blindids: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 561–574. [Online]. Available: https://doi.org/10.1145/3052973.3053013

[7] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[8] N. Hubballi and V. Suryanarayanan, "False alarm minimization techniques in signature-based intrusion detection systems: A survey," *Computer Communications*, vol. 49, pp. 1 – 17, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366414001480

[9] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers & Security*, vol. 70, pp. 238 – 254, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404817301165

[10] A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos, "From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3369–3388, 2018.

[11] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on sdn based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, Mar 2019. [Online]. Available: https://doi.org/10.1007/s12083-017-0630-0

[12] J. P. Anderson, "Computer security threat monitoring and surveillance," *Technical Report, James P. Anderson Company*, 1980.

[13] S. Othman, N. Alsohybe, F. Ba-Alwi, and A. Zahary, "Survey on intrusion detection system types," vol. 7, pp. 444–462, 12 2018.

[14] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences*, vol. 9, p. 4396, 10 2019.

[15] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42 – 57, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804512001178

[16] S. Otoum, B. Kantarci, and H. Mouftah, "A Comparative Study of AI-based Intrusion Detection Techniques in Critical Infrastructures," *arXiv:2008.00088 [cs]*, Jul. 2020, arXiv: 2008.00088. [Online]. Available: http://arxiv.org/abs/2008.00088

[17] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *CoRR*, vol. abs/1802.09089, 2018. [Online]. Available: http://arxiv.org/abs/1802.09089

[18] S. Zavrak and M. İskefiyeli, "Anomaly-based intrusion detection from network flow features using variational autoencoder," *IEEE Access*, vol. 8, pp. 108 346–108 358, 2020.

[19] A. M. Vartouni, S. S. Kashi, and M. Teshnehlab, "An anomaly detection method to detect web attacks using stacked auto-encoder," in *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, 2018, pp. 131–134.

[20] Q. T. Nguyen, K. Phuc Tran, P. Castagliola, T. Thu Huong, M. K. Nguyen, and S. Lardjane, "Nested one-class support vector machines for network intrusion detection," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, 2018, pp. 7–12.

[21] N. Takeishi, "Shapley values of reconstruction errors of pca for explaining anomaly detection," in *2019 International Conference on Data Mining Workshops (ICDMW)*, 2019, pp. 793–798.

[22] Y. Otoum and A. Nayak, "AS-IDS: Anomaly and Signature Based IDS for the Internet of Things," *Journal of Network and Systems Management*, vol. 29, no. 3, p. 23, Mar. 2021. [Online]. Available: https://doi.org/10.1007/s10922-021-09589-6

[23] J. Dromard, G. Roudière, and P. Owezarski, "Online and scalable unsupervised network anomaly detection method," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 34–47, 2017.

[24] A. Safari Khatouni, N. Seddigh, B. Nandy, and N. Zincir-Heywood, "Machine Learning Based Classification Accuracy of Encrypted Service Channels: Analysis of Various Factors," *Journal of Network and Systems Management*, vol. 29, no. 1, p. 8, Oct. 2020. [Online]. Available: https://doi.org/10.1007/s10922-020-09566-5

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

[26] J. L. Leevy and T. M. Khoshgoftaar, "A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data," *Journal of Big Data*, vol. 7, no. 1, p. 104, Nov 2020. [Online]. Available: https://doi.org/10.1186/s40537-020-00382-x

[27] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. n/a, no. n/a, p. e4150. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4150

[28] M. Al-Omari, M. Rawashdeh, F. Qutaishat, M. Alshira'H, and N. Ababneh, "An Intelligent Tree-Based Intrusion Detection Model for Cyber Security," *Journal of Network and Systems Management*, vol. 29, no. 2, p. 20, Feb. 2021. [Online]. Available: https://doi.org/10.1007/s10922-021-09591-y

[29] M. Aloqaily, S. Otoum, I. A. Ridhawi, and Y. Jararweh, "An intrusion detection system for connected vehicles in smart cities," *Recent advances on security and privacy in Intelligent Transportation Systems*, vol. 90, p. 101842, Jul. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870519301131

[30] A. Narayanan and V. Shmatikov, "How to break anonymity of the netflix prize dataset," *ArXiv*, vol. abs/cs/0610105, 2006.

[31] T. U. of California, "Kdd cup 1999 data," no. 28, Oct. 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[32] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[33] D. Protic, "Review of kdd cup '99, nsl-kdd and kyoto 2006+ datasets," *Vojnotehnicki glasnik*, vol. 66, pp. 580–596, 07 2018.

[34] I. Sharafaldin, A. Gharib, A. Habibi Lashkari, and A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, vol. 2017, pp. 177–200, 01 2017.

[35] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," 01 2018, pp. 108–116.

[36] T. U. of California, "Cicflowmeter." [Online]. Available: https://github.com/ahlashkari/CICFlowMeter

[37] A. Habibi Lashkari, G. Draper Gil, M. Mamun, and A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related features," 02 2016.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[39] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[40] J. Shlens, "A tutorial on principal component analysis," 2014.

[41] F. T. Liu, K. Ting, and Z.-H. Zhou, "Isolation forest," 01 2009, pp. 413 – 422.

[42] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, 1987, pp. 318–362.

[43] E. Plaut, "From principal subspaces to principal components with linear autoencoders," 2018.

[44] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[45] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[46] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," vol. 12, 01 1999, pp. 582–588.

[47] Y. Xu and R. Goodacre, "On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning," *Journal of Analysis and Testing*, vol. 2, 10 2018.

[48] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

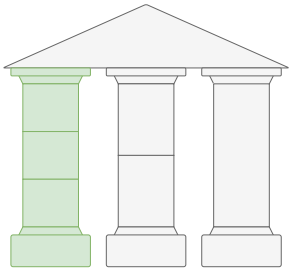
[49] M. Verkerken, "Gitlab repository containing code for paper "towards model generalization for intrusion detection: Unsupervised machine learning techniques"," https://gitlab.ilabt.imec.be/mverkerk/cic-ids-2018, 2021.

[50] A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145 – 1159, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320396001422

Appendix

2.A Overview features CIC-IDS-2017 and CSE-CIC-IDS-2018

Number	Name	Kept	Modified	Dropped	Note
1	Flow ID			○	Remove dataset specific info
2	Source IP			○	Remove dataset specific info
3	Source Port			○	Remove dataset specific info
4	Destination IP			○	Remove dataset specific info
5	Destination Port			○	Remove dataset specific info
6	Protocol	○			
7	Timestamp			○	Remove dataset specific info
8	Flow Duration	○			
9-10	Total Fwd/Bwd Packets	○			
11-12	Total Length of Fwd/Bwd Packets	○			
13-16	Fwd Packet Length Max/Min/Mean/Std	○			
17-20	Bwd Packet Length Max/Min/Mean/Std	○			
21	Flow Bytes/s		○		Remove NaN / Infinity
22	Flow Packets/s		○		Remove NaN / Infinity
23-26	Flow IAT Mean/Std/Max/Min	○			
27-31	Fwd IAT Total/Mean/Std/Max/Min	○			
32-36	Bwd IAT Total/Mean/Std/Max/Min	○			
37	Fwd PSH Flags	○			
38	Bwd PSH Flags			○	No variance
39	Fwd URG Flags			○	No variance
40	Bwd URG Flags			○	No variance
41-42	Fwd/Bwd Header Length	○			
43-44	Fwd/Bwd Packets/s	○			
45-49	Packet Length Min/Max/Mean/Std/Variance	○			
50	FIN Flag Count	○			
51	SYN Flag Count	○			
52	RST Flag Count	○			
53	PSH Flag Count	○			
54	ACK Flag Count	○			
55	URG Flag Count	○			
56	CWE Flag Count			○	No variance
57	ECE Flag Count	○			
58	Down/Up Ratio	○			
59	Average Packet Size	○			
60-61	Avg Fwd/Bwd Segment Size	○			
62	Fwd Header Length.1			○	Duplicate column
63	Fwd Avg Bytes/Bulk			○	No variance
64	Fwd Avg Packets/Bulk			○	No variance
65	Fwd Avg Bulk Rate			○	No variance
66	Bwd Avg Bytes/Bulk			○	No variance
67	Bwd Avg Packets/Bulk			○	No variance
68	Bwd Avg Bulk Rate			○	No variance
69-70	Subflow Fwd/Bwd Packets	○			
71-72	Subflow Fwd/Bwd Bytes	○			
73	Init_Win_bytes_forward	○			
74	Init_Win_bytes_backward	○			
75	act_data_pkt_fwd	○			
76	min_seg_size_forward	○			
77-80	Active Mean/Std/Max/Min	○			
81-84	Idle Mean/Std/Max/Min	○			
85	Label			○	Evaluation only



3

A Novel Multi-Stage Approach for Hierarchical Intrusion Detection

This chapter is situated within the first pillar of this dissertation, AI for Cybersecurity, and builds upon insights from the benchmark evaluations introduced in Chapter 2. Motivated by the generalization limitations of existing single-model approaches, it proposes a multi-stage ML-NIDS architecture combining anomaly detection, multi-class classification, and zero-day attack detection into a three-stage design. This architecture improves classification performance, supports bandwidth-efficient n-tier deployments, and enables flexible threshold-based tuning without the need for model retraining. Using modern flow-based datasets, the system is evaluated and compared against both a baseline open-set classifier and a state-of-the-art multi-stage IDS. Results demonstrate that it outperforms existing solutions across most metrics, including classification performance and robustness to zero-day attacks, while enabling hierarchical multi-tier deployments. This deployment strategy offers additional benefits such as reduced bandwidth usage and improved privacy preservation. The work presented in this chapter is the result of a collaborative effort with the high-speed networking research group at NYCU, Taiwan.

This chapter contributes to Research Questions two and three by addressing the generalization limitations of prior ML-NIDS from a model-centric perspective (RQ2), proposing a multi-stage architecture for hierarchical IDS (RQ3). Chapter 4 will evaluate the practical deployment of the proposed multi-stage NIDS in a modern cloud environment.

⋆ ⋆ ⋆

M. Verkerken, L. D'hooge, D. Sudyana, Y. Lin, T. Wauters, B. Volckaert, and F. De Turck

Published in IEEE Transactions on Network and Service Management, September 2023.

Abstract An intrusion detection system (IDS), traditionally an example of an effective security monitoring system, is facing significant challenges due to the ongoing digitization of our modern society. The growing number and variety of connected devices are not only causing a continuous emergence of new threats that are not recognized by existing systems, but the amount of data to be monitored is also exceeding the capabilities of a single system. This raises the need for a scalable IDS capable of detecting unknown, zero-day attacks. In this chapter, a novel multi-stage approach for hierarchical intrusion detection is proposed. The proposed approach is validated on the public benchmark datasets, CIC-IDS-2017 and CSE-CIC-IDS-2018. Results demonstrate that our proposed approach, besides effective and robust zero-day detection, outperforms both the baseline and existing approaches, achieving high classification performance, up to 96% balanced accuracy. Additionally, the proposed approach is easily adaptable without any retraining and takes advantage of n-tier deployments to reduce bandwidth and computational requirements while preserving privacy constraints. The best-performing models with a balanced set of thresholds correctly classified 87%, or 41 out of 47 zero-day attacks, while reducing the bandwidth requirements up to 69%.

3.1 Introduction

Our society is continuously exposed to an increased risk of cybersecurity threats due to the ongoing digitization in the modern world [1]. The never-ending growing number and variety of interconnected devices, including critical systems such as power grids, does not only expand the attack surface for a malicious actor but is also negatively affecting the possible consequences in case of a successful attack [2]. Furthermore, the increasing generated load on existing security monitoring systems is exceeding single system capabilities and challenging their scalability to detect threats in near real-time [3]. As a result, the historical arms race between attackers and defenders is shifting advantageously towards the attackers. This demands consistent efforts from the research community to further improve the existing defenses in place as well as develop novel approaches that contribute to the mitigation of the cybersecurity risk.

Traditional security mechanisms such as a firewall are effective at detecting specific types of attacks but are unable to detect unknown or more advanced attacks. Intrusion detection systems (IDS) are often deployed as a second line of defense and are an example of a security monitoring system capable of detecting known as well as unknown and more sophisticated attacks. The detection can happen before, during, or after an attack is executed. In case the system is not only capable of detecting such an intrusion but also actively prevents the attack from succeeding, it is called an intrusion prevention system (IPS). An IDS and IPS can be categorized by the source of the

input used for detection. Host-based intrusion detection systems (HIDS) use features gathered from the host machine such as resource usage and system calls, therefore a monitoring module needs to be deployed on every single device that needs to be secured. On the other hand, a network intrusion detection system (NIDS) is deployed on a particular node in the network and monitors all traffic passing through it. When a combination of both sources is used as input for the detection, it is called a hybrid intrusion detection system. These systems can also be differentiated by the applied method for detection. Signature or misuse-based systems rely on a database consisting of patterns or signatures of known attacks. This technique is an effective tool for detecting known attacks with few false alerts but fails to detect any attack not present in the signature database and therefore is unable to detect zero-day attacks. On the contrary, anomaly-based detection models normal behavior instead of the attack itself, and everything deviating too much from normal is flagged as malicious. This approach allows the detection of both known and unknown attacks as long as they diverge sufficiently from the learned baseline. Anomaly-based detection often relies on machine-learning techniques to learn the normal baseline from data.

Currently proposed IDS solutions often rely on a single machine learning model for either attack detection or classification. This poses multiple challenges. First, a single model generally excels either in attack detection or classification. Using a combination of multiple models particularly trained for a specific task could potentially improve the classification performance. Secondly, all samples need to transmitted to and processed by this single model. In case the monitored network is distributed, this will not only lead to high computational costs but also have high bandwidth requirements, leading to increased latency. Last, the proposed models often lack the ability to detect unknown or zero-day attacks.

In this chapter, a novel multi-stage approach for hierarchical intrusion detection is proposed. Fundamentally, this approach is applicable for each type of IDS, independent of the source of the input or used detection method, considering the characteristics of each layer are met. Our novel approach features improved classification performance over both a single model baseline and existing multi-stage approaches, in addition to its ability to effectively detect zero-day attacks. The classification is performed by a system consisting of three stages. The first stage performs a lightweight outlier detection, associating an anomaly score to each event. By only forwarding events to the second stage with an anomaly score above a certain threshold, this layer acts as a filter. This allows the use of more computationally expensive methods in the next layer as it will only be applied to a small share of the total number of events. The second stage classifies each of the suspicious events from the first stage to a known attack type with a certain confidence. Events with low prediction confidence, most likely do not belong to any of the known attack classes and are forwarded to the third stage. This last stage reuses the anomaly score of the first stage to separate miss-classified benign events from the first stage and zero-day attacks. Events with an anomaly score higher than another threshold which is higher than the previous threshold in the first stage, are flagged as an unknown attack while events with a lower score are corrected as benign. Since each layer employs a threshold to define its prediction, the final performance can easily be tuned by adjusting these values without retraining any of the used models. This flex-

ibility also allows adapting the models' specific trade-offs to changing requirements over time. Finally, the novel approach is designed to take advantage of a hierarchical deployment. Each of the stages can either be deployed separately or combined. This empowers an n-tier deployment of our novel proposed approach that minimizes bandwidth requirements and latency associated with the predictions. In case the first stage is deployed close to the network being monitored, a privacy-aware operation is ensured as only suspicious events are forwarded, retaining most of the benign traffic locally.

The main contributions of this chapter are three-fold.

- We propose a novel multi-stage approach for hierarchical intrusion detection performing both binary and multi-class detection. Our highly adaptable novel approach is specifically designed to empower a multi-tier deployment to minimize latency and bandwidth requirements while preserving privacy constraints. Furthermore, both known and zero-day attacks can be detected.
- A hyperparameter optimization using machine learning best practices is performed for two unsupervised, autoencoder and one-class support vector machine, and two supervised, random forest and neural network, algorithms for the first and second stage, respectively.
- An extensive validation and analysis of our novel proposed approach is performed on modern flow-based network intrusion datasets, CIC-IDS-2017 and CSE-CIC-IDS-2018 [4], against both a single baseline model and the existing state-of-the-art multi-stage approach.

The remainder of this chapter is structured as follows. First, the related work regarding IDS and more specifically multi-stage IDS is discussed in Section 3.2. Afterward, the novel multi-stage approach for hierarchical intrusion detection is presented in section 3.3.1, followed by section 3.4 describing the methodology used to validate the newly proposed approach, ensuring sound and reproducible results. In section 3.5 the results of both the baseline, all intermediate stages, and the final novel approach on a modern flow-based network dataset are presented. An extensive analysis and discussion of these results are laid out in section 3.6. Future work is listed in section 3.7 before stating a final conclusion in section 3.8.

3.2 Related Work

In the literature several studies regarding multi-layer or multi-stage IDS exist, starting from a different definition. A first definition refers to a hierarchical context where each layer has different input data available to execute the detection or classification, the final prediction is then a combination of each tier's prediction. Zhang et al. [5] propose an IDS to detect cyberattacks in smart grids. This smart grid exists of three layers where each layer has the ability to monitor a unique set of features used for attack detection. Through internal communication between the different layers, the smart grid is able to identify malicious traffic. Similarly, Ali and Yousaf [6] proposed an approach composed of three tiers to detect intrusions in software-defined networks (SDN). The

first tier validates user authentication through an RFID tag and encrypted signatures using routers as edge devices. Next, the second tier located on switches validates the raw network packets using fuzzy filtering. In the third tier, located in network controllers, the reconstructed flows from the raw packets are used as input in a convolutional neural network (CNN) for the detection of malicious traffic.

A second definition found in the literature describes multi-stage approaches as a cascade of detection and classification methods on the same input data. The goal is often to achieve a higher classification performance. Li et al. [7] proposed such a cascade to classify network traffic to the correct attack type. The first stage consists of a collection of binary classifiers whose output is aggregated to form a prediction. Traffic for which the aggregated prediction is uncertain is sent to the second stage to determine the correct class using the k-nearest neighbors model (KNN). Likewise, Pajouh et al. [8][9] used a two-stage feature reduction method followed by a two-stage classification approach chaining the naive Bayes algorithm and certainty factor KNN (CF-KNN) to classify network traffic. All traffic classified as benign by the naive Bayes algorithm is forwarded to the CF-KNN for further investigation. The final prediction is formed by combining both stages. On the contrary, Al-Yaseen et al. [10] chained multiple stages with each stage consisting of a classifier able to detect a single attack type instead of using a second layer to reclassify predictions with low confidence. This chain forms a waterfall pattern where each sample propagates to the next classifier until successful detection of an attack type. In case the last classifier also fails to classify the sample as a known attack, the sample is classified as an unknown attack. The study conducted by Ji et al. [11] proposed a two-stage model with the same goal of improving the classification performance but uses rule-based detection followed by an anomaly-based model for the first and second stage, respectively. The approach proposed by Khan et al. [12] consists of two layers with both layers containing a deep autoencoder (DAE) and a soft-max classifier. The DAE is used to construct the latent space of the input data in an unsupervised manner. On top of this latent space, a soft-max classifier is placed which is fine-tuned using a limited set of labeled data. The first layer will output an anomaly probability which is then used as an extra feature in the second layer. The second layer performs the final classification based on both the input data and anomaly probability outputted by the first layer.

Next to achieving a higher classification performance by employing multiple detection methods, the third cluster of studies exists that relies on a combination of anomaly detection, often based on unsupervised machine learning techniques, and a multi-class classifier. Here the goal of the first stage is to filter out suspicious samples in a lightweight manner, which are then sent to the next, more computational complex stage for attack type classification. The goal of these studies is to achieve a high classification performance while reducing latency, bandwidth and computational requirements, often combined with unknown attack detection capabilities [13]. Divyatmika and Sreekesh [14] use a three-stage approach to classify network traffic, where the first stage relies on KNN model to compare incoming traffic against the baseline traffic of the network. New behavior and attacks are not matched and are sent to the following stages where the multi-layer perceptron (MLP) algorithm and reinforcement learning is used for misuse and anomaly detec-

Table 3.1: Taxonomy of the related work

Study	Detection	Classification	Stages	Zero-Day	Hierarchical	Computational Reduction
Zhang et al. [5]	±	✓	3	✗	✓	✗
Ali and Yousaf [6]	±	✓	3	✗	✓	✗
Li et al. [7]	✗	✓	2	✗	✗	✗
Pajouh et al. [8][9]	✗	✓	2	✗	✗	✗
Al-Yaseen et al. [10]	✗	✓	5	✓	✗	✗
Ji et al. [11]	✓	✓	2	✗	✗	✗
Khan et al. [12]	✓	✓	2	✗	✗	✗
Divyatmika and Sreekesh [14]	✓	✓	3	✓	✗	±
Umer et al. [15]	✓	✓	2	✗	✗	✗
Abuadlla et al. [16]	✓	✓	2	±	✗	✗
Bovenzi et al. [17]	✓	✓	2	✓	✓	±
Verkerken et al.	✓	✓	3	✓	✓	✓

tion, respectively. The first stage is acting as a filter to reduce the load by the computational more expensive algorithms in the succeeding stages. The study by Umer et al. [15] presents a multi-stage model for next-generation networks consisting of an anomaly detection and a multi-class classifier relying on solely unsupervised machine-learning techniques. The first layer uses a one-class support vector machine (OC-SVM) for binary detection. Only traffic predicted as malicious by the first layer is classified by the second layer using a self-organizing map. Abuadlla et al. [16] present an easy expandable two-stage model. Both stages rely on a neural network (NN) for respectively, anomaly detection and multi-class classification in the first and second stages. The study mentions the ability to detect unknown attacks but does not further specify in detail how the supervised NN achieves this in the absence of labeled samples of unknown attacks. The two-stage hierarchical approach proposed by Bovenzi et al. [17] consists of a multi-modal DAE and soft output classifiers in the first and second stage, respectively. The soft output classifier in the second stage is able to detect unknown attacks using a threshold on the confidence of the prediction. In case the confidence of a sample belonging to a known attack class or benign traffic is lower than the threshold, the sample is predicted as an unknown attack. The multi-class classifier is trained using the open-set approach, removing one of the known attack types from the training data and considering it as an unknown attack. By repeating this approach for each of the known attack types, the threshold on the confidence can be optimized. Further, this model is optimized for a distributed and privacy-preserving deployment with the need for limited retraining to adjust performance trade-offs. This study is the closest to our novel approach and is used as the current state-of-the-art multi-stage approach for comparative purposes.

Other studies in the literature also describe their work as multi-layer or multi-stage but do not refer to classification in multiple steps or a chain of classifiers. For example, Injadat et al. [18] simply refers to the multiple steps in a data processing pipeline, such as preprocessing, feature selection, hyperparameter optimization, and classification, to claim her proposed model as multi-stage.

While few of the previously discussed related works mentioned the reduction of the required computational capacity of the proposed architecture, none of them provided experimental results or

analyzed this statement more thoroughly. This chapter discusses the reduction on basis of the share of samples propagating to the second layer, thus requiring extra computational resources and bandwidth in a hierarchical deployment. A taxonomy of the related work can be found in table 3.1. For each study it is examined if separate anomaly detection and classification are obtainable, the number of used stages, the ability to detect zero-day attacks, the possibility for a hierarchical deployment, and achieving reduction of required computational capacity. The taxonomy clearly shows the unique characteristics of this chapter.

The use of machine learning techniques for intrusion detection is a well-studied topic by the research community. Several surveys [19][20][21] advocate the usage of both supervised and unsupervised methods to assist existing IDS or even replace them completely. More specifically, promising results have previously been achieved by employing AE [22] [23] and OC-SVM [24] to detect anomalies in streams of network traffic. Similarly, tree-based algorithms, such as an RF [25] [26], and deep learning algorithms, in particular convolutional neural networks [27] [28] and artificial neural networks [29] have been confirmed to achieve promising results on academic benchmark datasets. This chapter evaluates the use of an AE and OC-SVM for anomaly detection and RF and NN for multi-class classification, respectively, in the first and second stage of the proposed novel multi-stage approach. Furthermore, a RF trained using an open-set option for zero-day detection serves as a single model baseline.

3.3 Novel Approach

In this section, the novel multi-stage approach for hierarchical intrusion detection is described. First, the overall architecture is introduced and the design choices made throughout the development are explained in depth. Afterward, the stages composing the overall architecture are individually discussed. Finally, the benefits of a hierarchical deployment are highlighted as well as the implementation choices to be made for an operational system.

3.3.1 General Architecture

The multi-stage hierarchical architecture proposed in this chapter consists of three stages with each a distinct characteristic and objective. Figure 3.1 presents the overall architecture. The feature vector denoted as X serves as an input to the system and is sent to the first stage. The first stage consists of an anomaly detector that outputs an anomaly score, λ_B, specifically a high value indicates a high probability that X is not benign and thus an intrusion. If this anomaly score is lower than a threshold τ_B, the model is confident enough to predict the sample as benign and no further processing is necessary. However, if the anomaly score is higher than τ_B, the sample is forwarded to the second stage where an attack classifier predicts if the sample belongs to one of the known attack classes (ATK_i). If the attack classifier fails to match the sample to a known attack class with a certainty higher than a threshold τ_M, the sample is sent to the third and last stage. The last stage or extension stage does not introduce another classifier but rather reuses

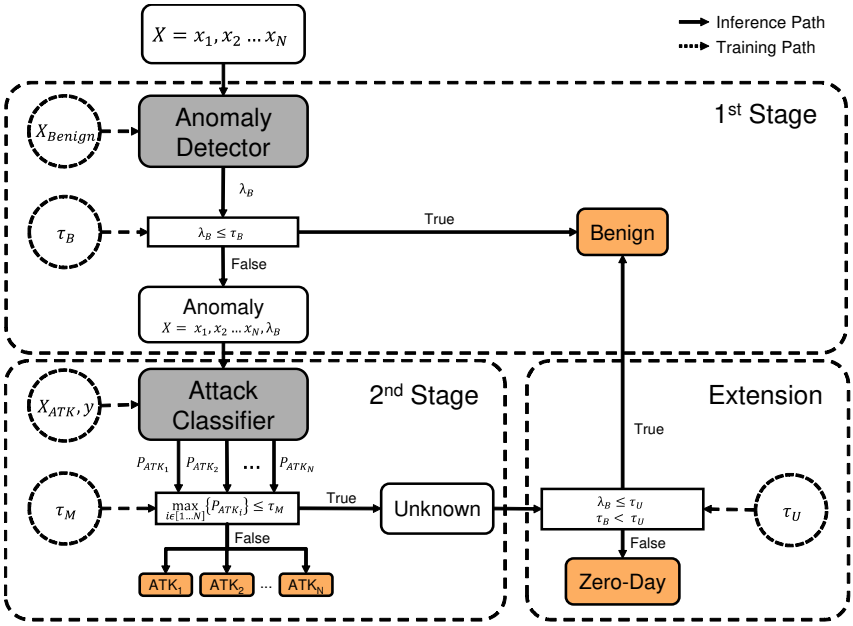


Figure 3.1: The proposed architecture of the multi-stage hierarchical intrusion detection system.

the anomaly score λ_B outputted by the first layer. In case the anomaly score is lower than the threshold τ_U, the output of the first layer is corrected by eventually classifying the sample as benign, on the contrary, the sample is predicted as an unknown or zero-day attack if the anomaly score is higher than τ_U.

Additionally, live adjustment of the thresholds τ_B, τ_M, and τ_U enables our novel approach to adjust its classification performance in real-time without the need for any retraining. This is done by balancing the trade-off between the number of false positives and false negatives in each stage.

The proposed architecture is developed with a scalable hierarchical deployment in mind. The first stage acts as a lightweight filter with minimal hardware requirements, forwarding only suspicious samples. This results in minimal computational cost for most of the benign traffic as these are not subjected to further analysis by the subsequent stages. Ideally, the first stage is located close to the network being monitored, for example in a fog or edge device, while the other stages can be deployed further away, for instance in a centralized cloud. Subsection 3.3.5 describes in more detail the configuration and benefits of a hierarchical deployment.

$$
\begin{aligned}
T_{stage1} &= \mathcal{O}(T_B) \\
T_{stage2} &= \mathcal{O}(T_B + T_M) \\
T_{stage3} &= \mathcal{O}(T_B + T_M + T_U) \\
T_{total} &= \mathcal{O}(T_B + \alpha T_M + \alpha\beta T_U) \\
&\Rightarrow \mathcal{O}(T_B + T_M)
\end{aligned}
\tag{3.1}
$$

where: T_B = Time Complexity Anomaly Detector
T_M = Time Complexity Attack Classifier
T_U = Time Complexity Extension Stage
α = fraction of flows forwarded by 1st stage
β = fraction of flows forwarded by 2nd stage

Equation 3.1 uses the big O notation to describe the computational complexity of our novel hierarchical approach, regardless of the algorithms used in each stage. The sum of all the stages creates the final performance, which will be impacted accordingly by the algorithms that are ultimately selected at each stage. Note that the computational complexity of the extension stage is equal to $\mathcal{O}(1)$ and thus has no impact on the final complexity since it reuses the anomaly score of the first stage with a new threshold.

While our proposed architecture is independent of both the used input source and employed classification methods, this chapter evaluates the novel approach with machine-learning models for detection and classification on a modern flow-based network intrusion detection dataset.

3.3.2 Stage 1: Anomaly Detection

The first stage has the goal to filter out malicious samples in a computationally efficient manner. This way the number of samples that need to be analyzed in the subsequent stages is greatly reduced. Because the first stage is applied to each sample and ideally located close to the monitored network generating the input, the binary classifier is required to perform the anomaly detection in a lightweight manner on limited hardware.

The binary model in the first stage is exclusively trained on benign data. This allows the anomaly detector to learn a representation of the normal behavior of the monitored network. During training, the optimal hyper-parameters are selected using the area under the receiver operating characteristic (AUROC) as a validation metric with a validation set consisting of both malicious and benign data. The AUROC is selected because it is independent of the eventually employed threshold τ_B on the anomaly score, which determines the final prediction. After the hyper-parameter optimization of a model, multiple candidate thresholds τ_B are selected by computing the F1 till F9 metric for every unique value encountered in the anomaly scores of the validation set. The value corresponding to a maximum f-score is added to the list of candidate thresholds for this particular model. The eventually optimal threshold depends on the intended result. Since τ_B balances

both a performance and computational tradeoff, it is crucial to be carefully set. The impact of the candidate thresholds is analyzed more in detail when the overall performance of the multi-stage approach is assessed.

This chapter evaluates unsupervised machine-learning techniques to execute the anomaly detection because of their ability to detect both known and unknown intrusions, even when only trained on benign data. But in practice, any technique that outputs an anomaly score is suited.

3.3.3 Stage 2: Multi-Class Classification

The second stage attempts to classify the samples predicted as malicious by the first stage to a known attack class using a multi-class classifier. This classifier is trained on exclusively malicious data and is accordingly only able to classify samples to the attack classes present in the training data. A validation set consisting of both malicious and benign data is used to select the optimal hyper-parameters for the model using the F1 score as the validation metric. The classifier outputs a vector, $[P_{ATK_1}, P_{ATK_2}, ..., P_{ATK_N}]$, with the probabilities for each of the known attack classes that the input vector X belongs to that attack. The attack with the highest probability is then the predicted class, except if this probability is lower than the threshold τ_M, the sample is predicted as unknown and forwarded to the last stage. The threshold is set to the value that yields the highest weighted f1-score on the validation set. This threshold is responsible for defining how confident the model needs to be before assigning a sample to a known attack class, as a result, it balances a false positive, false negative tradeoff. Since the validation set is composed of both benign and malicious data, the model is not only forced to correctly classify the malicious samples but also to output a low probability for each of the known attack classes for the benign samples, to achieve a high validation score. Similarly, an unknown attack will be associated with a low probability for each of the known attack classes, which will successfully send the sample to the next stage. This resolves the challenge to detect unknown attacks without corresponding labeled data.

This chapter evaluates supervised machine-learning techniques for the attack classification of known intrusions but could be replaced by any technique able to produce prediction probabilities. However, the model should be able to output if a sample does not correspond to any of the learned attacks by explicitly outputting an extra probability estimation for an unknown class or implicitly by associating low probabilities to each of the known classes.

3.3.4 Stage 3: Extension

The third stage or also extension stage does not implement another classifier but rather reuses the anomaly score outputted by the first stage, to produce its prediction. The main goal of this stage is to reduce the number of false positives, namely benign traffic being classified as an attack while providing the ability to effectively detect zero-day attacks. Currently, one of the main challenges anomaly-based IDS are facing is the high number of false positives.

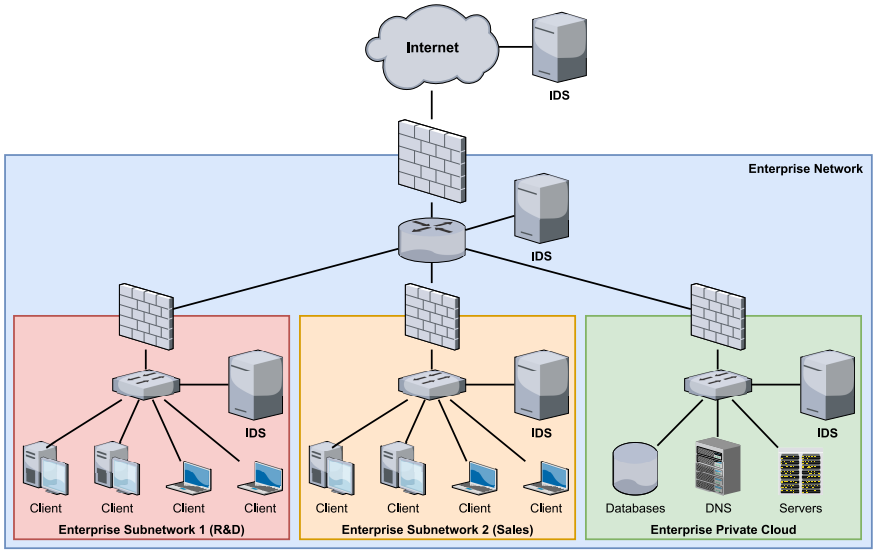


Figure 3.2: Representation of an enterprise network consisting of a three-tier IDS and three local subnetworks: R&D, sales, and private cloud.

Since this stage reuses the model of the first stage, no new training or validation sets are required to train the model. Only the threshold τ_U needs to be defined using the same validation set of the first stage. Because one of the main purposes of this stage is to reduce the number of false positives, the threshold is set using the quantile of the anomaly score of the benign samples. As a result, the maximum false positive rate can approximately be set using this threshold. The third stage is only effective if the threshold in this layer (τ_U) is greater than the threshold in the first stage (τ_B). This way the extension layer can achieve a higher precision because a sample needs a higher anomaly score before being marked as an attack. The resulting samples that are eventually detected as malicious by the extension layer have a high probability to be a zero-day attack or known attack but not trained on in the second stage.

3.3.5 Hierarchical Deployment

Our novel multi-stage approach for intrusion detection is specifically designed to empower an n-tier hierarchical deployment. To illustrate the benefits of a hierarchical deployment we will introduce a simple enterprise network, see figure 3.2, with three local subnetworks secured by a three-tier IDS. To illustrate the advantages of our proposed approach we will use a simple scenario where each stage is placed bottom-up in the three-tier IDS, respectively, the first, second and extension stage are placed in the bottom, middle and upper tier.

The local deployment of the first stage has multiple advantages. On one hand, locating the first stage in the same (sub)network, close to the systems being monitored, will result in low la-

tency predictions and a privacy-preserving operation since most of the benign samples are filtered out by the anomaly detector and not forwarded to the next stage, thus never leaving the local premises. On the other hand, this allows the deployment of multiple unique anomaly detectors trained on local data to learn a representation of normal behavior specific to the monitored (sub)network. As a result, a higher degree of flexibility is achieved, potentially improving the global classification performance. For example in our simple enterprise network, what is benign in the private cloud might not be in the sales and R&D department or vice-versa. Similarly, certain (sub)networks might require more strict thresholds. A unique anomaly detector, specifically developed and configured for each (sub)network, can then offer a solution.

The threshold τ_B previously introduced in the first stage to differentiate benign from attack, now also influences the overall computational and bandwidth requirements because τ_B controls the number of samples forwarded to the next stage for further analysis, which need to be transmitted over the network. Since the extension layer has the ability to correct benign samples falsely classified as malicious in the first layer, the thresholds τ_B and τ_U allow to balance both the computational requirements and classification performance.

In earlier work [30] we simulated such a three-tier IDS and optimized the capacity and task allocation of the individual stages using simulated annealing and queueing theory. The study concluded that either a single edge or dual edge-cloud deployment are optimal for the lowest delay and stable performance, with the latter being favored by low cloud computing costs.

3.4 Methodology

This section describes the applied methodology to evaluate the novel multi-stage approach for hierarchical intrusion detection using a modern flow-based network dataset and contains all the necessary information to reproduce the reported results in section 3.5. First, subsection 3.4.1 introduces the dataset used for validation together with the applied preprocessing steps. Next, the algorithms used in both the first and second stages are presented before the evaluation strategy is laid out in subsection 3.4.3. At last, the hardware specification on which the experiments are executed is given in subsection 3.4.4.

3.4.1 Data

The CIC-IDS-2017 dataset is a modern flow-based network intrusion dataset developed by Sharafaldin et al. [4]. Before generating CIC-IDS-2017, Sharafaldin et al. already developed NSL-KDD, an altered version of the most popular intrusion detection dataset of the last decades, KDD99 [31]. NSL-KDD resolved many of the found issues present in the original version developed by the Defense Advanced Research Projects Agency (DARPA). In 2016, a list with 11 criteria was published that a proper intrusion detection dataset needs to satisfy [32]. The CIC-IDS-2017 was built from scratch to be the first to successfully fulfill all 11 criteria, justifying the use of this dataset for benchmark purposes. The generation of the dataset was spanned over a period of 5 days using 14 machines.

Table 3.2: Original and down-sampled attack occurrences in CIC-IDS-2017.

Attack Class	Details	Original	Sampled
Benign	ALL	2,071,822	2,071,822
(D)DOS	ALL	3,216,637	1,948
	Hulk	172,726	1,046
	DDOS	128,014	775
	GoldenEye	10,286	63
	DoS slowloris	5,383	33
	Slowhttptest	5,228	31
Port scan	ALL	90,694	1,948
Brute Force	ALL	9,150	1,948
	FTP-Patator	5,931	1,263
	SSH-Patator	3,219	685
Web-Attack	ALL	2,143	1,948
	Brute Force	1,470	1,336
	XSS	652	593
	SQL Injection	21	19
Botnet	ALL	1,948	1,948
Unknown	ALL	47	47
	Infiltration	36	36
	Heartbleed	11	11

The dataset consists of both benign and malicious traffic. The benign traffic is simulated using B-profiles, derived from the benign behavior of a group of 25 humans using statistical techniques and machine learning. On the other hand, the malicious traffic is generated by executing existing attack tools at specific time windows. The benign and malicious traffic is combined into a single dataset and distributed in machine-learning friendly flow-based CSV files and packet captures (PCAP). CICFlowMeter [33] is used to generate the bidirectional flows from the raw PCAP files. A biflow aggregates and computes 80 statistical network features over all the sent packets within a single connection and is identified by the source and destination IP address and port as well as a timestamp.

The original CIC-IDS-2017 dataset is cleaned in three steps by removing columns with redundant information, dropping rows with missing or infinity values, and eventually, the resulting dataset is filtered from duplicates. After cleaning, the data is scaled by normalizing the features using the StandardScaler implementation from scikit-learn [34]. The exact same cleaning and scaling approach is described more in-depth in previous work [35]. Following best practices to avoid any bias, it is important to note that a separate scaler is used to normalize the data for the anomaly detector and multi-class classifier fitted on their respective training set.

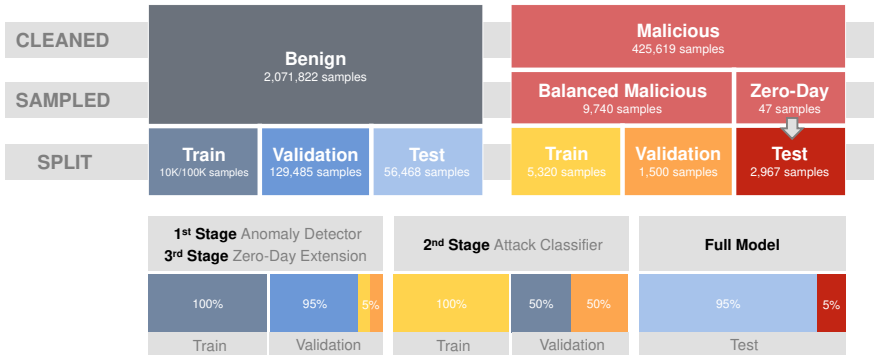


Figure 3.3: Visualization of the used train, validation and test strategy for all the stages and final multi-stage approach.

An overview of the number of occurrences for each of the attack classes and benign traffic of the cleaned data is given in table 3.2. This chapter aggregated the original attack classes into six more high-level attack categories. For example, both FTP- and SSH-patator are categorized as brute force. Because the dataset is highly unbalanced, containing mostly (D)DOS traffic, a sampling technique is used. The minimum number of occurrences of the newly created categories, 1948 in the botnet category, is used to downsample the other categories to the same number so that all attack categories are equally represented in the final dataset. A stratified random sampling technique is chosen because of the abundant number of samples in the majority classes. Since only 11 and 36 samples are present in the original dataset for the attack class infiltration and heartbleed, respectively, these two classes are well suited to be used as a proxy for unknown or zero-day attacks. As a result, the unknown category contains 47 samples, which will only be used to evaluate the proposed approach zero-day detection.

Figure 3.3 visualizes the train, validation, and test split strategy for each of the stages and eventually the complete multi-stage approach. The first two rows indicate the applied downsampling technique. Afterward, the resulting dataset is split into three parts for training, validation, and testing. The down-sampled dataset with malicious samples is split into 70% train and 30% test. The 47 zero-day samples are added to the test set with malicious samples. From the malicious train set, 300 samples for each attack class are sampled, resulting in a validation set containing 1,500 samples. The cleaned benign samples are also split into train, validation, and test sets to match the required distribution between benign and malicious in the final datasets. Eventually, the composition of the train and validation sets for the individual stages, together with the test is presented. The anomaly detector in the first stage is trained using only benign data, respectively, 10,000 and 100,000 samples for the OC-SVM and AE. On the other hand, the validation set is composed of 95% benign and only 5% malicious traffic. An unbalanced validation set is chosen because the anomaly detector needs to filter malicious traffic out of a stream of mainly benign traffic. On the contrary, the classifier in the second stage is trained using a balanced set containing only ma-

licious traffic. This is important to prevent any bias towards a known attack category. Since no samples of unknown attacks exist, they can not be used to validate the classifier in the second stage. Instead, benign traffic is used in such a way that classifiers that output a vector with low probabilities for each of the trained known attacks are rewarded with a higher validation score. The validation set for the second stage consists equally of benign and malicious samples. Finally, the test set consists of 95% benign and 5% malicious traffic not used before to train or validate any of the stages, simulating a realistic network stream of mainly benign traffic. Important to note is that all train, validation, and test sets are sampled in a stratified manner from the previously balanced dataset.

The CSE-CIC-IDS-2018 dataset, the successor of the CIC-IDS-2017, is generated using the same tools but deployed in the cloud rather than on a local university network. From the 2018 dataset 127,844 additional infiltration samples are obtained and cleaned in an identical manner as described before. These samples are kept separately to test the robustness of the zero-day capability of our novel proposed approach.

3.4.2 Algorithms

3.4.2.1 Anomaly Detection

The first stage relies on an anomaly detector to filter the malicious samples from benign traffic. Unsupervised machine learning models are well suited for this task since they model the normal baseline traffic and everything deviating from this is flagged as an anomaly. Since the model only learns the benign baseline from data, it is capable to detect known as well as unknown attacks. This chapter evaluates the use of both an autoencoder (AE) and one-class support vector machine (OC-SVM) as anomaly detector in the first stage.

An AE is a neural network composed of an encoder and decoder. The encoder projects the input vector onto a lower-dimensional or latent space. The decoder reconstructs the input vector as close as possible from this latent space. The reconstruction error, calculated as the sum of squared errors (SSE), is used as an anomaly score. In case the encoder and decoder are built from more than one layer, it is called a deep autoencoder (DAE). The Keras framework [36] on top of Tensorflow [37] is used for the experimental implementation. The following hyper-parameters are optimized during training: number of hidden layers, number of neurons per layer, activation function, and regularisation terms.

An OC-SVM is a special SVM that instead of separating two classes using a hyper-plane, encloses a single class as tight as possible using a hyper-sphere. The use of the radial basis function as (rbf) kernel function allows a complex, non-linear boundary for this hyper-sphere. The scikit-learn library is used for the implementation. The input features are first reduced using a PCA transformation. During training, the number of components in the PCA transformation, kernel coefficient, and regularization parameter is optimized.

3.4.2.2 Multi-Class Detection

The second stage classifies forwarded suspicious samples from the first stage using a multi-class classifier to a known attack class. Supervised machine learning models are well suited to learn this representation from labeled data. This chapter evaluates two classifiers: a random forest (RF) and a neural network (NN).

An RF is an ensemble of decision trees where each tree is built using the whole or sub-sample of the dataset. The final prediction is defined as the average output of all the trees, or in the case of classification as the most predicted class. The hyper-parameters considered during optimization are the number of trees, sub-sample percentage, and the number of features used for each split in a single tree. The scikit-learn implementation is used in this chapter.

A NN is composed of an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple nodes which are connected with a certain weight to the nodes of the next layer. Data is sent from one layer to another if the value is above a particular threshold, defined by the activation function. The structure and name are inspired by the human brain and form the basis of deep learning algorithms. The number of layers, the respective number of nodes per layer, and a regularization term are optimized during hyperparameter tuning. For the implementation, the Keras framework on top of TensorFlow is used.

3.4.3 Evaluation Strategy

The novel multi-stage approach for hierarchical intrusion detection proposed in this chapter is composed of three stages. Before the performance of the complete IDS can be analyzed and compared with both a single model baseline implementation and existing state-of-the-art multi-stage approaches, the individual models in the stages and their corresponding thresholds need to be defined. Both the anomaly detector in the first stage and multi-class classifier in the second stage are trained individually using the training and validation set introduced in subsection 3.4.1. The optimal hyper-parameters for each model, as described in subsection 3.4.2, are obtained by performing hyper-parameter optimization with as validation metrics the AUROC and weighted F1 score for the first and second stage, respectively. Optuna [38], an open-source optimization framework, is used for the implementation with a Tree-structured Parzen Estimator (TPE) as sampling algorithm. The possible thresholds corresponding to a model are also computed on the same validation set. For each model in the first stage, ten possible values for τ_B are proposed, selected by the anomaly score corresponding to the maximum F1 to F9 score. The threshold used in the extension layer, τ_U is also computed on the same anomaly score but is selected using quantiles on the anomaly score of the benign flows in the validation set. Four possible candidates are examined using the 0.995, 0.99, 0.975, and 0.95 quantiles. Finally, only a single candidate for the threshold τ_M in the second stage is proposed, computed by the cut-off value on the confidence of the prediction achieving the maximum weighted f1 score.

The ten instances for each algorithm with the highest validation score and their corresponding

Table 3.3: Best results first stage: anomaly detection

Algorithm	AUROC	AUPR	F1	F2	F3	F4	F5	F6	F7	F8	F9	Training (s)	Inference (s)
AE	0.9117	0.3265	0.3984	0.5650	0.6666	0.7351	0.7863	0.8370	0.8715	0.8958	0.9135	39.671 ± 2.046	1.785 ± 0.017
OC-SVM	0.8947	0.3256	0.3893	0.5016	0.6426	0.7267	0.7823	0.8276	0.8584	0.8804	0.8976	0.771 ± 0.021	8.117 ± 0.277

thresholds are further analyzed for their use in the multi-stage approach. The final performance on the test set is evaluated for each permutation of the two models, anomaly detection and attack classification, and the three values for the applied thresholds. These permutations are then ranked and analyzed based on several criteria such as accuracy, bandwidth requirements, and ability to detect unknown attacks. Eventually, the performance of the complete multi-stage approach is compared with a baseline RF implementation and previous state-of-the-art multi-stage approach as described by Bovenzi et al. [17]. The baseline RF model is trained on a mixed dataset of benign and malicious samples, using an open-set approach to add the ability to detect zero-day attacks. The threshold used by the baseline model to predict a sample as a zero-day is computed analogously to the threshold τ_M in the second stage.

Eventually, the robustness of the zero-day detection is evaluated by classifying the additional obtained infiltration samples from the CSE-CIC-IDS-2018 dataset. The fully optimized model, trained on data from the CIC-IDS-2017 dataset, is used to predict all the additional infiltration samples. A robust model is expected to transfer the zero-day detection capabilities with only a small drop in performance.

3.4.4 Hardware Setup

The experimental results in this chapter are obtained on GPULab, a distributed job-based platform hosted and maintained by the university department IDLab in collaboration with IMEC. All the experiments executed are submitted as a job that runs in an isolated container built upon a basic Python 3.7 Docker image with all the needed libraries installed. Each job was assigned 4 CPUs, Intel(R) Xeon(R) Silver 4108 CPU @ 1.80GHz and 16GB of RAM.

3.5 Results

3.5.1 Anomaly Detection

An overview of the classification performance using the recommended metrics for binary detection, as well as training and inference timings, are given in table 3.3. The table contains two records with the test results of respectively the AE and OC-SVM model achieving the highest validation AUROC score for the first stage. An AE with a single hidden layer consisting of 42 nodes and a regularisation term of $2.45e-5$ achieved the highest AUROC of 0.9062 during hyper-parameter optimization on the validation set of stage 1 after 7 epochs. On the test set, this resulted in an AUROC of 0.9117. The fully optimized OC-SVM scored a slightly lower AUROC on both the validation and test set with, respectively, 0.8931 and 0.8947, but it eventually performs better in combination

Table 3.4: Best results second stage: attack classification

Algorithm	F1 weighted	F1 macro	Accuracy	Balanced Accuracy	Training (s)	Inference (s)
RF	0.9870	0.9094	0.9846	0.9654	1.249 ± 0.034	0.610 ± 0.005
NN	0.9525	0.7096	0.9403	0.9113	4.885 ± 0.189	1.555 ± 0.190

with the next stages in the multi-stage approach. The best hyper-parameters for the OC-SVM in the first stage are a kernel coefficient of 0.0633, a regularisation term of $2.317e-4$, and PCA transformation with 56 components. Both models are able to process the test set just shy of 60 thousand samples in a matter of seconds, in other words, a single sample takes less than a millisecond. Even with 2 orders of magnitude difference in training time between the AE and OC-SVM, both models are suitable to be practically used because training is not required to happen in real-time on a resource-constrained device. The main reason for the gap in training time is the number of used training samples, respectively 10 thousand and 100 thousand for the OC-SVM and AE. For comparison purposes, the OC-SVM is also retrained on the identical dataset as the AE resulting in a similar classification performance but increased training time of $273.0 \pm 0.2s$. This confirms our previous work [39] showing that models quickly converge even with limited available training samples on academic intrusion datasets. The non-linear decrease in training time combined with no classification degradation when reducing the number of training samples, allowed us to perform the hyper-parameter optimization for the OC-SVM without limiting the search space using a smaller training set.

3.5.2 Multi-Class Classification

Table 3.4 summarizes the results in the second stage for the RF and NN model achieving the highest classification performance on the test set after hyper-parameter optimization. The RF outperforms the NN across all metrics for the task at hand. The best performing random forest consists of 97 fully grown trees with only 90% of the samples used to train each individual tree and with 12 features considered at each split. This RF achieved a weighted F1 score of 0.9710 and 0.9870 on the validation and test set, respectively. The NN consisting of a single layer with 41 nodes, 0.0379 as regularisation term and 17 epochs of training, achieves a weighted F1 of 0.9203 on the validation set of stage 2 and 0.9525 on the test set. The macro F1 score dropped from 0.9198 to 0.7096 for the validation and test set, respectively. This is because the test set in contrary to the validation set consists for 50% of benign data. Therefore, even a small percentage of benign traffic classified as one of the known attacks can heavily affect the F1 score associated with the known attacks and thus also the macro F1 score. Even with the NN model classifying less than half of the number of samples in the same time period as the RF model, both models are well suited to be implemented in a high-speed network environment, with a single sample taking less than a fraction of a millisecond of processing time. The training time for the RF and NN currently only takes a few seconds, allowing for more training samples if available.

Table 3.5: Results full multi-stage approach

	τ_B	τ_M	τ_U	F1 weighted	F1 macro	Accuracy	Bal. Accuracy	Bandwidth reduction	Zero-day recall	Inference (s)
Max F-score	F5-8	F1	0.995	**0.9897**	**0.8276**	**0.9877**	0.8954	68.75%	0.5957	7.808 ± 0.009
Max bACC	F9	F1	0.95	0.9580	0.7496	0.9341	**0.9608**	57.91%	**0.9574**	8.043 ± 0.065
Balanced	F5-8	F1	0.99	0.9875	0.8231	0.9834	0.9342	68.75%	0.8723	7.882 ± 0.054
RF Baseline	-	-	-	0.9849	0.7981	0.9832	0.8877	-	0.8936	1.525 ± 0.013
Bovenzi et al. [17]	F3	F1	-	0.9383	0.7549	0.8957	0.8550	**86.22%**	**0.9574**	6.969 ± 0.399

3.5.3 Full Model Performance

The performance of the complete multi-stage model can be assessed based on several criteria. First, the overall classification performance of the three stages combined. Next, the ability to not only reduce the computation requirements but also decrease the bandwidth requirements in case of a hierarchical deployment, and lastly, the capability to detect unknown attacks. Unfortunately, not a single permutation of models and thresholds scores the highest on all of the criteria but a trade-off needs to be made using the defined thresholds τ_B, τ_M, and τ_U. The best performing permutations are consistently composed of the best performing OC-SVM and RF models from the first and second stage, respectively.

Table 3.5 gives an overview of the results for three interesting permutations of the complete multi-stage model, the single model baseline and the current state-of-the-art multi-stage approach for hierarchical intrusion detection. The first row presents the results of the permutation achieving the highest classification performance in terms of accuracy and both weighted and macro F1 scores. This permutation used the anomaly score corresponding to the maximum F5 to F8 score, an identical anomaly score, in the first stage as a value for τ_B. The anomaly score corresponding to the 0.995 quantiles is used as the value for τ_U in the extension stage. Using these thresholds, a maximum weighted F1 score is achieved of 0.9897 and an accuracy of 0.9877 on the test set, but only 59.57% of the unseen attacks are successfully classified as a zero-day attack when using the complete multi-stage approach. In case it is deployed in a hierarchical manner a bandwidth reduction of almost 69% is achieved between the local anomaly detector and centralized attack classifier, in comparison with an IDS forwarding all its traffic to a centralized system.

The second row in the table presents the results of the permutation scoring the highest on the balanced accuracy with a value of 0.9608. This configuration also achieves an extremely high recall of zero-day attacks, with 45 out of 47 samples, while preserving high overall classification scores. Contrary to the first row, a higher value is used for τ_B, corresponding to the anomaly score with the maximum F9 score in the first stage, leading to more traffic forwarded to the next stage. On the other hand, the value for τ_U is decreased to the anomaly score corresponding to the 0.95 quantiles. This results in a more swift classification as zero-day and thus higher recall of zero-day attacks.

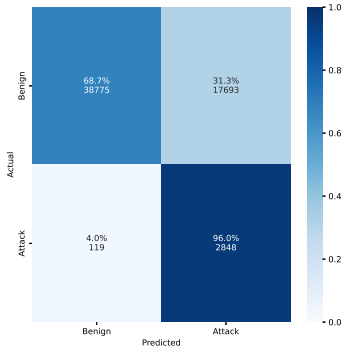
The third row presents the results for a permutation balancing between the high classification performance of the first row and the high zero-day recall of the second row. For both the threshold τ_B and τ_U, a value is chosen in between the thresholds selected in the first and second row. This permutation serves as a middle ground in the challenge to balance the multiple trade-offs.

The performance results of the single model baseline can be found in the fourth row. There are no values present for the thresholds as well as for the bandwidth reduction since these are not relevant. The hyper-parameters obtained through optimization for the baseline RF are 57 fully grown trees with only 87% of the samples used to train each individual tree and with 30 features considered at each split. Additionally, the threshold used in the open-set classification is set to 0.93, yielding a zero-day recall of 89%.

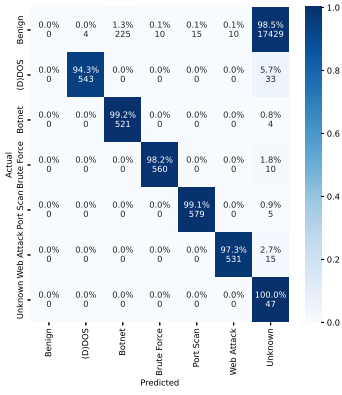
The last row contains the performance metrics for the state-of-the-art multi-stage approach. There is no value for the threshold τ_U since their approach lacks an extension stage. Overall the approach by Bovenzi et al., which was not previously validated on CIC-IDS-2017, performs fairly well. Especially the bandwidth reduction and zero-day recall are high, while the other classification metrics are consistently out-preformed by our novel approach.

The last column in Table 3.5 lists the execution time to classify all the samples in the test. The single model RF baseline yields the lowest execution time, while the novel approach has the longest execution time.

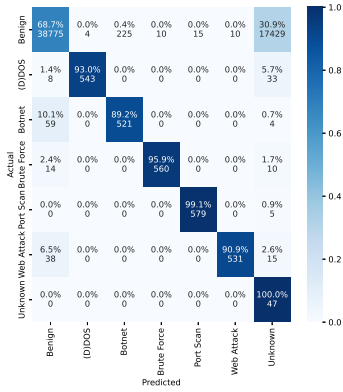
Figure 3.4 presents six confusion matrices for each of the individual stages, intermediate result and final result of the complete multi-stage approach on the test set using the thresholds of the permutation balancing the multiple trade-offs together with the final confusion matrix of the existing multi-stage approach by Bovenzi et al. [17] applied to the CIC-IDS-2017 dataset. The confusion matrix in figure 3.4a visualizes the true negatives, false positives, false negatives, and true positives in, respectively, the upper left, upper right, lower left, and lower right quadrant for the anomaly detector in the first stage. Only the positive samples are forwarded to the next stage, therefore already 119 samples or 4% of the fraud samples are misclassified which are unable to be corrected by the succeeding stages. Simultaneously, nearly 69% of the benign samples are correctly classified and prevented from propagating further in the system. Figure 3.4b contains the confusion matrix for all the suspicious classified samples by the anomaly detector in the first stage, corresponding to the positives from the first confusion matrix. Most of the known, as well as unknown attacks, are correctly classified, while most of the benign samples are properly predicted as unknown and thus forwarded to the final extension stage. Before the confusion matrix of the extension layer is presented, the matrix for the intermediate result of the first and second stage combined is given in figure 3.4c. Except for the high number of benign samples classified as unknown, the results already look good. To correct the mistakes in the first stage to send these benign samples to the second stage, the extension stage was introduced. Figure 3.4d shows the confusion matrix for the extension stage. Clearly, most of the benign traffic is being successfully corrected and eventually classified as benign. The confusion matrix in figure 3.4e eventually presents the results of the predictions of all three stages combined for our novel proposed approach. The last confusion matrix in figure 3.4f is of the existing multi-stage approach applied to the test dataset. The results are similar to the confusion matrix in figure 3.4c albeit the number of benign samples classified as unknown is lower but still significantly high. Furthermore, most attack classes are reliably detected except for botnet followed by brute-force attacks.



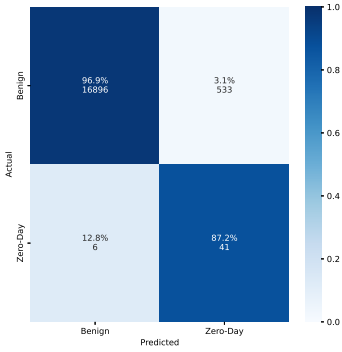
(a) Stage 1: Anomaly Detection



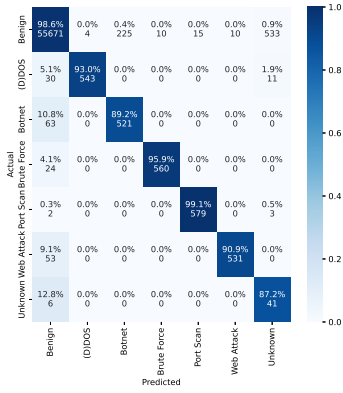
(b) Stage 2: Multi-class Classification



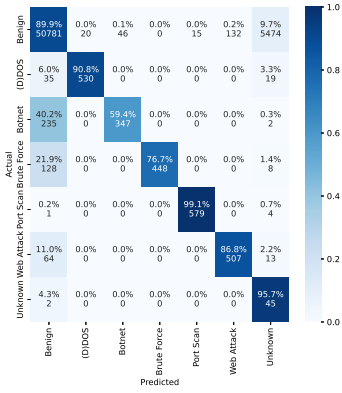
(c) Stage 1 and 2 combined



(d) Extension Stage



(e) Full Multi-Stage Approach



(f) SotA Bovenzi et al. [17]

Figure 3.4: Confusion Matrices.

3.5.4 Robustness Zero-day detection

The robustness of the zero-day detection capabilities is evaluated on 127,844 additional samples from the CSE-CIC-IDS-2018 dataset. Our proposed approach correctly classifies 100,199 samples as unknown attack, equal to a recall of 78.38%. The baseline RF only succeeded in classifying 76 samples correctly as unknown, which is less than 0.01% recall. The approach by Bovenzi et al. classifies 111,215 samples correctly, resulting in a recall of 86.99%.

3.6 Discussion

3.6.1 Improved Classification over Baseline and State-of-the-art

Section 3.5 and more specifically table 3.5 present the results for three permutations of the proposed novel multi-stage approach, the open-set RF baseline, and the previously proposed multi-stage approach by Bovenzi et al.

When we compare our novel proposed multi-stage approach against the baseline RF and state-of-the-art approach, the classification metrics are consistently being out-preformed. Only the state-of-the-art approach can present the same zero-day recall and even higher bandwidth reduction. This can be explained by the extension stage in our novel approach since it creates the opportunity for the model to correct mistakes of the first stage. As a result, the threshold τ_B in the first stage can be configured less strictly which has a negative effect on the bandwidth reduction but more significant improvement of classification performance. When bandwidth reduction is a priority, our novel approach also allows tuning τ_B to obtain at least an equal reduction.

Recent literature questions the marginal improvements made in studies to not improve performance in the real world because of the lack of generalization power [35][40] [41]. As a result, the main takeaway is that the classification performance at least matches the state-of-the-art while providing additional features such as extra flexibility, zero-day detection, and bandwidth reduction.

3.6.2 Advantage of Extension Stage

The extension stage serves multiple purposes. First of all, it improves the classification performance by predicting samples classified as unknown by the second stage as benign. In figure 3.4c, more than 30% of all the benign samples are classified as unknown. Without an additional stage, the model would fail to correct them. Next, the extension stage permits the first stage to make mistakes. Therefore, the first stage can mark more samples as suspicious, knowing that forwarded benign samples can be corrected by the following stages. The improved classification performance comes at a tradeoff with the bandwidth requirement since forwarded samples need to be transmitted over the network in a hierarchical deployment.

3.6.3 Hierarchical Deployment

The novel multi-stage approach proposed in this chapter is designed for a scalable hierarchical architecture. Each of the individual stages can be deployed on its own location in the network, for example in the edge, fog, or cloud. Ideally, the anomaly detector is situated near the network being monitored. This way privacy is preserved during operation because mostly malicious data is forwarded from the local premises.

Furthermore, the bandwidth and computational reduction are achieved by applying a threshold on the anomaly score predicted by the first stage such that only a fraction of all samples are forwarded to the next stage, which requires transmission over a network in case of a n-tier deployment. Since only a small share of all data is being forwarded by the first stage, most samples are only processed by the lightweight filter skipping the potentially more computational expensive multi-class classifier in the second stage. Contrarily, recent studies such as Khan et al. [12] which employs a similar architecture consisting of binary detection followed by attack classification is less suited for a hierarchical deployment because the anomaly score is used as an extra feature for improved classification in the second stage and not as a lightweight filter.

The execution times in Table 3.5 show a small increase in overhead between the approach by Bovenzi et al. and our novel approach due to the addition of an extra stage. Evaluating the execution times between the different configurations of our novel approach only show small differences. But keep in mind that these results are obtained on a single machine, reiterating the same experiment with a n-tier deployment will also include the effect of the bandwidth reduction as shown in our previous work [30].

3.6.4 Thresholds τ_B, τ_M and τ_U

Figure 3.5 plots the distribution of both the benign and malicious samples in function of the anomaly score for the OC-SVM with the highest AUROC and eventually used as the anomaly detector in the complete multi-stage model. The values for τ_B and τ_U are marked using a vertical dashed line. All samples left from the first dashed line, thus having a lower anomaly score than the threshold τ_B are classified as benign, while all samples on the right side with an anomaly score higher than τ_B are forwarded to the next stage. Carefully selecting this threshold is crucial to obtaining proper results. If the threshold is set too low, many samples are forwarded to the next stage, consuming both computational resources and bandwidth with an additional risk of classifying benign samples wrongly as a known attack. When the threshold is set too high, the first stage classifies actual intrusion as benign. As a result, the threshold τ_B balances the computational and bandwidth requirements as well as final classification performance. The second dashed line visualizes the applied cut-off value to classify a sample as a zero-day attack. If the anomaly score is higher than τ_U and thus on the right side of the line, the sample is marked as a zero-day attack. However, if the anomaly score is between τ_B and τ_U the sample will be classified as benign in the extension stage. Setting the threshold τ_U too low will wrongly classify benign samples as zero-day attacks while setting it too high prevents zero-day attacks from being

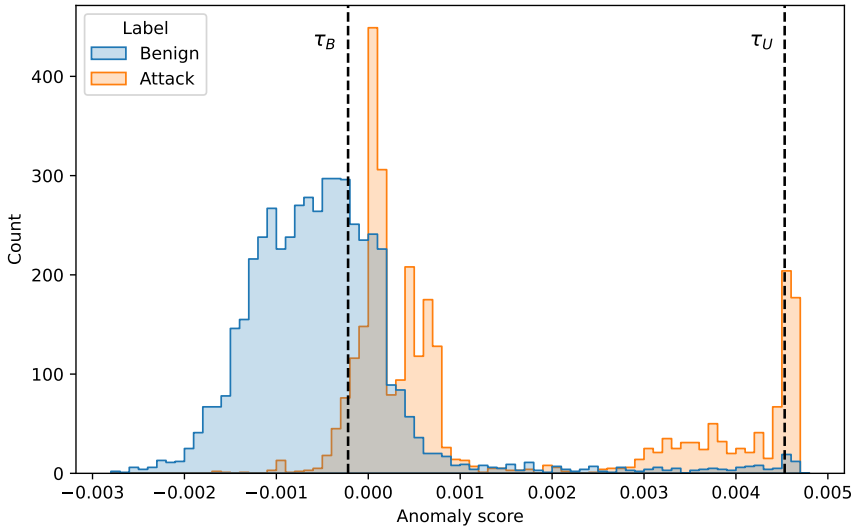


Figure 3.5: Histogram of the anomaly score outputted by the first stage for both benign and fraud traffic with the possible thresholds τ_B and τ_U visualized.

discovered. Accordingly, defining τ_U is essential to successfully detect zero-day attacks.

The threshold τ_M controls the cut-off on the confidence of the prediction by the attack classifier in the second stage. In case τ_M would be equal to zero, none of the samples would be classified as unknown and thus not forwarded to the extension stage. Rather, if τ_M is set to one, only samples where the attack classifier is absolutely sure will be classified as one of the known attack classes. Consequently, setting the threshold too low allows loose classification to one of the known classes which will wrongly predict benign samples to one of the known attacks. Yet, setting the threshold too high will prevent actual attacks to be correctly classified.

3.6.5 Manual Selection of Thresholds

In this chapter, an optimization technique is used to select the optimal values for the thresholds τ_B, τ_M, and τ_U with mixed validation set consisting of both benign and malicious samples. This is a valid approach to obtaining good results on benchmark datasets but not always feasible in a real-world setting. The collection of labeled malicious samples is complex and time-demanding. The inclusion of benign samples in the validation set of the multi-class classifier breaks the privacy preserved training and becomes increasingly complex when samples of multiple (sub)networks need to be included.

Following rules of thumb can help with the initial configuration of the thresholds. Afterward, the performance tradeoff can be iteratively adjusted until a favorable configuration is obtained.

3.6.5.1 τ_B

This chapter optimized the value for the threshold yielding the maximal f-score. Interesting is that this f-score corresponds to a certain false positive ratio (fpr). When we look at the top hundred permutations with the highest global classification performance from subsection 3.4.3, then we find that the average and median fpr is equal to 0.2259 and 0.2107 ± 0.0661, respectively. As a result, a value for the threshold τ_B can be selected using only benign samples.

3.6.5.2 τ_U

The same approach as with τ_B can be taken for τ_U with as difference that the selected fpr is an upper limit for the fpr of the whole multi-stage approach. Both the median and average fpr used in the top hundred best-performing permutations are 0.995 ± 0.001.

3.6.5.3 τ_M

The selection of τ_M can also intuitively be done but often vary depending on the used multi-class classifier. Most classifiers produce a probability value associated with each of the classes, the threshold then determines the minimum required confidence to classify a sample as a known attack. The mean and average τ_M for the best performing classifiers after optimization are respectively, 0.95 and 0.93 ± 0.03.

3.6.6 Adaptability

All the thresholds τ_B, τ_M, and τ_U allow live adjustment to dynamically balance the trade-offs in the individual stages, and as a result, adapt the final classification performance without the need for retraining the machine-learning models.

Our novel approach has not been designed specifically to tackle concept-drift challenges in mind, but rather to enable these performance trade-offs in real-time. Although, we expect this approach to withstand concept-drift to a certain degree, for example, a radical shift of the underlying benign traffic will at least require retraining of the anomaly detector in the first stage.

3.6.7 Robustness of Zero-day Detection

A limitation of our chapter is that only 47 samples from the CIC-IDS-2017 dataset are used as unknown or zero-day attack. Therefore, the robustness of the zero-day detection capabilities is evaluated on 127,844 additional infiltration samples extracted from the CSE-CIC-IDS-2018 dataset. The results show that the baseline RF fails completely to maintain a similar zero-day detection capability with a recall that dropped from almost 90% to less than 0.01%. On the contrary, the approach by Bovenzi et al. and our novel approach are able to maintain the majority of their zero-day detection capabilities with a drop in recall of less than 10%. As a result, a multi-stage approach is not only able to achieve a higher zero-day detection rate but is also more robust.

3.7 Future Work

The experimental implementation of the multi-stage approach for hierarchical intrusion detection only relies on network features as input while the general proposed architecture is also applicable to both host-based and hybrid IDS. Future work could extend this chapter by evaluating the proposed approach on multiple input sources. For instance, ensemble techniques can then be applied to aggregate the output of a machine-learning model on each of the input vectors.

Recently published work [42] [43] together with our previous work demonstrates that most, if not all, currently proposed models trained on network IDS datasets lack generalization strength. Future work should evaluate if a multi-stage model is more resilient against this classification performance degradation. Moreover, our proposed approach is capable of having a separate anomaly detector specifically trained for each subnetwork in a hierarchical deployment. This enables the simulation of a more realistic diverse network, for example consisting of both IoT and general purpose devices, using a multi-data set evaluation.

Parallel to this chapter, we have evaluated 10 possible task allocations, which assign to each task a capacity in a three-tier network consisting of the edge, fog, and cloud. A simulation is performed with queueing theory, resulting in multiple optimal configurations depending on specific requirements [44]. A follow-up on this work will try to confirm the theoretical results using an experimental setup of a multi-stage IDS on a multi-tier architecture.

3.8 Conclusion

In this chapter, a novel multi-stage approach for hierarchical intrusion detection is proposed. First, the general architecture is introduced and the design choices made, are justified. The strengths of the new approach are the high adaptability without the necessity to retrain any of the classifiers, the empowerment of an n-tier deployment to reduce the bandwidth and computational requirements, and the capability to detect zero-day attacks. Additionally, in the case of a hierarchical deployment, privacy is preserved during both training and operational service.

After the novel approach is introduced an experimental implementation is evaluated using two modern network intrusion datasets, CIC-IDS-2017 and CSE-CIC-IDS-2018. An AE and OC-SVM are evaluated for anomaly detection, while an RF and NN are evaluated for the attack classification. The experimental results show that the classification performance at least matches or even outperforms both the baseline single model and existing multi-stage approaches for most of the evaluated metrics. Besides the improved classification performance, the novel multi-stage approach proved its robust capability to detect unseen, zero-day attacks and the ability to reduce the computational and bandwidth requirements. The implementation of the novel multi-stage approach that balances the trade-off between a high zero-day recall, bandwidth reduction, and classification performance achieves a weighted F1 and balanced accuracy score of 0.9875 and 0.9342, respectively, as opposed to 0.9383 and 0.8550 for the state-of-the-art approach for multi-stage

intrusion detection. In particular, 41 out of 47 or 87% zero-day samples were correctly classified from CIC-IDS-2017. Moreover the robustness of the zero-day detection is validated by correctly classifying 100,199 out of 127,844 or 78% zero-day samples from CSE-CIC-IDS-2018. The bandwidth was reduced by almost 69% between the local anomaly detector and centralized attack classifier in case of a hierarchical deployment in comparison with an IDS forwarding all its traffic to a centralized system. The specific results depend on the selected thresholds and are highly adaptable to obtain the desired trade-offs.

Bibliography

[1] A. Jurcut, T. Niculcea, P. Ranaweera, and N.-A. Le-Khac, "Security considerations for internet of things: A survey," *SN Computer Science*, vol. 1, pp. 1–19, 2020.

[2] T. Alam, "A reliable communication framework and its use in internet of things (iot)," vol. 3, 05 2018.

[3] E. Viegas, A. Santin, A. Bessani, and N. Neves, "Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks," *Future Generation Computer Systems*, vol. 93, pp. 473–485, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18307635

[4] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," 01 2018, pp. 108–116.

[5] Y. Zhang, L. Wang, W. Sun, R. C. Green II, and M. Alam, "Distributed intrusion detection system in a multi-layer network architecture of smart grids," vol. 2, p. 796–808, Dec 2011.

[6] A. Ali and M. M. Yousaf, "Novel three-tier intrusion detection and prevention system in software defined network," *IEEE Access*, vol. 8, p. 109662–109676, 2020.

[7] L. Li, Y. Yu, S. Bai, Y. Hou, and X. Chen, "An effective two-step intrusion detection approach based on binary classification and k -nn," *IEEE Access*, vol. 6, p. 12060–12073, 2018.

[8] H. H. Pajouh, G. Dastghaibyfard, and S. Hashemi, "Two-tier network anomaly detection model: a machine learning approach," *Journal of Intelligent Information Systems*, vol. 48, no. 1, p. 61–74, Feb 2017.

[9] H. H. Pajouh, R. Javidan, R. Khayami, A. Dehghantanha, and K.-K. R. Choo, "A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, p. 314–323, Apr 2019.

[10] W. L. Al-Yaseen, Z. A. Othman, and M. Z. A. Nazri, "Multi-level hybrid support vector machine and extreme learning machine based on modified k-means for intrusion detection system," *Expert Systems with Applications*, vol. 67, p. 296–303, Jan 2017.

[11] S.-Y. Ji, B.-K. Jeong, S. Choi, and D. H. Jeong, "A multi-level intrusion detection method for abnormal network behaviors," *Journal of Network and Computer Applications*, vol. 62, p. 9–17, Feb 2016.

[12] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "A novel two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, p. 30373–30385, 2019.

[13] L. Yang, A. Moubayed, and A. Shami, "Mth-ids: A multitiered hybrid intrusion detection system for internet of vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 616–632, 2022.

[14] Divyatmika and M. Sreekesh, "A two-tier network based intrusion detection system architecture using machine learning approach," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, Mar 2016, p. 42–47.

[15] M. F. Umer, M. Sher, and Y. Bi, "A two-stage flow-based intrusion detection model for next-generation networks," *PLOS ONE*, vol. 13, no. 1, p. e0180945, Jan 2018.

[16] Y. Abuadlla, G. Kvascev, S. Gajin, and Z. Jovanovic, "Flow-based anomaly intrusion detection system using two neural network stages," *Comput. Sci. Inf. Syst.*, 2014.

[17] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "A hierarchical hybrid intrusion detection approach in iot scenarios," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Dec 2020, p. 1–7.

[18] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Multi-stage optimized machine learning framework for network intrusion detection," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, p. 1803–1816, Jun 2021.

[19] R. Coulter, Q.-L. Han, L. Pan, J. Zhang, and Y. Xiang, "Data-driven cyber security in perspective—intelligent traffic analysis," *IEEE Transactions on Cybernetics*, vol. 50, no. 7, pp. 3081–3093, 2020.

[20] Y. Miao, C. Chen, L. Pan, Q.-L. Han, J. Zhang, and Y. Xiang, "Machine learning–based cyber attacks targeting on controlled information: A survey," *ACM Comput. Surv.*, vol. 54, no. 7, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3465171

[21] I. H. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," *Journal of Big Data*, vol. 7, no. 1, p. 41, Jul 2020.

[22] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *CoRR*, vol. abs/1802.09089, 2018. [Online]. Available: http://arxiv.org/abs/1802.09089

[23] S. Zavrak and M. İskefiyeli, "Anomaly-based intrusion detection from network flow features using variational autoencoder," *IEEE Access*, vol. 8, pp. 108 346–108 358, 2020.

[24] Q. T. Nguyen, K. Phuc Tran, P. Castagliola, T. Thu Huong, M. K. Nguyen, and S. Lardjane, "Nested one-class support vector machines for network intrusion detection," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, 2018, pp. 7–12.

[25] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33 789–33 795, 2018.

[26] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Comput. Surv.*, vol. 51, no. 3, May 2018. [Online]. Available: https://doi.org/10.1145/3178582

[27] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1222–1228.

[28] P. Wu and H. Guo, "Lunet: A deep neural network for network intrusion detection," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019, pp. 617–624.

[29] M. Al-Zewairi, S. Almajali, and A. Awajan, "Experimental evaluation of a multi-layer feed-forward artificial neural network classifier for network intrusion detection system," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, 2017, pp. 167–172.

[30] Y.-C. Lai, D. Sudyana, Y.-D. Lin, M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. D. Turck, "Task assignment and capacity allocation for ml-based intrusion detection as a service in a multi-tier architecture," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[31] T. U. of California, "Kdd cup 1999 data," no. 28, Oct. 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[32] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An evaluation framework for intrusion detection dataset," in *2016 International Conference on Information Science and Security (ICISS)*, Dec 2016, p. 1–6.

[33] T. U. of California, "Cicflowmeter." [Online]. Available: https://github.com/ahlashkari/CICFlowMeter

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Towards model generalization for intrusion detection: Unsupervised machine learning techniques," *Journal of Network and Systems Management*, vol. 30, no. 1, p. 12, Oct 2021.

[36] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[38] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[39] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Classification hardness for supervised learners on 20 years of intrusion detection data," *IEEE Access*, vol. 7, pp. 167 455–167 469, 2019.

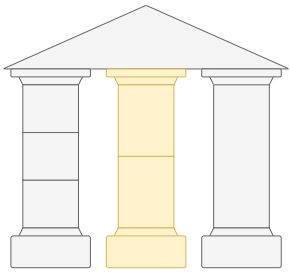
[40] D'hooge, Laurens and Wauters, Tim and Volckaert, Bruno and De Turck, Filip, "Inter-dataset generalization strength of supervised machine learning methods for intrusion detection," *JOURNAL OF INFORMATION SECURITY AND APPLICATIONS*, vol. 54, p. 13, 2020. [Online]. Available: http://dx.doi.org/10.1016/j.jisa.2020.102564

[41] C. F. T. Pontes, M. M. C. de Souza, J. J. C. Gondim, M. Bishop, and M. A. Marotta, "A new method for flow-based network intrusion detection using the inverse potts model," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1125–1136, 2021.

[42] S. Layeghy and M. Portmann, *On Generalisability of Machine Learning-based Network Intrusion Detection Systems*, May 2022.

[43] M. Catillo, A. Del Vecchio, A. Pecchia, and U. Villano, "Transferability of machine learning models learned from public intrusion detection datasets: the cicids2017 case study," *Software Quality Journal*, Mar 2022. [Online]. Available: https://doi.org/10.1007/s11219-022-09587-0

[44] Y.-C. Lai, D. Sudyana, Y.-D. Lin, M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Machine learning based intrusion detection as a service: Task assignment and capacity allocation in a multi-tier architecture," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, ser. UCC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3492323.3495613



4

ChronosGuard: A Hierarchical Machine Learning Intrusion Detection System for Modern Clouds

This chapter contributes to the second thematic pillar, Research to Practice, by addressing the practical deployment of ML-NIDS. Specifically, it tackles Research Questions 3 and 4, which research how ML-NIDS can be deployed for large-scale cloud-native environments and explores potential practical barriers that may prevent ML-NIDS adoption in real-world environments. The work presented in this chapter builds on our earlier joint research with NYCU into task assignment and capacity allocation for hierarchical deployment of multi-stage ML-NIDS using optimization algorithms and queueing theory [1, 2], as well as the development of the multi-stage ML-NIDS introduced in Chapter 3. Expanding on that architecture, we present ChronosGuard*, a deployment-ready, hierarchical ML-NIDS optimized for containerized infrastructures.*

Through thousands of automated experiments conducted on a Kubernetes cluster, this chapter systematically evaluates how different factors, such as deployment strategies, network topologies, and orchestration algorithms, affect system performance and resource efficiency. These findings provide unique practical insights for deploying ML-NIDS from research prototypes to production-ready solutions in modern cloud-native environments.

⋆ ⋆ ⋆

M. Verkerken, J. Santos, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck

Published in the proceedings of the 20th IEEE International Conference on Network and Service Management (CNSM), December 2024

Abstract Traditional Intrusion Detection Systems (IDSs) have been a cornerstone of network security for many years. Nevertheless, with the advent of containerized applications in the last few years, there is a growing need to understand how intrusion detection can adapt to these dynamic environments. This chapter presents ChronosGuard, a hierarchical machine learning (ML) IDS designed for containerized environments. ChronosGuard's adaptable architecture consists of multiple components, each optimized for deployment in varying configurations ranging from monolithic to micro-service architectures. The performance impact of various factors such as network topology, workload orchestration, and deployment strategies has been assessed through extensive experiments concerning the scalability and resource utilization of ChronosGuard. Results show the effective prioritization of benign traffic of up to 85% compared to malicious traffic, the negligible impact of small network delays on performance metrics, and up to 10% decrease in response times with network-aware orchestration for complex deployment configurations. This chapter introduces a robust, containerized IDS that can be easily adapted to meet various operational needs, ranging from a full privacy-preserving local deployment to a scalable cloud deployment but also provides foundational insights for future research into optimizing containerized security solutions.

4.1 Introduction

In our digitized world, the extensive use of interconnected systems and digital technologies amplifies the importance of cybersecurity for individuals, institutions, and organizations [3, 4]. Critical infrastructure systems such as power grids, electronic healthcare, and financial systems are often distributed, and their potential breaches pose severe global risks. Operational, financial, and personal losses can be significant in such events. To counteract such threats, it is essential to implement robust cybersecurity measures. Various tools and techniques [5, 6, 7] exist for preventing, detecting, and mitigating cybersecurity risks. These complementary tools can be collectively deployed to satisfy the required security standards. Among these, Intrusion Detection Systems (IDSs) [8, 9] play a crucial role.

An IDS is designed to monitor a network or system and to identify malicious activities or potential intrusions. Typically, it serves as a second line of defense, supplementing a firewall. A firewall, which is the primary defensive layer, enhances security by managing incoming and outgoing traffic and enforcing network restrictions. However, firewalls also have inherent limitations [10, 11]. They only secure the perimeter of the system, offering no protection against internal threats. Historically, the focus was on preventing unauthorized external access while safeguarding authorized internal users, a concept contradicting the zero-trust principle, which assumes no secure zones. In

addition, their capacity to examine network traffic is limited, and the adoption of encrypted traffic further undermines their deep packet inspection capabilities. This is where an IDS, commonly advertised as the next-generation firewall, proves its worth.

Over the last decade, the intersection of Machine Learning (ML) and IDS has gained the attention of cybersecurity researchers, primarily driven by breakthroughs in Artificial Inteligence (AI) [12, 13]. The ML algorithms empower IDS to learn patterns and identify anomalies from massive amounts of data, enabling the detection of both known and unknown intrusions. With the widespread use of containerized applications and cloud infrastructures, monitored systems are typically distributed across various geographical locations [14, 15]. To protect these modern complex environments, there is a need for a distributed hierarchical IDS, capable of effectively monitoring and protecting each layer from the user devices to the cloud infrastructure itself [16].

This chapter introduces ChronosGuard, a containerized implementation of a highly adaptable, multi-tiered hierarchical IDS designed to minimize latency and bandwidth requirements while preserving privacy constraints. A comprehensive experimental analysis has been conducted to measure ChronosGuard's performance through multiple metrics, such as latency, throughput, and resource utilization. Also, the evaluation assesses the influence of other factors including cloud infrastructure topologies, deployment strategies, container orchestrators, queue sorting algorithms, workload scenarios, and load patterns. The main contributions of this chapter are twofold:

- **Deployment-ready containerized hierarchical multi-stage IDS:** The implementation of a multi-stage hierarchical IDS, consisting of four containerized key components, deployed via four distinct deployment strategies. The artefacts [1] have been made publicly available to facilitate future research and replication of our results.
- **Comprehensive evaluation of a containerized IDS deployment:** Through a multitude of experiments conducted on a K8s cluster, the impact of infrastructure topologies, container orchestrators, queue sorting algorithms, deployment strategies, workload scenarios, and traffic load patterns on the performance of containerized IDSs is extensively studied.

4.2 Related Work

The field of IDS has evolved significantly since the adoption of rapid advancements in AI. Researchers increasingly adopt ML techniques to learn malicious patterns from data or detect suspicious activity deviating from baselines. Numerous studies focus on the design, implementation, and optimization of ML-based IDS, aiming to improve classification performance and tackle aspects such as recall, precision, and the number of false positives. However, a considerable gap remains in addressing scalability and the practical deployment of IDS in dynamic, real-world environments. Sharma et al. [17] conducted a systematic review on multi-objective optimizations for intrusion detection, analyzing 25 studies across a set of nine objectives. Yet, crucial aspects such

[1] https://github.com/idlab-discover/ChronosGuard

as scalability were not considered. Similarly, Liu et al. [18] performed a literature review on IDS in the cloud, and only 7% of the selected studies addressed scalability as part of their work. According to Apruzzese et al. [19], the current ML-IDS research does not meet the practical demands of industry developers which require an increased focus on operational viability and practical requirements. This demonstrates the general oversight in current IDS research and its implications for real-world applications.

Recently, several new ML-based architectures have been proposed. In their comparison of different IDS deployment strategies in the Internet of Things (IoT) landscape, Khraisat and Alazab [20] found that the hierarchical strategy was extremely deployable across large and heterogeneous networks, with the only drawback being the complexity of the IDS. Pundir et al. [21] specified three essential requirements for an effective IDS, including the necessity to minimize system and network resource consumption. In addition, Lai et al. [22, 23] have defined a theoretical three-stage ML IDS consisting of three in-sequence tasks: pre-processing, binary detection, and multi-class detection. Using queueing theory and simulated annealing, the authors evaluated ten different task assignments across edge, fog, and cloud. The results conclude that while each task assignment presents unique benefits and drawbacks, a hierarchical approach empowers cloud offloading, cost minimization, and enhanced privacy.

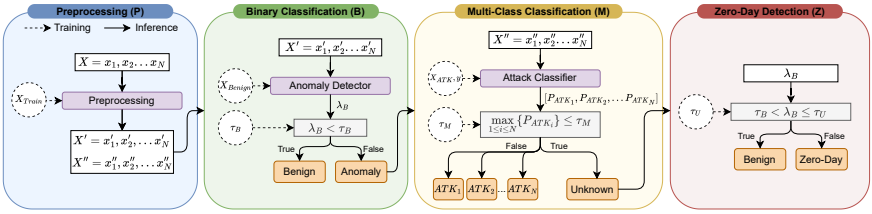
In summary, ChronosGuard is the first chapter, to the best of our knowledge, to extensively evaluate how various factors impact the performance of IDSs in containerized environments. These factors include network topologies, deployment strategies, container orchestrators, and traffic load patterns. The outcomes of this chapter, combined with an in-depth analysis of these results, offer unique insights that contribute towards the development of more scalable and efficient security-oriented applications.

4.3 System Design

This section introduces ChronosGuard, a containerized multi-stage ML-IDS specifically designed for hierarchical deployments. Building on our previous research [24], which optimized and analyzed the ML pipeline, ChronosGuard is evaluated through extensive real-world experiments using the K8s orchestration platform to assess its performance.

4.3.1 Components

ChronosGuard leverages a state-of-the-art multi-stage hierarchical IDS methodology consisting of four main components: preprocessing, binary classification, multi-class classification, and zero-day detection. Figure 4.1a illustrates the connections and internals of the key components within the ML-based IDS pipeline. This pipeline creates a cascading processing sequence where each network flow is sequentially analyzed through successive components until a final classification is achieved. An unsupervised one-class SVM and supervised random forest are used respectively for the anomaly detector and attack classifier with a combined performance of 0.9875



(a) Overview of the multi-stage hierarchical IDS four key components: preprocessing (P), binary classification (B), multi-class classification (M), zero-day detection (Z)

(b) Monolith (c) 2-pod (d) 3-pod (e) 4-pod

Figure 4.1: (a) Illustration of ChronosGuard, detailing the interactions of the four main components. (b-e) Illustration of the multiple deployment schemes, organized into numerous pods: (a) Monolith, (b) 2-pod, (c) 3-pod, and (d) 4-pod

f1-score, see our prior work [24] for more details on the hyperparameters and optimal thresholds of the ML pipeline.

Preprocessing (P) The first component receives a vector, $X = x_1, x_2, ...x_N$, encapsulating network flow characteristics such as packet count, duration, and packet inter-arrival times. This vector is transformed into a suitable format for subsequent analysis by the ML-based anomaly detector and multi-class classifier. **Binary Classification (B)** The second component performs a lightweight anomaly detection, assigning an anomaly score, λ_B, to each network flow based on the network flow characteristics. Flows with a corresponding anomaly score below a predefined threshold, τ_B, are classified as benign and are excluded from further analysis. Contrarily flows whose scores exceed this threshold are flagged as suspicious and are forwarded to the subsequent component. This filtering process forwards only a subset of all network flows to subsequent stages. By doing so, it significantly reduces bandwidth requirements and allows for the use of more computationally intensive techniques in the stages that follow. **Multi-class Classification (M)** The third component classifies flows identified as suspicious into known attack types based on confidence levels. A flow is assigned to the attack category with the highest calculated probability, unless this prediction confidence, P_{ATK_i}, falls below the threshold τ_M. Flows failing to meet the minimum confidence threshold are considered unrelated to any attack classes seen during training and are consequently forwarded to the last component for final analysis. **Zero-day Detection (Z)** The fourth and final component utilizes the anomaly score, λ_B, previously calculated by the Binary Classifier to distinguish between zero-day or unseen attack types and benign flows erroneously flagged as suspicious. Flows with scores exceeding a more rigorous threshold, τ_Z, are categorized as zero-day attacks, whereas those below are reclassified as benign.

4.3.2 Containerized Deployments

ChronosGuard's architecture is highly configurable within a containerized environment, impacting overall system performance significantly. This research evaluates four distinct deployment configurations, depicted in Figures 4.1b through 4.1e, ranging from a monolithic to a fully micro-serviced architecture. A "pod" in K8s refers to the smallest deployable unit, typically containing one or more containers that share resources. ChronosGuard's key components are preprocessing (P), binary classification (B), multi-class classification (M), and zero-day detection (Z).

Monolithic Deployment (PBMZ) integrates all four components within a single pod. This configuration allows the least flexibility in terms of scaling due to its non-granular scaling.

2-Pod Deployment (PB-MZ) separates the components into two groups: preprocessing with binary classification and multi-class classification combined with zero-day detection.

3-Pod Deployment (P'B-P"M-Z) divides the preprocessing component into two parts: preprocessing for the binary classifier (P'), which computes X', and preprocessing for the multi-class classifier (P''), which computes X''. These are paired with the binary (B) and multi-class (M) components, respectively. The zero-day detection remains by itself.

4-Pod Deployment (P-B-M-Z): Represents a true micro-service deployment, with each component operating independently within its own pod. This configuration allows the most scaling flexibility due to its granularity.

4.4 Methodology

This section outlines the methodology employed to evaluate ChronosGuard. The evaluation framework is described, including experiment automation, data collection, and load generation, followed by the specification of the used software and hardware in the experimental testbed.

4.4.1 Evaluation Framework

To support a comprehensive analysis of ChronosGuard's performance across various configurations and scenarios within cloud environments, a custom testbed has been designed and implemented on top of K8s. This testbed is designed to automate the execution of a multitude of experiments, each exploring a different permutation of factors that potentially impact the IDS's performance. Cumulatively, several thousand experiments were executed over the course of one month. The automation of this testing methodology is achieved through custom bash and Python scripts. These scripts are responsible for critical functions, such as experiment initialization, load generation, and data collection.

Key to the evaluation framework is the capability to provide a controlled and consistent environment for each test iteration. This is achieved by initiating each experiment with a fresh deploy-

ment within K8s, ensuring that no residual data or configurations from previous tests influence the results. This approach guarantees that each set of tests is conducted under uniform conditions, producing reliable and reproducible results. To facilitate this, a Bash script is used to orchestrate the experimental setup. The script interfaces directly with the K8s API using Kubectl, the command-line tool that allows for efficient management and operation of Kubernetes clusters.

Load Generation for ChronosGuard is generated using Locust [25], a popular open-source load testing framework. This framework enables the simulation of both benign and malicious user interactions with the IDS. Benign users submit benign network flows to the hierarchical IDS, simulating a normal baseline network operation, while malicious users send malicious network flows, emulating potential cybersecurity threats. To reflect a realistic scenario, the ratio of benign to malicious users is set at 4:1, similar to popular benchmark datasets [26]. Furthermore, the probability of a malicious user selecting a known attack type is intentionally set at twice the likelihood of selecting an unknown attack type, further mirroring potential real-world attack distributions.

At the beginning of each experiment, the load generator loads a collection of 59,435 network flows extracted from the CIC-IDS-2017 benchmark dataset [27], comprising benign flows, five distinct known attacks, and unknown attacks. After initialization, each user submits only one simultaneous request to ChronosGuard, ensuring a controlled load simulation. The simulated users randomly sample a network flow from the appropriate category from the initially loaded collection.

Data Collection within the ChronosGuard evaluation framework is divided into two primary categories: resource and performance metrics, each essential for a comprehensive analysis of system performance. This dual approach to data collection not only improves the understanding of ChronosGuard's operational efficiency but also enables performance evaluation by correlating resource utilization with system responsiveness.

Resource Metrics are continuously monitored by K8s kubelets, which track CPU usage, memory consumption, and network bandwidth every two seconds. This data is aggregated by Prometheus at the same interval. After each experimental run, the data is queried from the Prometheus metric server and saved in CSV format. This systematic collection and aggregation process ensures detailed resource usage statistics are available for subsequent analysis.

Performance Metrics are the primary focus here is on the collection of response times for each network request processed by ChronosGuard. To achieve this, custom event handlers within Locust are implemented to record the latency from when a network flow is transmitted to when its associated prediction is received. These response times are then exported into a single CSV file on the host running Locust, along with experiment and flow properties, providing a granular view of the system's responsiveness under various load conditions.

The Testbed Infrastructure utilizes the imec Virtual Wall (VWall) infrastructure located at IDLab, Belgium. The setup comprises a single K8s cluster with ten nodes (Ubuntu 20.04.2 LTS), each equipped with dual hexacore Intel E5645 CPUs, 24GB RAM, and dedicated Gigabit NICs. The K8s

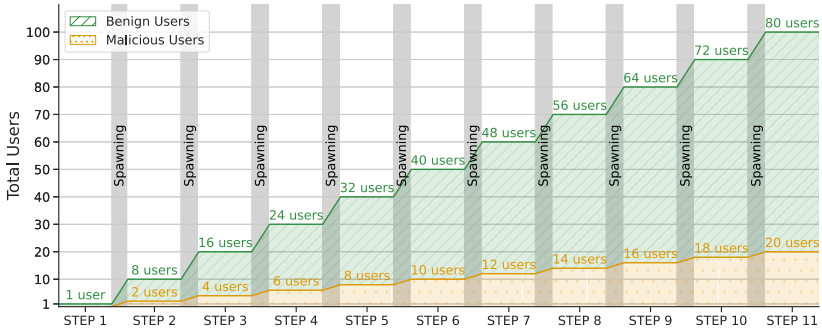


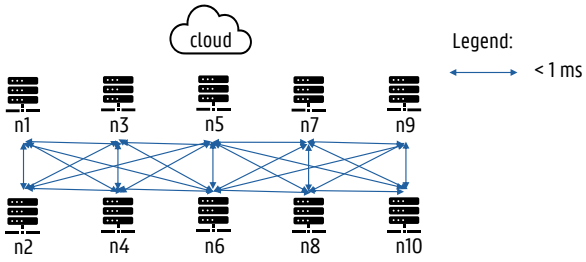
Figure 4.2: Diagram showing a single experimental run: traffic load incrementally increases from one to a maximum of 100 users.

Table 4.1: Overview of the experimental parameters.

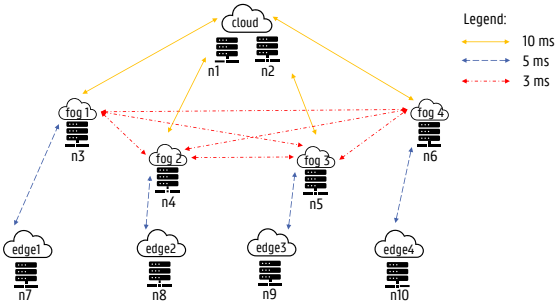
Variable	#	Values
Topology	2	Cluster, Edge-Cloud
Deployment	4	Monolith, 2-pod, 3-pod, 4-pod
Scheduler	2	KS, Diktyo
Sorting Alg.	9	Priority, QoS, Cycle, (Alt. / Rev.) Kahn / Tarjan
Scenarios	3	Initial, Scale-up, Scale-down
Users	11	1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Repetition	8/10	1-8 (Edge-fog-cloud), 1-10 (Cluster)

cluster has been set up with Kubeadm and Kubectl version v1.22.4, and Docker version 20.10.12. Network delays are emulated using TC [28] to simulate different network topologies.

The Experimental Setup systematically evaluates each permutation of variables through a series of repeated independent experiments, each comprising eleven distinct steps characterized by a progressively increasing load. In each step, network load is generated by a predefined number of users over a 30-second interval. This period is followed by a 10-second user spawning phase, during which new users are gradually introduced at a consistent rate of one user per second. The procedural flow of these operations is illustrated in Figure 4.2. The experiment begins with a single user, gradually increasing the user count linearly in each subsequent step, ending with 100 users in the final step. To reflect a realistic scenario, the ratio of benign to malicious users is set at 4:1. At each step, key performance and resource utilization metrics are collected. These detailed results at every stage enable a thorough assessment of the system's scalability and responsiveness, providing insights into its performance under varying operational loads and experiment configurations.



(a) Cluster topology with similar network connections.



(b) Edge-fog-cloud topology with different network delays.

Figure 4.3: Illustration of the evaluated cloud infrastructures.

4.4.2 Evaluated Variables

Table 4.1 presents a detailed overview of the evaluated variables and their respective values.

Network Topology Figure 4.3 shows the two distinct cloud topologies evaluated in this chapter: the cluster and edge-fog-cloud. On one hand, Figure 4.3a presents a highly available cluster configuration, where nodes are provisioned within a single region with similar network connections. Consequently, this produces negligible inter-node delays in the order of microseconds. On the other hand, Figure 4.3b presents a multi-region cluster, where network delays vary considerably. Delays range from negligible among the cloud nodes to as much as 15ms between cloud and edge.

Deployment Strategy In this chapter, four distinct deployment configurations are evaluated, ranging from a combined monolithic deployment of all four stages to a true micro-service deployment, where each stage is deployed individually. The CPU and memory requests and limits for each configuration are displayed in Table 4.2. The resource allocation is designed such that the cumulative total remains consistent across all deployment strategies. Additionally, the first components (P, B) are allocated a relatively larger share of resources compared to the subsequent components (M, Z), as they handle the full load before passing only a fraction of the requests to the subsequent ML components in the pipeline.

Table 4.2: Deployment properties of ChronosGuard components: CPU and memory requests (R) and limits (L).

App.	Deployment	CPU R/L (mCPU)	MEM R/L (MiB)
Monolith	PBMZ	1000/2000	512/1024
2-pod	PB	700/1400	384/768
	MZ	300/600	128/256
3-pod	P'B	500/1000	256/512
	P"M	400/800	192/384
	Z	100/200	64/128
4-pod	P	450/900	192/384
	B	250/500	128/256
	M	200/400	128/256
	Z	100/200	64/128

Orchestration Schedulers and Sorting Algorithms The K8s component responsible for scheduling operations is known as KS, the default scheduler in K8s. Our experiments considered two distinct schedulers: KS and Diktyo [29]. While KS primarily focuses on optimizing resource usage across all cluster nodes, Diktyo considers both application dependencies and infrastructure topology during pod scheduling within K8s to find network-aware placement schemes for containerized applications. To determine the optimal allocation order, Diktyo also considers pod dependencies specified by developers to sort pods based on topological sorting information [30]. Our evaluation assessed several topological sorting algorithms, which define the preferred deployment order for pods in a multi-pod application with interdependencies, as detailed in Table 4.3. In contrast, KS applies the default priority sorting algorithm available in K8s, yielding an identical order to *Kahn* since all our pods have the same priority level.

Scenario Each experiment is conducted across three sequential scenarios: the initial phase, with one instance per pod in the deployment strategy; the scale-up phase, where each pod is increased to five replicas; and the scale-down phase, where three instances of each pod are terminated, leaving two instances per pod. This increases the total available resources by 5-fold and 2-fold compared to the initial phase and enables studying the impact of application scheduling.

Simulated Users The variable *users* quantifies the number of concurrent users spawned by the Locust load generator. Each user is configured to only send a single concurrent request to ChronosGuard; therefore, the number of users directly determines the load intensity directed at the system. Thus, an increase in the number of users directly corresponds to an increased load.

Repetition To ensure statistical significance and reliability of the experimental results, each experiment is replicated multiple times. Specifically, for the cluster topology, each experiment is replicated ten times, with the load generator, simulating an ingress point, systematically rotated across each node in the cluster to ensure a balanced evaluation. Similarly, for the edge-fog-cloud

Table 4.3: Orchestration order for the different deployment schemes of ChronosGuard.

Deployment	Algorithm	Topological order
Monolith	*KS*	[PBMZ]
2-pod	*KS*	[PB, MZ]
	Kahn Tarjan & Cycle	[PB, MZ]
	Alt. Kahn & Alt. Tarj.	[PB, MZ]
	Rev. Kahn & Rev. Tarj.	[MZ, PB]
3-pod	*KS*	[P'B, P"M, Z]
	Kahn Tarjan & Cycle	[P'B, P"M, Z]
	Alt. Kahn & Alt. Tarj.	[P'B, Z, P"M]
	Rev. Kahn & Rev. Tarj.	[Z, P'M, P"B]
4-pod	*KS*	[P, B, M, Z]
	Kahn Tarjan & Cycle	[P, B, M, Z]
	Alt. Kahn & Alt. Tarj.	[P, Z, B, M]
	Rev. Kahn & Rev. Tarj.	[Z, M, B, P]

topology, each experiment is repeated eight times, with the load generator cycled twice over each edge node.

4.5 Results

This section presents the obtained results through the evaluation of ChronosGuard. The results analyze the effectiveness of different architectural approaches across several performance metrics including response times, throughput, and resource utilization.

4.5.1 Prioritization of Benign Traffic

In an optimal scenario, an IDS should minimally disrupt the normal operations of the networks it protects, adding as little latency as possible to benign traffic while detecting attacks within seconds. Our analysis of the experimental data collected reveals a significant difference in processing times between benign and malicious network flows. Notably, benign traffic benefits from substantially lower median response times than attack and zero-day traffic across all evaluated deployment strategies, as illustrated in Figure 4.4. The most pronounced differences in response times are observed in the configurations utilizing 2-pod and 3-pod deployments, where the median response time for benign flows is reduced by 85% and 75%, respectively, in comparison to malicious traffic. Conversely, the difference in response times is less pronounced within the monolith and 4-pod deployment, with benign traffic showing a smaller reduction in median response times of respectively 10% and 12% relative to malicious flows. These results for the 4-pod deployment deviate from the other microservice deployments, which can be attributed to suboptimal static

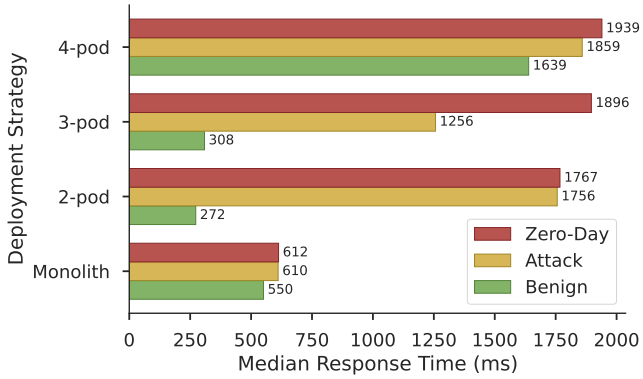


Figure 4.4: Median response times by flow type: a significant reduction for benign versus malicious traffic across all deployment strategies, minimizing latency for normal operations.

resource allocation across the pods (see Section 4.5.4). The lowest median response times for benign traffic were observed for the 3-pod deployment during the scale-up scenario with 1 and 10 users, respectively, yielding 27 and 46 ms of latency.

In conclusion, the multi-stage hierarchical IDS consistently prioritizes benign traffic over malicious by achieving lower average processing times, regardless of the deployment strategy selected. However, it is the adoption of a micro-service architecture that significantly enhances this effect, demonstrating the capability to reduce benign response times by up to eight-fold compared to a monolithic deployment through strategically offloading the processing of suspicious network flows and the ability to allocate resources for each IDS component independently. Moreover, a micro-service architecture can reduce the absolute median response time for benign flows up to 51% (for 2-pod) compared to a monolith architecture.

4.5.2 Cluster vs Edge-Fog-Cloud architecture

The comparative analysis of the results reveals no significant differences between the cluster and edge-fog-cloud topologies in terms of throughput, latency, or resource utilization. Figure 4.5 presents the throughput averaged over the repeated experiment runs, relative to the load generated by the locust users for the default KS across the three scenarios and two topologies. The results were nearly identical for the initial, scale-down, and scale-up scenarios in both topologies. The detailed performance metrics in Table 4.4 reveal only minor variations between the cluster and edge-fog-cloud topology in terms of throughput and response time. The minimal introduced delay in node-to-node communication within the edge-fog-cloud topology does not significantly affect performance. Specifically, the small increase in communication delay, in the order of a few milliseconds, is considered negligible in contrast to the more substantial inference times associated with the employed ML models.

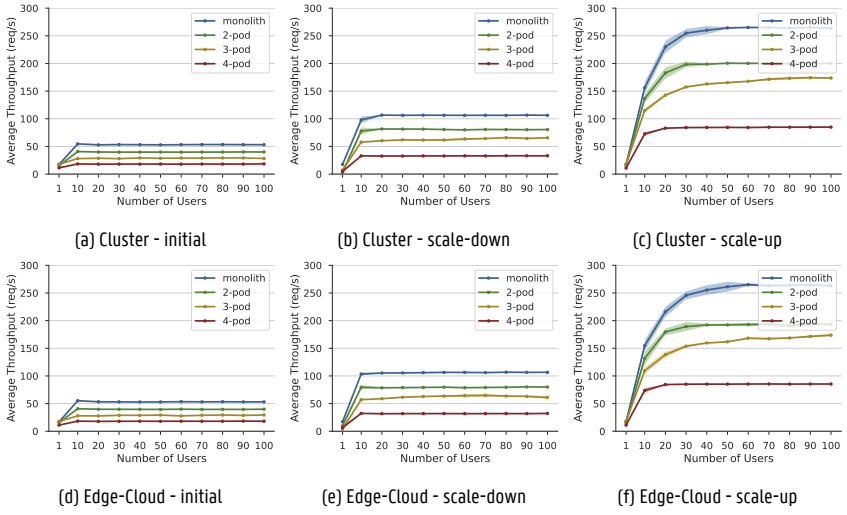


Figure 4.5: The impact of the deployment strategy on the average obtained throughput: a higher number of K8s pods per deployment has a significantly negative influence on the throughput. Once resources are saturated, increasing the users has no further effect.

Table 4.4: Impact of the topology on throughput and response times for KS, computed over all users and scenarios.

Scenario	Topology	Throughput (req/s)	Response Time (ms)		
			Median	Mean	95th Pctl
Initial	Cluster	33	1235	1725	3364
	Edge-fog-cloud	33	1240	1730	3380
Scale-down	Cluster	65	637	882	1856
	Edge-fog-cloud	64	640	896	1845
Scale-up	Cluster	157	267	349	796
	Edge-fog-cloud	154	266	351	817

4.5.3 Topological-aware Orchestration

Topological-aware orchestration represents a paradigm within application deployment that takes into account pod dependencies and inter-node network latencies, in contrast to the default KS, which prioritizes optimal resource utilization across all the cluster nodes. This methodology ensures that the deployment order is optimized for network efficiency, potentially reducing latency and enhancing overall application performance. The selected deployment strategy is crucial as it directly correlates with the number of pods to be scheduled. For instance, a monolithic application, characterized by a singular pod, presents only one deployment permutation and lacks

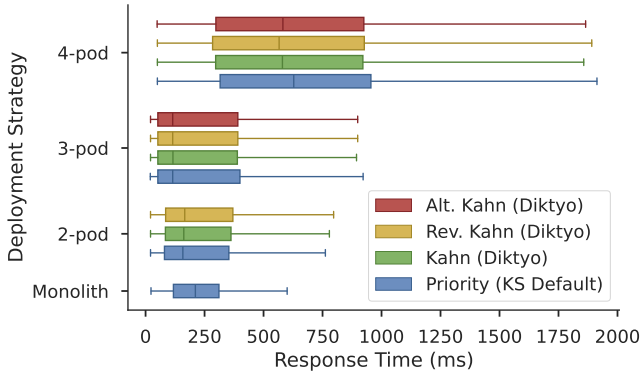


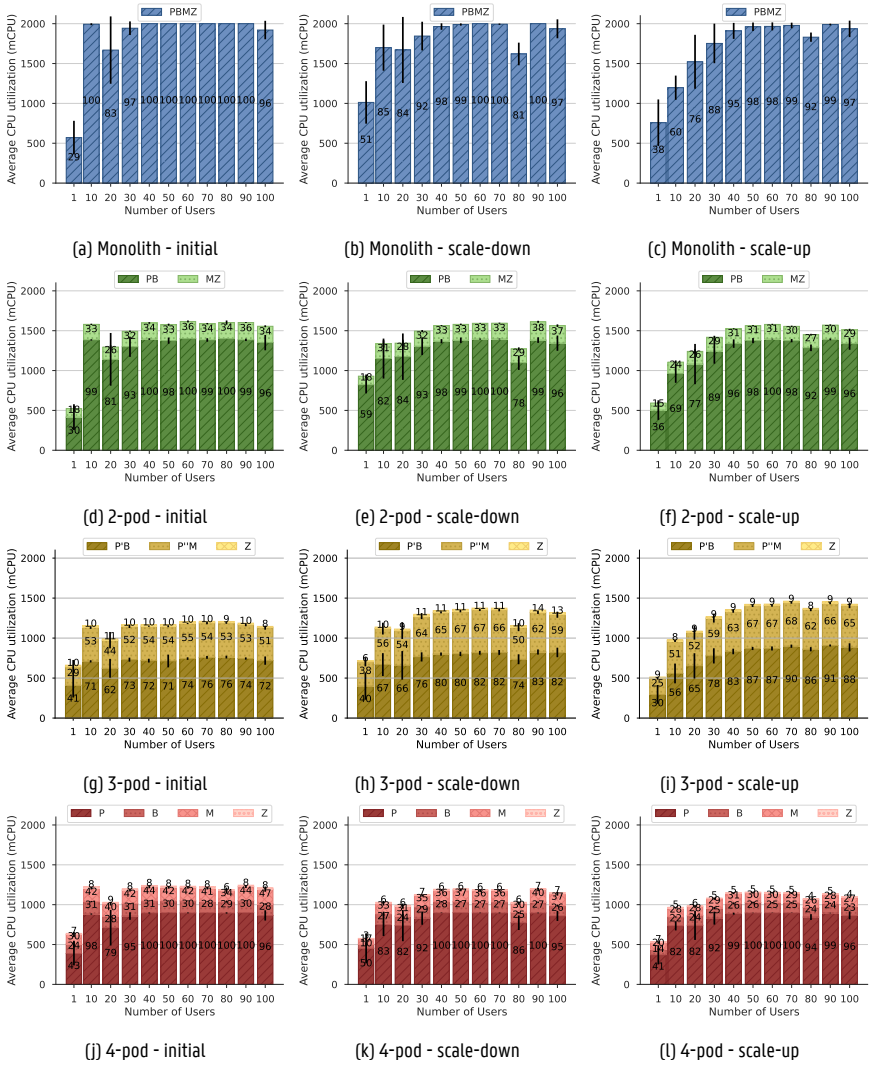
Figure 4.6: Distribution of response times across deployment strategies in cluster topology during scale-up scenario.

communication inter-dependencies, thus simplifying the orchestration process. Conversely, as the complexity and pod count of an application increase, so do the permutations for network-aware sorting and inter-pod dependencies, introducing a multitude of potential deployment sequences. The load scenario also significantly impacts the orchestration process, with the required number of pods varying—specifically, one, two, and five pods, respectively, for each scenario, multiplied by the number of pods of the selected deployment strategy.

Empirical evidence supporting the efficacy of network-aware sorting algorithms within topological-aware orchestration is presented in Figure 4.6. It illustrates the distribution of response times across different deployments within a cluster topology during scale-up. Results reveal that the influence of network-aware orchestration on application deployment demonstrates a correlation to the number of pods to be scheduled. For simpler deployments, such as those involving monolithic, 2-pod, and 3-pod, the impact of network-aware sorting on response times is negligible. This observation is consistent with insights from earlier studies [31], indicating a limited scope for optimization in less complex applications. However, noticeable variations are observed as the deployment complexity increases to four pods. In this case, network-aware orchestration regardless of the sorting algorithm selected, demonstrated the capability to enhance performance, as demonstrated by slightly lower response times. Particularly, changing from the default KS to the Reverse Kahn sorting algorithm decreased the average response times by 10%, significantly lowering the response time for the first quartile, median, and last quartile from 316 to 284, 628 to 566, and 955 to 927 ms, respectively. This signifies a tangible, albeit limited, improvement in deployment efficiency through topological-aware orchestration and sorting.

4.5.4 Joint Task Assignment in Container Clouds

This analysis evaluates four unique deployment strategies for container-based cloud environments (previously presented Figure 4.1), comparing their performance to determine the most ef-



(a) Monolith - initial (b) Monolith - scale-down (c) Monolith - scale-up

(d) 2-pod - initial (e) 2-pod - scale-down (f) 2-pod - scale-up

(g) 3-pod - initial (h) 3-pod - scale-down (i) 3-pod - scale-up

(j) 4-pod - initial (k) 4-pod - scale-down (l) 4-pod - scale-up

Figure 4.7: The average CPU utilization across deployment strategies and scenarios under varying loads: highlighting efficiency and scalability of micro-service architectures. The bars represent the usage percentage of the pod's CPU limit with 95% CI.

Table 4.5: Throughput and response times across the different scenarios for the KS and cluster topology.

Deployment	Scenario	Throughput (req/s)	Response Time (ms)		
			Median	Mean	95th Pctl
Monolith	Initial	50	971	974	1114
	Scale-down	98	486	490	627
	Scale-up	228	199	203	321
2 pods	Initial	38	440	1300	3313
	Scale-down	73	278	645	1809
	Scale-up	176	164	265	751
3 pods	Initial	28	680	1765	5567
	Scale-down	58	261	814	2941
	Scale-up	147	107	311	1161
4 pods	Initial	17	2847	2861	3463
	Scale-down	30	1523	1579	2045
	Scale-up	77	599	619	950

fective configuration and the trade-offs to consider. Among the configurations tested, the monolithic deployment outperforms the other strategies on the performance metrics, such as throughput and upper-bound response times. While it exhibits lower upper-bound response times, it is important to note that higher median response times were observed than most micro-service deployments. The greater performance of the monolithic architecture can be attributed to the absence of inter-pod communication, thereby reducing overhead when compared to micro-service architectures and eliminating the need for complex resource allocation strategies within ChronosGuard. However, this strategy does present drawbacks in terms of non-granular scaling and deployment flexibility:

- **Local Deployment**: Preserves data privacy but fails to leverage broader cloud computing benefits.
- **Cloud Deployment**: Compromises privacy and requires higher bandwidth, mitigating the primary advantages of localized computation.

With an increase in the number of pods within the deployment, throughput tends to decrease, a trend that primarily can be attributed to the suboptimal allocation of resources among the pods. Figure 4.7 illustrates this trend, showing that while the monolith utilizes resources at 100% capacity, deployments with 2, 3, and 4 pods utilize only 75%, 65%, and 55%, respectively. This is caused by underestimating the resource demands for the preprocessing component, as observed in the 4-pod deployment strategy where the preprocessing pod reaches maximum CPU utilization under low loads, while the other components remain underutilized. Similar, albeit less pronounced under-utilization of pod resources is present for the 3 and 2 pod deployment strategies.

When the throughput values are adjusted for resource utilization, a smaller performance gap is revealed between the micro-service architectures and the monolith application, indicating that the reduced throughput in multi-pod deployments is partly due to the inherent overhead associated with micro-services. This overhead includes the additional computational resources required for inter-service communication, such as serialization and deserialization.

Despite their computational overhead, microservice architectures can reduce overall response times, see Table 4.5, minimizing the disruption for normal traffic and speeding up the detection of known and unknown attacks. This advantage results from the ability to deploy each component independently, allowing the system to handle the majority of traffic—which is benign—without being slowed down by the computationally intensive ML techniques required for attack detection. From a resource utilization perspective, micro-service architectures consume approximately 2x to 2.5x more memory than monolithic setups. Furthermore, while they may require more CPU resources, they enable more sophisticated hierarchical deployments. For example, a deployment model that localizes the preprocessing and binary detection components can significantly reduce bandwidth requirements toward the cloud—by 25%, 33%, and 44% for 2, 3, and 4 pod deployments, respectively, compared to a monolithic deployment.

Overall, while monolithic architectures offer robust performance and lower resource demands, micro-service architectures provide enhanced flexibility and responsiveness, particularly in complex, scalable environments.

4.5.5 Limitations

A limitation of this chapter is the static resource allocation among ChronosGuard components. The total sum of resources assigned across all deployment strategies is equal and the initial components received a larger share, as they handle the full load and pass only a fraction to subsequent components. This allocation has proven suboptimal and could be further improved, potentially by dynamically scaling resources based on real-time demand. Additionally, the scope of this evaluation of ChronosGuard is limited to the containerized components of the ML pipeline. Crucial aspects such as traffic collection and flow reconstruction, which are essential to IDS, were not included in this chapter. Future studies could expand on this by incorporating these components to provide a more end-to-end assessment of IDS performance.

4.6 Conclusion

In this work, a containerized hierarchical ML IDS, ChronosGuard, consisting of four individual components, that can be deployed using four deployment strategies, is presented. Through systematic experimentation, the impact of various factors including network topology, workload orchestration, and throughput on the performance and resource utilization of each deployment strategy is evaluated. The analysis resulted in several key insights: benign traffic is effectively prioritized across all configurations with a maximum reduction of 85% for the median response

times compared to malicious traffic and up to 51% lower response times for the micro-service architectures compared to the monolithic deployment; minor network delays have negligible effects on throughput and response times; network-aware orchestration enhances the performance of complex security ML-based applications in cloud environments up to 10%; and there are notable trade-offs associated with different task assignment strategies in containerized settings. This research not only introduces a deployment-ready, containerized IDS to the field but also establishes a robust foundation for future studies aimed at optimizing the containerization of ML-based security components.

Acknowledgments

José Santos is funded by the Research Foundation Flanders (FWO), grant number 1299323N. This work is supported by the Belgian Chancellery of the Prime Minister (Grant: AIDE-BOSA).

Bibliography

[1] Y.-C. Lai, D. Sudyana, Y.-D. Lin, M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Machine learning based intrusion detection as a service: Task assignment and capacity allocation in a multi-tier architecture," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, ser. UCC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3492323.3495613

[2] Y.-C. Lai, D. Sudyana, Y.-D. Lin, M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. D. Turck, "Task assignment and capacity allocation for ml-based intrusion detection as a service in a multi-tier architecture," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[3] A. Singh and K. Chatterjee, "Cloud security issues and challenges: A survey," *Journal of Network and Computer Applications*, 2017.

[4] F. Sierra-Arriaga, R. Branco, and B. Lee, "Security issues and challenges for virtualization technologies," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–37, 2020.

[5] Z. Cheng, M. Apostolaki, Z. Liu, and V. Sekar, "Trustsketch: Trustworthy sketch-based telemetry on cloud hosts," in *The Network and Distributed System Security Symposium (NDSS)*, 2024.

[6] D. P. McKee, Y. Giannaris, C. Ortega, H. E. Shrobe, M. Payer, H. Okhravi, and N. Burow, "Preventing kernel hacks with hakcs." in *NDSS*, 2022, pp. 1–17.

[7] A. Van't Hof and J. Nieh, "{BlackBox}: a container security monitor for protecting containers on untrusted operating systems," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 683–700.

[8] F. Wei, H. Li, Z. Zhao, and H. Hu, "{xNIDS}: Explaining deep learning-based network intrusion detection systems for active intrusion responses," in *32nd USENIX Security Symposium (USENIX Security 23)*.

[9] T. Saranya, S. Sridevi, C. Deisy, T. D. Chung, and M. A. Khan, "Performance analysis of machine learning algorithms in intrusion detection system: A review," *Procedia Computer Science*, 2020.

[10] A. Voronkov, L. H. Iwaya, L. A. Martucci, and S. Lindskog, "Systematic literature review on usability of firewall configuration," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–35, 2017.

[11] L. Ceragioli, P. Degano, and L. Galletta, "Are all firewall systems equally powerful?" in *Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security*, 2019, pp. 1–17.

[12] G. Apruzzese, L. Pajola, and M. Conti, "The cross-evaluation of machine learning-based network intrusion detection systems," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 5152–5169, 2022.

[13] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When malware is packin'heat; limits of machine learning classifiers based on static analysis features," in *Network and Distributed Systems Security (NDSS) Symposium 2020*.

[14] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[15] A. Luckow, K. Rattan, and S. Jha, "Pilot-edge: Distributed resource management along the edge-to-cloud continuum," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 874–878.

[16] R. V. Mendonça, A. A. Teodoro, R. L. Rosa, M. Saadi, D. C. Melgarejo, P. H. Nardelli, and D. Z. Rodríguez, "Intrusion detection system based on fast hierarchical deep convolutional neural network," *IEEE Access*, vol. 9, pp. 61 024–61 034, 2021.

[17] S. Sharma, V. Kumar, and K. Dutta, "Multi-objective optimization algorithms for intrusion detection in IoT networks: A systematic review," *Internet of Things and Cyber-Physical Systems*, vol. 4, pp. 258–267, Jan. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667345224000038

[18] Z. Liu, B. Xu, B. Cheng, X. Hu, and M. Darbandi, "Intrusion detection systems in the cloud computing: A comprehensive and deep literature review," *Concurrency and Computation: Practice and Experience*, 2022.

[19] G. Apruzzese, P. Laskov, and J. Schneider, "SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection," Apr. 2023, arXiv:2305.00550 [cs]. [Online]. Available: http://arxiv.org/abs/2305.00550

[20] A. Khraisat and A. Alazab, "A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges," *Cybersecurity*, vol. 4, no. 1, p. 18, Mar. 2021. [Online]. Available: https://doi.org/10.1186/s42400-021-00077-7

[21] S. Pundir, M. Wazid, D. P. Singh, A. K. Das, J. J. P. C. Rodrigues, and Y. Park, "Intrusion Detection Protocols in Wireless Sensor Networks Integrated to Internet of Things Deployment: Survey and Future Challenges," *IEEE Access*, 2020, conference Name: IEEE Access.

[22] Y.-C. Lai, D. Sudyana, Y.-D. Lin, M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Machine learning based intrusion detection as a service: task assignment and capacity allocation in a multi-tier architecture," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, Dec. 2021.

[23] Y.-C. Lai, D. Sudyana, Y.-D. Lin, M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. D. Turck, "Task Assignment and Capacity Allocation for ML-based Intrusion Detection as a Service in a Multi-tier Architecture," *IEEE Transactions on Network and Service Management*, 2022.

[24] M. Verkerken, L. D'hooge, D. Sudyana, Y.-D. Lin, T. Wauters, B. Volckaert, and F. D. Turck, "A Novel Multi-Stage Approach for Hierarchical Intrusion Detection," *IEEE Transactions on Network and Service Management*, 2023.

[25] "locustio/locust," Mar. 2024, original-date: 2011-02-17T11:08:03Z. [Online]. Available: https://github.com/locustio/locust

[26] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, "Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018," in *2022 IEEE Conference on Communications and Network Security (CNS)*, Oct. 2022, pp. 254–262.

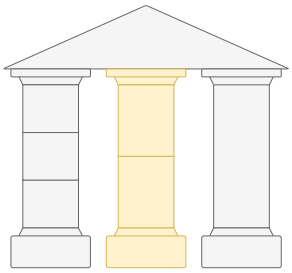
[27] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006639801080116

[28] A. N. Kuznetsov, "tc(8) — linux manual." accessed on 28 May 2023. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html.

[29] J. Santos, C. Wang, T. Wauters, and F. De Turck, "Diktyo: Network-aware scheduling in container-based clouds," *IEEE Transactions on Network and Service Management*, 2023.

[30] T. Ahammad, M. Hasan, and M. Zahid Hassan, "A new topological sorting algorithm with reduced time complexity," in *Intelligent Computing and Optimization: Proceedings of the 3rd International Conference on Intelligent Computing and Optimization 2020 (ICO 2020)*, 2021.

[31] J. Santos, M. Verkerken, L. D'Hooge, T. Wauters, B. Volckaert, and F. De Turck, "Performance Impact of Queue Sorting in Container-Based Application Scheduling," in *2023 19th International Conference on Network and Service Management (CNSM)*, Oct. 2023, pp. 1–9, iSSN: 2165-963X. [Online]. Available: https://ieeexplore.ieee.org/document/10327792



5

RustiFlow: Bridging the Gap Between Security Research and Practice using eBPF-based Network Flow Extraction

This chapter contributes to the second thematic pillar of this dissertation, Research to Practice, by addressing the fourth research question (RQ4) through an analysis of network flow feature extractors and their impact on the practical adoption of ML-NIDS.

The chapter first presents a systematic literature review of recent ML-NIDS research, analyzing the network flow feature extractors commonly used in the field. The review reveals a fragmented ecosystem of tools, with a heavy reliance on custom solutions that often lack performance, standardization, and reproducibility. Building on these findings, the chapter benchmarks the most widely used flow extractors in both real-time and offline processing, highlighting significant limitations in throughput, resource efficiency, and robustness. To overcome these challenges, we introduce RustiFlow*, an open-source, high-performance network flow feature extractor built with eBPF and implemented in Rust. Designed to bridge the gap between research and production environments,* RustiFlow *excels in real-time and offline processing, offers compatibility with multiple benchmark feature sets, and provides flexible configuration and extensibility for diverse use cases. The chapter concludes with two real-world case studies demonstrating* RustiFlow*'s robustness, scalability, and suitability for production deployments. As such, this work provides a critical missing link between ML-NIDS research and practical deployment.*

⋆ ⋆ ⋆

M. Verkerken, M. Callewaert, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck

Accepted for publication in the proceedings of the IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), June 2025

Abstract Large organizations generate billions of network flows daily, creating a high-velocity data challenge for modern security monitoring and threat detection. Researchers frequently develop custom flow extraction tools tailored for AI-driven security analyses, but these solutions often lack the performance, scalability, and interoperability required for real-world use. At the same time, existing production-ready flow extractors lack flexibility and customization, limiting their application for advanced security research.

To bridge this gap, we introduce RustiFlow, an open-source, eBPF-based network flow feature extractor developed in Rust. Designed for both research and operational deployments, RustiFlow delivers high throughput, real-time processing, and modular feature extraction, ensuring adaptability across diverse security applications. Our performance evaluation demonstrates that RustiFlow outperforms established extractors such as NFStream, nProbe, and CICFlowMeter, offering the fastest offline PCAP processing and zero packet loss while monitoring a multi-gigabit interface under load, while maintaining minimal resource overhead. Real-world case studies in a university data center and a network security testbed validate RustiFlow's reliability, efficiency, and practical applicability. During a 24-hour test in a data center, RustiFlow processed over 1 billion packets and 5.8TB of traffic with zero packet loss, while maintaining stable resource usage. In an adversarial security scenario, it operated with negligible resource consumption, demonstrating its efficiency for constrained environments. RustiFlow has the potential to become an essential tool for AI-based network security analysis, empowering future research and closing the gap between research and practice.

5.1 Introduction

Network traffic analysis has long been essential in understanding and monitoring computer networks [1]. As ML techniques continue to advance, there has been a shift towards ML-based analysis, classification, and predictions based on network traffic [2]. ML-NIDS is one of the oldest applications of ML in cybersecurity, drawing significant attention from both academic researchers and industry practitioners [3, 4, 5]. A popular approach is to rely on statistical network flow features as (one of the) input(s) of such an ML model, which provides a high-level aggregation of network packets [6]. However, the adoption of network flow analysis for ML-based security applications remains challenging due to skepticism among security practitioners regarding ML-based traffic analysis [7, 8, 9, 10]. One of the main limitations is the use of unrealistic evaluation and deployment environments. This chapter argues that the shortcomings of existing feature extraction tools are a major contributing factor.

Specifically, existing flow feature extraction tools were initially developed for observability-focused projects, such as netflows or IPFIX probes, and were not designed to meet the requirements of security analysis [11]. This mismatch in application focus not only results in many small custom research tools tailored to specific researchers' needs but also leads to a lack of standardized feature sets across the adopted flow extractors [12, 13]. Without a common feature set, researchers struggle to cross-validate models trained on different data sources, further complicating security analysis and leading to low ML generalization [14, 15, 16]. Moreover, these research-oriented tools often lack the performance and operational robustness required for real-world security applications, reinforcing skepticism among security practitioners regarding ML-based traffic analysis [7, 8, 9, 10]. In addition to these issues, most available tools are constrained to either offline processing or real-time traffic handling, but not both, limiting their practicality in dynamic security environments. This fragmented landscape widens the gap between security research and practice, limiting the adoption of advanced traffic analysis techniques in practice[17, 18].

To address these limitations, we present RustiFlow, a high-performance, open-source[1] network flow feature extractor designed to close the gap between research and practice. RustiFlow empowers researchers and practitioners to (i) work with an expanded superset of features from widely used feature extractors, (ii) handle both real-time and offline traffic analysis, and (iii) customize configuration parameters (e.g., flow expiration, metadata inclusion) to suit diverse research and application requirements.

RustiFlow leverages extended Berkeley Packet Filter (eBPF), a powerful technology that enables efficient and programmable in-kernel packet processing on Linux systems [19]. By combining the flexibility of eBPF with the performance and memory safety guarantees of the Rust programming language, RustiFlow delivers robust flow reconstruction and feature extraction in a high-throughput, resource-efficient manner. As a result, it serves not only as a practical tool for security researchers but also as a building block for future work that aims to bridge academic development with the operational needs of real-world security environments.

Contributions. First, we review the recent literature for network flow feature extractors to position our work in Section 5.2. Then, we provide the following threefold contributions. We:

- Design and implement RustiFlow in Section 5.3: an open-source, eBPF-based network flow feature extractor offering high performance, stability, and flexibility for research and practical applications.
- Perform a comprehensive performance comparison of RustiFlow against existing flow extractors, highlighting its capabilities, in Section 5.4.
- Demonstrate RustiFlow's robustness and applicability in practical network security scenarios with **two real-world case studies** in distinct environments—a university data center and a network security test lab, in Section 5.5.

[1] Code available at https://github.com/idlab-discover/RustiFlow

5.2 Background and Related Work

This section surveys the use of network flow feature extractors in the context of ML-based security research, identifying the issues that motivate the design of RustiFlow. We begin with an overview of our literature review methodology (§5.2.1) and present its findings regarding existing tools and their limitations (§5.2.2). Finally, a brief overview of eBPF technology and its applications in security research is provided (§5.2.3).

5.2.1 Literature Review Methodology

Our review methodology follows the recent user-centric literature survey by Dietz et al. [20], which examines IDS research from a stakeholder's perspective. We selected papers published in top-tier network management, security, and privacy conferences and journals between 2018 and 2023, based on the CORE ranking. Venues that received an "A" ranking at least once during this period were included, covering six network management (CNSM, SIGCOMM, ICC, IM, NOMS, TNSM) and six security/privacy (NDSS, CCS, Usenix Security, S&P, ESORICS, TrustCom) venues, along with their associated workshops.

To refine our selection, we included only papers that explicitly mention "anomaly detection" or "intrusion detection" in the full text and focus on network traffic features in enterprise networks. This filtering resulted in 155 unique papers. Next, two researchers independently reviewed these papers, analyzing the network datasets and flow exporters used. Of these, 85% employed network flow features; the remaining 24 papers were excluded.

For the remaining 131 papers, we documented the network flow exporters used. If an exporter was not explicitly stated, we assumed the dataset's default flow features were used (e.g., CICFlowMeter for CIC-IDS-2017). When custom flow features were derived from existing datasets, both the original dataset's exporter(s) and any additional tools were recorded. If no specific tool was mentioned, we labeled it as "custom". All tools that output Cisco "NetFlows" are aggregated into a single "netflow" category and will be represented by the most commonly used tool, nProbe [21].

To ensure consistency, two researchers independently reviewed and cross-validated all papers. This systematic approach provides a comprehensive overview of how flow feature extractors are utilized in ML-NIDS research within network management, security, and privacy communities.

5.2.2 Limitations of Flow Extraction Tools

Figure 5.1 presents the flow feature extractors that appear in at least two studies in our literature review. Remarkably, over half of the analyzed papers (74) relied on custom tooling. While researchers develop custom tools to explore and evaluate novel flow features, the lack of standardized solutions prevents reproducibility and real-world adoption. Many of these custom tools are not open-sourced [34], are designed for narrow research objectives, and lack the robustness needed for real-world environments.

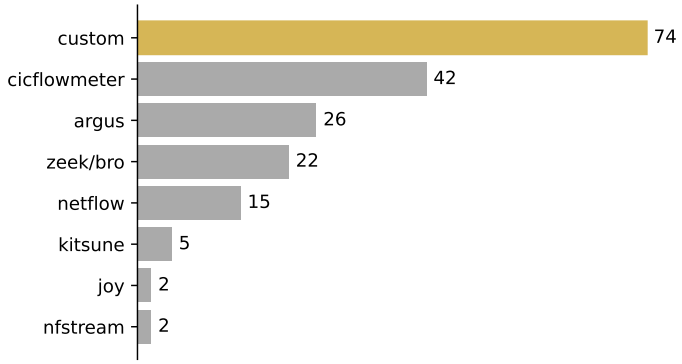


Figure 5.1: **Custom flow extractors dominate in research.** The figure presents the flow extraction tools mentioned in at least two papers, ordered by frequency. Some papers evaluate multiple tools, leading to a total tool count exceeding the number of reviewed papers.

Table 5.1: Overview of the Flow Extractors used in Security and Networking Research.

Name	Open-Source	Release Year	Language	Custom Features	Offline	Realtime	GUI	Compatible Feature Sets
Argus [22]	✓	1984*	C	✗	✓	✓	✗	✗
Zeek [23, 24]	✓	1990*	C++	✓	✓	✓	✗	✗
nProbe [21]	✗	2003*	C	✗	✓	✓	✗	✗
Joy [25]	✓	2016	C	✗	✓	✓	✗	✗
Kitsune [26, 27]	✓	2018	Python, Cython	✗	✓	✗	✗	✗
CICFlowMeter [28, 29]	✓	2018*	Java	✗	✓	GUI Only	✓	✗
Go-Flows [12, 30]	✓	2020	Go	✓	✓	✗	✗	✗
NFStream [31, 32]	✓	2022*	Python, Cython	✓	✓	✓	✗	✗
RustiFlow (Ours) [33]	✓	2025*	Rust, eBPF	✓	✓	✓	✓	CICFlowMeter, CIDDS, NFStream

*Received at least one update in the last year

The most widely used open-source tool is CICFlowMeter [35, 36], primarily due to its connection to the popular datasets from the Canadian Institute for Cybersecurity (e.g., CICIDS2017, CICIDS2018). While CICFlowMeter has enabled significant ML-based security research, it has several limitations. Engelen et al.[28] identified three logical errors in the source code, including incorrect TCP termination handling, while Flood et al.[37] reported CICFlowMeter crashing when processing large traffic captures. A forked version of CICFlowMeter [38] fixed among others the logical errors, but the performance issues and lack of real-time support are still present. Other frequently used extractors, such as Argus, Zeek (formerly known as Bro), and NetFlow, originate from network observability applications. While well-established among practitioners, these tools were not originally designed for ML-driven security assessments and have only recently been adapted for such use cases [11].

Several key challenges arise from the current landscape of flow feature extraction tools:

- **Limited Compatibility with Benchmark Datasets:** Each academic dataset defines its own set of flow features, making cross-study comparisons difficult and requiring dataset-specific tools and ML pipelines [39].

- **Performance Bottlenecks:** Many tools suffer from limited throughput, high CPU and memory usage. As network sizes and traffic volumes grow, these inefficiencies can result in dropped flows, delayed real-time detection, and failures when processing large-scale traffic captures [37].
- **Limited Customization:** Some tools provide extensibility (e.g., Zeek scripts, NFStream plugins), but the flexibility and feature sets vary significantly across platforms, resulting in inconsistent customization capabilities. This inconsistency restricts their applicability for research [12].
- **Offline vs. Real-Time:** Many extractors are optimized for either offline forensic analysis of PCAP files (e.g., CICFlowMeter) or real-time traffic monitoring (e.g., Argus, nProbe). Few solutions (e.g., NFStream) provide seamless support for both, complicating research that requires processing academic benchmark datasets alongside real-world validation [40].

Table 5.1 summarizes these findings, comparing the flow extractors most used in research based on open-source availability, offline vs. real-time capabilities, customization options, and implementation language. Additionally, Go-Flows [12], a recent reference implementation of a flow exporter in Go for reproducible network research, was added to the table for comparison. The fragmentation in the ecosystem highlights the lack of a standard solution that balances the performance needs of real-world deployments with the flexibility required for academic research.

5.2.3 eBPF for Security Applications

Parallel to these challenges, eBPF has emerged as a highly efficient framework for Linux in-kernel packet analysis and filtering. Originally designed to replace the traditional Berkeley Packet Filter (BPF), eBPF allows sandboxed, event-driven programs to run at various kernel points, enabling low-overhead network packet processing [19].

The security community has quickly adopted eBPF for high-throughput applications such as microservice fingerprinting [41], inter-container communication [42], and ML-based traffic classification [43]. Major enterprises, including Netflix, leverage eBPF for performance monitoring and system profiling [44], while Cloudflare uses it for DoS mitigation and load balancing [45]. Recent projects utilize eBPF hooks to capture and aggregate packet metadata in near real-time. Bachl et al. [46] implemented a flow-based IDS entirely within eBPF, achieving a 20% performance improvement over a user-space equivalent.

Despite these advancements, no research-oriented flow feature extractors currently leverage eBPF. Most remain confined to user-space capture libraries (e.g., `libpcap`) or rely on inflexible kernel libraries unsuitable for advanced feature creation. This suggests that an eBPF-based flow extractor could bridge the performance gap while enabling deep configurability for security analytics. By operating within the kernel, such an approach could track fine-grained flow states and compute statistics with minimal context-switching overhead. Additionally, integrating eBPF with a modern systems language like Rust would ensure memory safety and efficient concurrency, making it

well-suited for large-scale or real-time monitoring.

Takeaway. Our literature review, highlights (i) the heavy reliance on custom tooling in ML-based flow-based network security research and (ii) the lack of performance, interoperability, and flexibility in existing solutions. We also found that eBPF as a next-generation in-kernel packet processing framework proves interesting potential to address the performance bottlenecks in current flow extractors.

5.3 Proposed Flow Extractor: RustiFlow

The major technical contribution of this chapter is the development of RustiFlow, a versatile research tool designed for both network analysis and practical applications. This section outlines its design goals (§5.3.1), architecture (§5.3.2), and implementation details (§5.3.3).

5.3.1 Overview and Design Goals

RustiFlow addresses the key limitations identified in prior research from Section 5.2 by focusing on three main goals:

- **High Throughput & Scalability:** The tool must handle multi-gigabit traffic efficiently, ensuring the high-speed packet processing required for modern networks. Additionally, it must process large network captures within a reasonable time with minimal resource overhead.
- **Flexible Feature Extraction:** The tool must be compatible with the most popular feature sets aligned with common benchmark datasets while empowering researchers to define additional custom features for novel experiments.
- **Real-time and Offline Operation:** The tool must support both real-time and offline processing, enabling reproducible PCAP analysis for research as well as practical validation and easy adoption in real-world deployments by practitioners.

5.3.2 Architecture and Design Choices

To achieve its design goals, RustiFlow follows a modular architecture optimized for both real-time and offline processing. It consists of three main components: configuration and user interface (§5.3.2.1), eBPF-based kernel programs (§5.3.2.2), and a user space program (§5.3.2.3).

5.3.2.1 Configurable Processing via CLI, GUI, and Config Files

RustiFlow offers flexible deployment through a command-line interface (CLI), an interactive terminal-based GUI, both supporting the use of configuration files. Users can easily adjust parameters such as flow timeouts, feature selection, and processing mode (real-time or offline).

The inclusion of a terminal-based GUI, inspired by CICFlowMeter, improves usability and adoption, even in headless terminal sessions. Unlike CICFlowMeter, RustiFlow also provides a fully functional CLI, making it well-suited for both research and production environments, including headless deployments.

5.3.2.2 eBPF Kernel Programs for Real-Time Processing

For real-time operation, RustiFlow employs four eBPF programs—two for IPv4 and two for IPv6—running on both ingress and egress traffic paths, see Figure 5.2a. These programs, integrated with Linux Traffic Control (TC) [47], extract minimal packet metadata and forward it to user space via four ring buffers. In user space, this metadata is used to compute network flow features. Because all relevant metadata is already extracted in the kernel, users can flexibly define and modify their custom flow features entirely in user space—without needing to change or recompile the eBPF code—thereby circumventing the well-known size restrictions of eBPF programs [48].

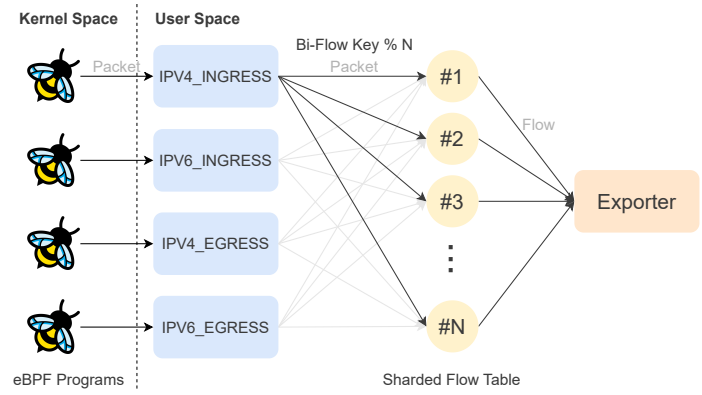
5.3.2.3 Real-Time and Offline User Space Processing

The user space program of RustiFlow handles both real-time and offline processing. (1) In real-time mode, four dedicated threads read packet metadata extracted by eBPF from their assigned ringbuffer. These first-in-first-out (FIFO) buffers ensure that packets are processed in order as they are received on the network interface. (2) In offline mode, a single thread reads the PCAP file, ensuring that packets are processed in order, and extracts the same packet metadata as the real-time mode.

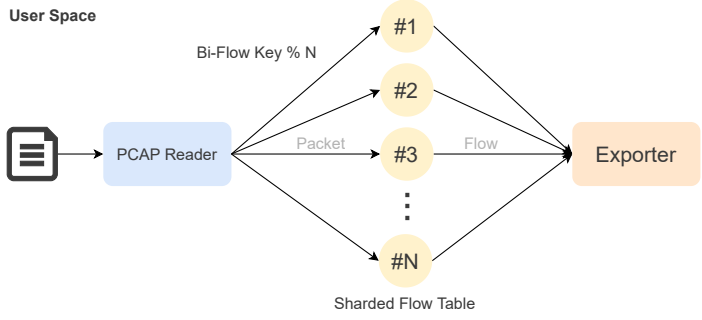
Since both approaches collect identical data, they share the subsequent components. Both modes use a sharded flow table, where a bi-directional five-tuple flow key ensures direction-invariant flow tracking. This key, when hashed, determines the flow table shard, enabling concurrent processing across multiple threads while ensuring that packets from the same flow are processed within the same shard. Each shard runs in a separate thread with a FIFO buffer ensuring in-order packet processing. Flow statistics are updated per packet and written to an output file or standard output upon flow termination or expiration. This shared processing logic guarantees consistency between real-time and offline feature extraction. Figure 5.2 presents an overview of the kernel and user space components for real-time (5.2a) and offline (5.2b) processing.

5.3.3 Implementation Details

We provide insights into RustiFlow's implementation, including our choice of Rust (§5.3.3.1), supported feature sets (§5.3.3.2), and extensive configuration options (§5.3.3.3).



(a) Realtime



(b) Offline

Figure 5.2: **Architectural overview of RustiFlow's components.** The figure illustrates the kernel and user space components involved in (a) real-time and (b) offline flow extraction. The number of processing threads (N) is configurable, allowing for scalable flow feature extraction.

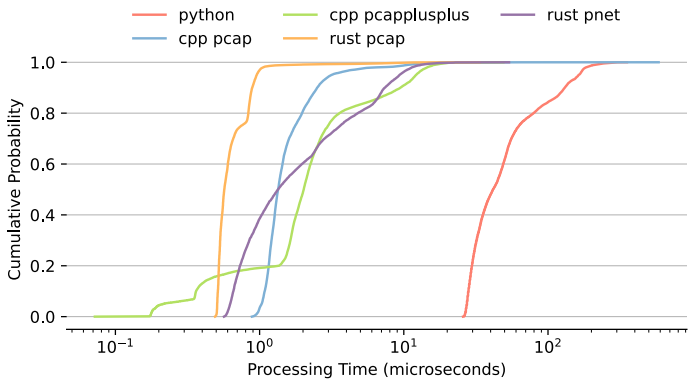


Figure 5.3: **Rust-Pcap achieves the fastest processing times.** The CDF graph compares per-packet processing times across implementations in different languages.

5.3.3.1 High-Performance Design in Rust

Rust provides memory safety, efficient concurrency, and a strong ecosystem for network applications. The Aya framework enables seamless eBPF integration, enabling lightweight in-kernel packet filtering and flow aggregation. Before selecting Rust, we benchmarked various programming languages and libraries to evaluate their performance. Figure 2 presents the results, highlighting RustiFlow's superior offline processing performance compared to Python and C++ when using libpcap for high-speed packet analysis.

5.3.3.2 Feature Sets

RustiFlow's features build upon a "BasicFlow" structure, which includes the 5-tuple flow key, timestamps, and a connection state to determine flow termination. To ensure compatibility with academic benchmarks, RustiFlow has built-in support for the feature sets from the Canadian Institute for Cybersecurity data sets (e.g., CICIDS2017), Coburg Intrusion Detection Data Sets (CIDDS 1 and 2), and NFStream's statistical flow feature set. Additionally, RustiFlow includes its own "RustiFlow" feature set, which is a superset of all the previous ones with 185 features. A detailed overview of the different feature sets and their exported features is provided in Table 5.5 in Appendix 5.B. Additionally, users can define a "Custom" feature set to export a selected subset of features or derive new flow features from packet metadata, offering flexibility for specialized use cases.

5.3.3.3 Configuration Options

RustiFlow provides extensive configurability beyond the already discussed processing mode (real-time or offline), feature set selection, and output options (file or standard output). Flow expiration behavior is adjustable by modifying the default active (1 hour) and idle (2 minutes)

Table 5.2: **Dataset overview for flow extraction evaluation.** The table summarizes the network captures used to assess flow extraction performance.

File	Num. of Packets	Size (GB)	Description
Monday	11 709 971	10.43	Benign traffic
Tuesday	11 551 954	10.66	Brute force attacks
Wednesday	13 788 878	12.96	(D)DoS attacks
Thursday	9 322 025	7.99	Web and infiltration attacks
Friday	9 997 874	8.51	Botnet, port scan and DDoS attacks
CICIDS2017	56 370 702	50.56	All days merged

timeouts. Rather than removing flows after an active or idle timeout, RustiFlow allows early export of flow statistics while retaining the flow. This enables the early processing of active flows, which is crucial for intrusion prevention systems. To reduce overfitting in ML models, RustiFlow can omit features prone to spurious correlations, such as IP addresses, timestamps, and port numbers, as identified in prior research [49]. Instead of removing port numbers, it transforms them into IANA categories ("well-known", "registered", or "dynamic"), following the approach in [50]. Performance tuning in RustiFlow can be achieved by configuring the number of flow table shards, which controls the degree of concurrent processing. Additionally, the interval for checking expired flows is adjustable: longer intervals improve performance in high-traffic environments but may delay the detection of idle flows. However, correctness is ensured by checking flow expiration on every packet before updating the flow statistics.

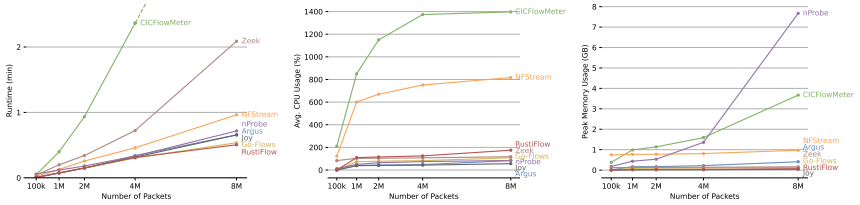
Takeaway. Our proposed network flow extractor, RustiFlow, addresses the key limitations in existing solutions by combining high performance, flexibility, and interoperability. It leverages eBPF for efficient in-kernel packet processing and supports both real-time and offline analysis. RustiFlow's modular architecture enables scalable, customizable flow feature extraction, making it a practical tool for both research and real-world deployment in network security.

5.4 Performance Evaluation

RustiFlow's performance is compared to the network flow exporters identified in Section 5.2 and summarized in Table 5.1. First, we describe the experimental setup and datasets (§5.4.1). Then, we assess flow exporter performance in offline (§5.4.2) and real-time (§5.4.3) scenarios, covering methodology, flow exporter configurations, and results. Finally, we discuss the findings (§5.4.4).

5.4.1 Experimental Setup

Our testbed consists of two bare-metal servers, each with dual 8-core Intel(R) Xeon(R) E5-2650 2.60GHz, 48GB RAM, and running Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-126-generic x86_64). The



(a) Runtime (b) CPU (c) Peak Memory

Figure 5.4: RustiFlow **shows the lowest linear increase in offline flow extraction runtime with the number of packets.** Overview of the (a) runtime, (b) CPU usage, and (c) peak memory consumption as packet count increases.

servers are connected via a dedicated 10 Gbit link and a separate control interface.

We use the raw network traffic from the CICIDS2017 dataset for evaluation in our experiments. The dataset spans five days: Monday contains only benign traffic, while each subsequent day introduces specific attack types. To facilitate large-scale testing, we merge all days into a single 50GB pcap file containing over 56 million packets. Table 5.2 summarizes the PCAP files, including their size, number of packets, and content.

To evaluate the performance in the offline and real-time experiments, performance and resource metrics are collected. Offline performance is measured by processing time, while real-time performance is evaluated via sustained throughput. The throughput is measured on the interface using `ifstat` [51] and configured using the throughput option of `tcpreplay` [52] when replaying the raw network traffic. CPU and memory usage are monitored with `psutil` [53], a cross-platform Python library for process and system monitoring.

5.4.2 Offline Traffic Analysis

5.4.2.1 Methodology

We evaluate offline traffic analysis by measuring the processing time for network captures with increasing packet counts, from 100K to 8M packets. CPU and memory usage are recorded at one-second intervals. To ensure statistical relevance, we select the first n packets from each day's capture, thus repeating the experiment five times for each desired number of packets. Finally, to evaluate the performance on large traffic captures, the flow exporters process the merged pcap with all days; this is repeated again five times for statistical relevance. Flow exporters failing to complete within one hour were terminated and considered unsuccessful.

5.4.2.2 Flow exporter configuration

All nine exporters in Table 5.1 support offline processing. However, Kitsune [26] required a custom Python script to extract the features from a file and write output to a CSV file. To ensure a fair

comparison despite variations in feature sets and flow logic between the different tools, RustiFlow is evaluated using both a minimal ("Basic") configuration with 10 exported features and a full-feature ("RustiFlow") configuration with 185 features. When configurable, we set idle and active timeouts to 2 minutes and 1 hour, respectively, while keeping all other settings at their defaults. Appendix 5.A provides a detailed overview of the specific version used for each tool in the experiments, including any custom configurations (if applicable) and information about the exported features.

5.4.2.3 Results

We conducted ten sets of experiments: two with RustiFlow ("Basic" and "RustiFlow") and eight with the existing flow exporters. Kitsune and CICFlowMeter failed to process all the PCAP files. Kitsune managed only 1 million packets before exceeding the one-hour limit. CICFlowMeter processed up to 8M packets but crashed due to an out-of-memory exception on the largest PCAP. All other flow exporters completed processing within the given constraints. Figure 5.4 plots the average runtime (5.4a), CPU (5.4b), and peak memory usage (5.4c) as functions of packet count. Most exporters show a linear runtime increase, except CICFlowMeter and Zeek. CICFlowMeter and Zeek, along with Kitsune—which failed to process larger PCAPs—are unsuitable for offline analysis of large traffic captures. NFStream and CICFlowMeter showed the highest CPU usage, requiring 10 and 14 cores, respectively, compared to 1–2.5 for other exporters. Similarly, nProbe, and CICFlowMeter showed significantly higher peak memory consumption, making them inefficient for large PCAP files.

The best-performing offline exporters are RustiFlow, Go-Flows, Joy, and Argus. RustiFlow ("Basic") has the lowest runtime and memory usage, closely followed by Go-Flows and RustiFlow ("RustiFlow"); see left bottom corner in Figure 5.5. Joy has the lowest CPU overhead but a comparatively longer runtime, making it less suitable for offline processing. Table 5.3 provides a detailed breakdown of performance and resource metrics, averaged over five runs including standard deviations.

Table 5.3: **Detailed performance overview of the evaluated flow extractors for offline traffic analysis.** The table presents the mean and standard deviation of the runtime, average CPU usage, and maximum memory usage for PCAP files with an increasing number of packets. The lowest value for each experiment is marked in bold.

Extractor	#Packets	Runtime (s)		Avg CPU Usage (%)		Max Memory Usage (MB)	
	(in millions)	Mean	Std	Mean	Std	Mean	Std
RustiFlow ("Basic")	0.1	0.3	0.1	0.0	0.0	6.6	0.3
	0.5	1.8	0.1	78.1	9.8	8.3	0.8
	1.0	3.8	0.5	103.3	12.6	9.6	0.9
	2.0	7.9	0.9	112.0	12.8	10.0	0.5
	4.0	16.8	1.1	115.1	3.9	10.4	0.5
	8.0	28.2	2.5	148.2	10.5	11.7	0.4

Continued on next page

Extractor	#Packets	Runtime (s)		Avg CPU Usage (%)		Max Memory Usage (MB)	
	(in millions)	Mean	Std	Mean	Std	Mean	Std
	56.4	117.3	3.2	246.3	7.2	20.5	5.2
RustiFlow ("RustiFlow")	0.1	0.3	0.1	0.0	0.0	7.9	0.6
	0.5	2.1	0.2	95.9	14.8	21.0	4.2
	1.0	4.4	0.3	109.4	16.9	23.6	5.4
	2.0	9.0	0.5	115.2	9.3	30.7	5.0
	4.0	18.7	1.2	124.5	8.6	37.1	7.0
	8.0	30.3	3.3	174.9	18.2	69.5	14.1
	56.4	147.0	12.4	261.1	20.7	192.4	6.1
Argus	0.1	0.7	0.1	0.0	0.0	8.8	11.4
	0.5	2.4	0.7	14.5	9.4	136.3	52.8
	1.0	4.5	1.4	40.7	10.9	169.4	30.6
	2.0	8.8	2.4	43.7	8.8	169.9	14.5
	4.0	20.4	0.5	48.1	8.1	220.4	31.9
	8.0	39.4	2.7	82.7	20.0	409.6	209.3
	56.4	189.8	7.1	137.2	6.2	1755.1	69.4
CICFlowMeter*	0.1	2.9	0.3	209.7	8.1	381.6	28.6
	0.5	10.8	0.7	520.6	102.9	856.3	151.5
	1.0	23.9	1.2	849.5	37.2	989.2	192.1
	2.0	56.0	2.1	1149.6	22.6	1136.6	173.1
	4.0	141.9	5.7	1373.6	34.7	1590.5	311.4
	8.0	313.4	28.7	1397.3	118.1	3671.6	1006.6
Go-Flows	0.1	0.3	0.1	0.0	0.0	4.0	0.1
	0.5	1.9	0.3	62.2	8.7	67.7	6.3
	1.0	4.3	0.4	73.4	7.9	84.3	5.7
	2.0	8.8	0.5	77.7	6.6	95.9	10.9
	4.0	18.2	1.2	82.6	6.7	106.1	12.5
	8.0	32.2	3.5	107.9	20.6	134.3	32.4
	56.4	133.4	22.6	209.8	39.0	215.4	1.6
Joy	0.1	0.4	0.1	0.0	0.0	4.8	0.5
	0.5	2.1	0.2	29.3	3.5	14.7	4.7
	1.0	4.6	0.2	38.2	8.6	17.5	4.3
	2.0	9.1	0.4	40.4	4.7	18.6	3.5
	4.0	19.0	0.7	43.1	4.5	19.2	3.3
	8.0	39.4	3.2	55.9	5.5	43.2	32.8
	56.4	194.5	18.9	80.5	5.5	99.1	1.3
NFStream	0.1	1.5	0.3	123.4	10.8	746.9	5.4
	0.5	4.2	0.1	494.2	28.6	764.4	18.2
	1.0	7.8	0.8	600.7	33.5	768.7	17.4
	2.0	15.4	1.3	669.5	68.8	775.6	15.8
	4.0	27.5	0.6	751.1	13.4	808.9	28.1
	8.0	57.8	8.5	816.9	52.9	973.4	177.7
	56.4	364.4	12.8	958.8	34.9	1207.6	35.0
nProbe	0.1	3.5	0.1	12.2	2.1	178.7	84.2
	0.5	5.2	0.4	37.6	3.4	379.3	260.0
	1.0	7.1	0.4	49.4	3.1	436.3	296.7
	2.0	10.9	0.3	64.1	3.8	534.3	283.4
	4.0	19.9	1.0	74.1	5.8	1359.7	698.1

Continued on next page

Extractor	#Packets	Runtime (s)		Avg CPU Usage (%)		Max Memory Usage (MB)	
	(in millions)	Mean	Std	Mean	Std	Mean	Std
	8.0	43.0	5.1	84.4	2.1	7673.1	3986.9
	56.4	248.5	2.2	87.5	2.1	20 169.4	12.6
	0.1	2.6	0.7	83.2	15.9	120.4	7.5
	0.5	7.7	2.1	102.1	8.7	125.8	9.0
	1.0	11.9	2.4	103.2	5.5	130.5	9.2
Zeek	2.0	20.4	2.5	104.0	3.2	133.6	6.8
	4.0	43.4	6.5	107.5	4.8	139.4	5.3
	8.0	125.3	16.2	117.7	5.6	163.0	25.1
	56.4	1000.2	14.9	125.0	0.7	219.6	0.7
	0.1	191.1	79.8	100.2	0.3	308.9	23.5
Kitsune*	0.5	1331.1	721.4	99.5	0.7	812.0	80.1
	1.0	2610.8	1230.1	99.4	0.7	1503.7	92.4

*Some runs did not finish within 1 hour or went out-of-memory

5.4.3 Real-time Traffic Analysis

5.4.3.1 Methodology

We evaluate real-time analysis by measuring the performance and resource usage of flow exporters monitoring a live network interface. Our testbed consists of two hosts connected via a dedicated 10 Gbit link. The first host replays the merged CICIDS2017 pcap using `tcpreplay`, while the second host captures the incoming traffic. The flow exporter deployed on the second host to monitor the receiving interface, with CPU and memory usage recorded at one-second intervals—consistent with the offline experiment. To assess performance under load, we configure `tcpreplay` to replay traffic at increasing throughput levels, starting at 1 Mbps and scaling up to 10 Gbps in order-of-magnitude steps. Each run lasts five minutes or until the full pcap is replayed. Before playback, the pcap is preprocessed using tcprewrite [54] to truncate packets exceeding the network's maximum transmission unit and modify source and destination MAC addresses for correct packet delivery.

Aside from evaluating resource consumption, we assess the exporters' ability to process traffic under load. However, directly comparing flow outputs across exporters is challenging due to differences in their flow processing logic. For instance, Argus maintains a full state machine to match ICMP responses with their corresponding requests, whereas CICFlowMeter aggregates all ICMP traffic into a single flow. This prevents fair comparison based on flow output alone. Instead, we leverage internal performance metrics—such as the number of processed and/or dropped packets—available only for Joy, nProbe, NFStream, and RustiFlow. Since other benchmarked flow exporters do not expose such metrics, they are excluded from this analysis. Simultaniously, we monitor the total number of packets received on the host using tcpdump. Together, these metrics enable the analysis of packet loss rates for these flow exporters in high-throughput conditions.

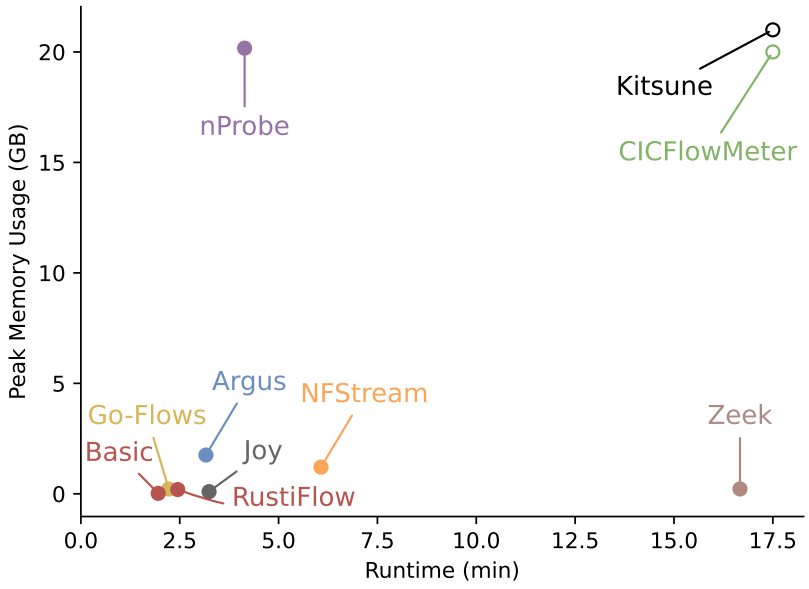


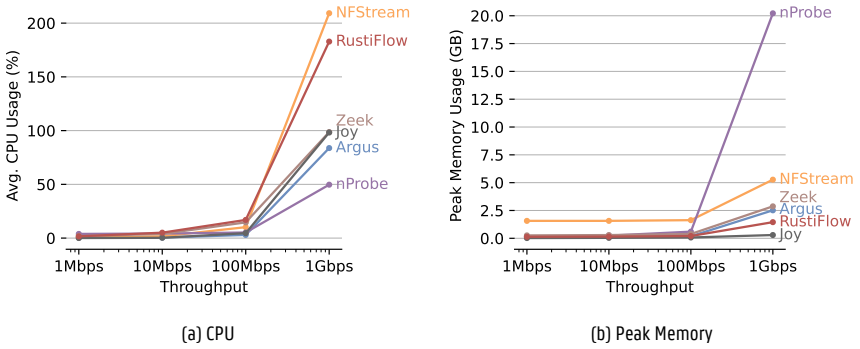
Figure 5.5: RustiFlow **shows minimal runtime and memory usage for offline flow extraction.** Overview of the peak memory usage and runtime for flow extraction on all CICIDS2017 network traffic merged (56.4M packets).

5.4.3.2 Flow Exporter Configuration

Only six of the nine flow exporters listed in Table 5.1 support real-time traffic analysis in headless mode. As in the offline evaluation, RustiFlow is evaluated using both a minimal ("Basic") configuration with 10 exported features and a full-feature ("RustiFlow") configuration with 185 features to ensure a fair comparison. The same timeout settings are also applied across all tools, with idle and active timeouts set to 2 minutes and 1 hour, respectively, alongside tool-specific configurations as detailed in Appendix 5.A. This results in a total of seven experimental configurations.

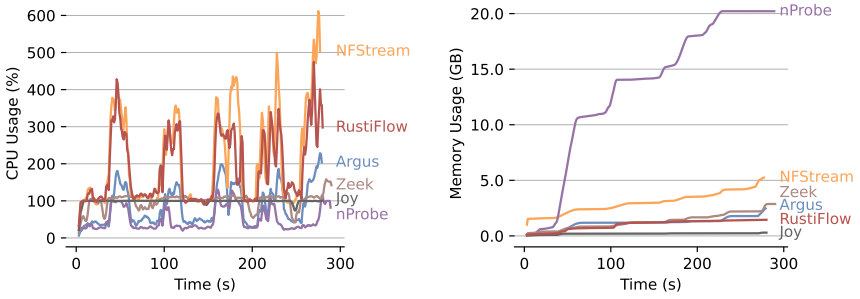
5.4.3.3 Results

Figure 5.6 presents the CPU (5.6a) and memory (5.6b) usage of the evaluated flow exporters as a function of the sustained throughput. nProbe demonstrates the lowest CPU usage but with the highest memory consumption—four times higher than Argus, the second-highest. Joy maintains the smallest memory footprint, closely followed by RustiFlow ("Basic"), while RustiFlow ("RustiFlow") has a comparable memory usage to Zeek and Argus. Up to 100 Mbps, CPU usage remains below 20% for all flow exporters. However, beyond this threshold, it increases quickly, peaking at 210% for NFStream, 183% for RustiFlow ("RustiFlow"), and 138% for RustiFlow ("Basic") at 1 Gbps. Zeek, Joy, and Argus sustain CPU usage below 100% at this level, while nProbe



(a) CPU (b) Peak Memory

Figure 5.6: Overview of (a) CPU usage and (b) peak memory consumption for real-time flow extraction across different throughput levels, ranging from 1Mbps to 1Gbps.



(a) CPU (b) Memory

Figure 5.7: Progress of the (a) CPU usage and (b) memory consumption for real-time flow extraction for the evaluated flow exporters at 1Gbps throughput.

records the lowest usage at 50%. Table 5.4 provides detailed values for average CPU as well as average and maximum memory consumption for all the flow exporters across the different throughput levels. It is important to note that although `tcpreplay` was configured for 10Gbps, it only achieved a maximum throughput of 1.8Gbps.

Figure 5.7 provides a detailed view of CPU (5.7a) and memory (5.7b) usage during the 1Gbps experiment. CPU consumption displays five peaks, corresponding to the days in the replayed CICIDS2017 PCAP. Since flow exporters typically process packet headers rather than payload, an increase in packet rate—despite constant throughput—results in a higher computational load. This leads to significant CPU spikes, with peak usage reaching up to six times the baseline. Memory consumption trends show that Joy maintains the most stable and smallest footprint, followed by RustiFlow. In contrast, Argus, Zeek, NFStream, and nProbe display steadily increasing memory usage under sustained 1 Gbps conditions.

Table 5.4: **Detailed performance overview of the evaluated flow extractors for real-time traffic analysis.** The table presents the average CPU usage and average and maximum memory usage under increasing load. Note that `tcpreplay` only achieved 1.8Gbps throughput when configured for 10Gbps. The best (lowest) value for each metric and throughput combination is marked in bold.

Extractor	Throughput (Mbps)	Avg CPU Usage (%)	Avg Memory Usage (MB)	Max Memory Usage (MB)
RustiFlow ("Basic")	1	1.5	122.8	124.0
	10	4.0	124.9	125.8
	100	15.3	126.9	129.8
	1000	137.9	250.0	328.6
	10 000	196.2	275.2	420.8
RustiFlow ("RustiFlow")	1	1.6	129.1	136.0
	10	5.1	147.1	151.6
	100	16.9	175.6	204.1
	1000	182.8	1011.7	1450.6
	10 000	232.5	1293.3	2340.3
Argus	1	0.2	150.0	154.6
	10	0.5	159.3	163.3
	100	**2.9**	167.4	180.2
	1000	83.8	1138.3	2515.7
	10 000	135.5	1361.4	2860.7
Joy	1	**0.1**	18.0	**21.6**
	10	**0.3**	**26.7**	**40.2**
	100	4.6	**58.4**	**72.2**
	1000	98.2	**193.7**	**293.7**
	10 000	98.2	**195.7**	**289.9**
NFStream	1	0.6	1544.9	1566.0
	10	1.6	1553.6	1567.5
	100	10.2	1581.9	1629.4
	1000	209.2	2987.3	5268.6
	10 000	324.4	3079.2	5229.1
nProbe	1	3.9	97.0	137.2
	10	4.2	208.6	243.0
	100	5.2	395.9	597.6
	1000	**49.7**	13 415.0	20 217.8
	10 000	**62.7**	11 889.6	17 752.2
Zeek	1	1.9	250.6	261.3
	10	3.9	276.1	290.2
	100	14.6	320.0	385.1
	1000	98.6	1303.8	2871.2
	10 000	108.3	1356.9	2689.0

To evaluate the robustness of the flow exporters under high throughput, we analyzed packet loss for RustiFlow, NFStream, nProbe, and Joy. RustiFlow ("Basic") demonstrated superior performance, successfully processing all packets without any losses, even at 1.8 Gbps (the maximum obtained throughput by `tcpreplay` in our setup). In contrast, RustiFlow ("RustiFlow") encountered minor packet loss at higher throughputs, dropping 1.1M packets (2.0%) at 1 Gbps and 0.7M packets (1.4%) at maximum throughput. The lower drop rate at maximum throughput is explained by a reduced number of packets arriving at the receiving host, despite the same total number of packets (56.4M) being replayed at each rate. At 1 Gbps, 55.9M packets were received, whereas at maximum throughput, only 48.6M arrived, reducing the opportunity for RustiFlow to drop packets. Despite packets being dropped, its loss rate remained significantly lower than that of NFStream, which failed to process 3.0% of packets at 1 Gbps and demonstrated a substantial increase to 12.7% at maximum throughput. nProbe recorded a higher packet loss rate with 6.9% of packets dropped at 1 Gbps and an even greater increase to 16.8% at maximum throughput. Joy had the highest dropped packet rate among the tested exporters with 72.6% and 90%, respectively, for 1 Gbps and at maximum throughput. There were no packets dropped by the exporters at rates below 1 Gbps. These results emphasize the efficiency of RustiFlow in handling high packet rates, with RustiFlow ("Basic") maintaining lossless performance and RustiFlow ("RustiFlow") sustaining only minimal losses. In comparison, NFStream, nProbe, and Joy show considerable degradation at higher throughputs, making them less reliable under heavy network loads.

5.4.4 Discussion

The evaluation results demonstrate that RustiFlow successfully meets its design goals by balancing high throughput, flexible network flow feature extraction, and real-time and offline processing capabilities.

First, RustiFlow demonstrates high throughput and scalability, efficiently handling multi-gigabit traffic while maintaining minimal resource overhead. The offline evaluation shows that RustiFlow ("Basic") achieves the fastest processing with the lowest memory consumption, even when exporting significantly more features using the "RustiFlow" feature set. This makes RustiFlow well-suited offline analysis of large network captures. In real-time settings, RustiFlow ("Basic") maintained a zero packet loss, even at high throughputs, outperforming NFStream, Joy, and nProbe, which suffered significant packet drops beyond 1 Gbps. RustiFlow ("RustiFlow") maintains strong performance with only minimal packet loss under extreme conditions, reinforcing its suitability for high-speed network monitoring.

Second, RustiFlow achieves flexible feature extraction while maintaining strong performance. The ability to support both widely used benchmark feature sets and user-defined custom features makes RustiFlow a powerful tool for researchers and practitioners. Remarkably, even while extracting more than twice as many features as the competing tools, RustiFlow remains highly competitive in terms of runtime and resource efficiency. Its well-balanced design ensures extensibility with minimal computational overhead, allowing customized traffic analysis without

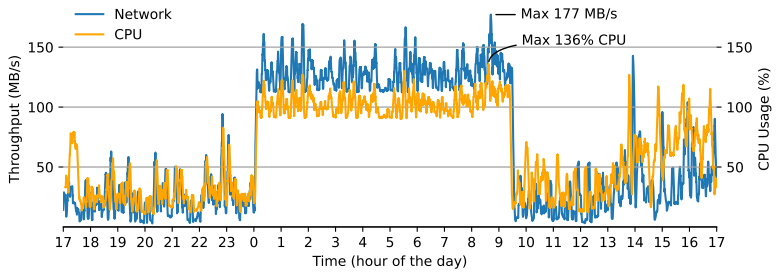


Figure 5.8: **Stable performance of RustiFlow in 24-hour data center monitoring.** The throughput and CPU usage measured over a full day in our research group's data center, visualized using a 5-minute sliding window.

sacrificing performance.

Lastly, RustiFlow successfully enables both real-time and offline processing, offering fast, low-resource PCAP-based analysis alongside robust real-time traffic monitoring. Leveraging eBPF, it achieves the lowest real-time packet loss among all evaluated exporters. Additionally, RustiFlow remains lightweight, outperforming tools like nProbe and NFStream, which display significantly higher memory usage under sustained traffic loads.

Takeaway. RustiFlow meets its design goals, delivering the fastest offline PCAP processing and achieving the lowest real-time packet loss under high loads by leveraging eBPF. It maintains a minimal resource footprint with competitive overhead while supporting widely used and custom feature sets, making it a flexible and efficient tool for research and real-world deployments.

5.5 Real-World Case Studies

This section demonstrates RustiFlow's stability, reliability, and performance in real-world deployments. We validate its performance in two distinct environments:

- **A university data center uplink**, where RustiFlow operates at scale under diverse workloads, monitoring the data center uplink.

- **A controlled adversarial network security testbed**, where RustiFlow monitors 50 hosts during a hands-on network security lab.

5.5.1 University Data Center

5.5.1.1 Environment

Our research group data center is designed for large-scale networking and cloud experiments. It hosts hundreds of servers, hardware switches, and high-performance GPUs used by researchers, external projects, and students. The diversity of applications results in highly heterogeneous network traffic patterns.

5.5.1.2 Deployment

RustiFlow was deployed by mirroring all uplink traffic over a 10Gbps connection. The "CIC" feature set was used for flow extraction, while `ifstat` monitored bandwidth usage at five-second intervals and `tcpdump` recorded packet counts on the interface. The default RustiFlow configuration was applied, with an active timeout of 1 hour, an idle timeout of 2 minutes, and a flow expiration check interval of 1 minute. All extracted flow data was exported to a CSV file.

5.5.1.3 Insights

Over a 24-hour period, RustiFlow operated without errors or packet loss, successfully processing 1.1 billion packets, corresponding to 5.8TB of total traffic. Resource usage remained stable, with an average CPU load of 64.5% and memory consumption of 156.7 MB, peaking at 175.0 MB. Throughput fluctuated significantly, with a peak of 395.4 MB/s over a five-second window, an average throughput of 67.3 MB/s, and a minimum of 1.6 MB/s. A clear daily traffic pattern emerged, as illustrated in Figure 5.8, with background jobs running at night. Peak activity was observed in the morning, where the highest recorded CPU usage reached 136% and the maximum throughput peaked at 177 MB/s over a five-minute rolling window. The successful deployment in this real-world environment confirms RustiFlow's reliability and scalability for large-scale network monitoring.

5.5.2 Network Security Testbed

5.5.2.1 Environment

The second case study took place in a network security lab with approximately 50 master's students enrolled in a Network Security course. Each student accessed a personal isolated private network with various services—including web, media, mail, and DNS—via a VPN connection. The lab was structured into three phases. First, students conducted network discovery using tools like `nmap` and `ping`. They then performed service enumeration with `nmap`, `ssh`, `ftp`, and `telnet`, before moving on to vulnerability scanning, utilizing `scapy`, `dirbuster`, and similar tools. Although the lab session was scheduled for the afternoon, RustiFlow was deployed from 10 AM to 11 PM, covering a 13-hour monitoring period. However, since attendance was not mandatory and the lab could be submitted later, some students completed their tasks outside of this timeframe, leading the inactivity of some hosts.

5.5.2.2 Deployment

RustiFlow was deployed across 50 hosts, monitoring the VPN network traffic and exporting the flows using the "CIC" feature set to a CSV file. `iftop` logged bandwidth statistics at ten-second intervals and `tcpdump` captured packets on the monitored interface. The same default RustiFlow configuration was used, with an active timeout of 1 hour, an idle timeout of 2 minutes, and a flow expiration check interval of 1 minute.

5.5.2.3 Insights

Among the 50 monitored hosts, 30 were actively engaged, determined by processing at least 1MB of traffic. The most active host received 488 MB of data, with an average of 11 MB per active host. The highest observed throughput reached 1.7 Mbps. Resource consumption remained negligible throughout the experiment. Average CPU usage was 0.01%, peaking at 0.15%, while memory usage remained stable at an average of 158 MB, with a peak of 159 MB. Traffic spikes were observed during two specific tasks. Web directory enumeration using `dirbuster` resulted in a significant increase in packet counts, while successfully uncovering a hidden RTSP stream led to a rise in throughput as students streamed the video data. Despite the adversarial nature of the lab, RustiFlow operated without errors or packet drops on any of the 50 monitored hosts, demonstrating its robustness in traffic monitoring under controlled attack scenarios.

Takeaway. RustiFlow demonstrates stability, reliability, and efficiency in monitoring large-scale and adversarial environments. It processes billions of packets with minimal resource usage, zero packet loss, and no errors, proving its application for real-time deployments.

5.6 Conclusion

In this chapter, we introduced RustiFlow, an open-source, high-performance network flow feature extractor built on eBPF and implemented in Rust. Addressing the critical gap between research-oriented and production-ready flow extraction tools, RustiFlow provides a scalable, flexible, and efficient solution for both security research and real-world operational deployments.

Our evaluation demonstrated that RustiFlow outperforms well-known flow extractors such as NFStream, nProbe, and CICFlowMeter in both real-time monitoring and forensic PCAP analysis. By leveraging eBPF for in-kernel packet processing, RustiFlow achieves minimal resource overhead while maintaining high throughput and stability, even under high-load scenarios. It delivers the fastest offline PCAP processing and achieves the lowest real-time packet loss, ensuring reliable performance in large-scale monitoring environments. The modular architecture supports widely used and custom feature sets, ensuring adaptability across various research and security applications. Furthermore, our real-world case studies validated RustiFlow's robustness in both a university data center and a network security test lab, proving its practicality for large-scale deployments.

Beyond performance improvements, our literature review highlighted the limitations of existing extractors in terms of interoperability and adaptability, particularly for ML-based security applications. RustiFlow directly addresses these challenges by supporting both widely used academic feature sets and customizable flow features, bridging the gap between researchers and practitioners. RustiFlow has the potential to become an essential tool for ML-based network security, driving both academic research and real-world adoption.

Acknowledgment

This work is supported by the Belgian Chancellery of the Prime Minister (Grant: AIDE-BOSA).

Bibliography

[1] J. P. Anderson, "Computer security threat monitoring and surveillance," *Technical Report, James P. Anderson Company*, 1980.

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

[3] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, pp. 243–256, 2007.

[4] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *International workshop on recent advances in intrusion detection*. Springer, 2006, pp. 226–248.

[5] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 251–261.

[6] D. Chou and M. Jiang, "A survey on data-driven network intrusion detection," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–36, 2021.

[7] M. De Shon, "Information Security Analysis as Data Fusion," in *IEEE Int. Conf. Inf. Fusion*, 2019.

[8] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of {SOC} analysts' perspectives on security alarms," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2783–2800.

[9] C. Crowley and J. Pescatore, "A sans 2021 survey: security operations center (soc)," *SANS, NY, Tech. Rep.*, pp. 1–22, 2021.

[10] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: a survey of case studies," *ACM computing surveys*, vol. 55, no. 6, pp. 1–29, 2022.

[11] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2018.

[12] G. Vormayr, J. Fabini, and T. Zseby, "Why are my flows different? a tutorial on flow exporters," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2064–2103, 2020.

[13] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "Netflow datasets for machine learning-based network intrusion detection systems," in *Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings 10*. Springer, 2021, pp. 117–135.

[14] G. Apruzzese, L. Pajola, and M. Conti, "The cross-evaluation of machine learning-based network intrusion detection systems," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 5152–5169, 2022.

[15] M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Towards model generalization for intrusion detection: Unsupervised machine learning techniques," *Journal of Network and Systems Management*, vol. 30, pp. 1–25, 2022.

[16] M. Sarhan, S. Layeghy, and M. Portmann, "Towards a standard feature set for network intrusion detection system datasets," *Mobile networks and applications*, pp. 1–14, 2022.

[17] G. Apruzzese, P. Laskov, E. Montes de Oca, W. Mallouli, L. Brdalo Rapa, A. V. Grammatopoulos, and F. Di Franco, "The role of machine learning in cybersecurity," *Digital Threats: Research and Practice*, vol. 4, no. 1, pp. 1–38, 2023.

[18] G. Apruzzese, P. Laskov, and J. Schneider, "Sok: Pragmatic assessment of machine learning for network intrusion detection," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2023.

[19] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[20] K. Dietz, M. Mühlhauser, J. Kögel, S. Schwinger, M. Sichermann, M. Seufert, D. Herrmann, and T. Hoßfeld, "The missing link in network intrusion detection: Taking ai/ml research efforts to users," *IEEE Access*, 2024.

[21] L. Deri and N. SpA, "nprobe: an open source netflow probe for gigabit networks," in *TERENA Networking Conferenc*, 2003.

[22] "Argus sensor repository," https://github.com/openargus/argus.

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

[24] "The zeek network security monitor," https://github.com/zeek/zeek.

[25] "A package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring," https://github.com/cisco/joy.

[26] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *NDSS*, 2018.

[27] "A network intrusion detection system based on incremental statistics (afterimage) and an ensemble of autoencoders (kitnet)," https://github.com/ymirsky/Kitsune-py.

[28] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 7–12.

[29] "Fixed version of the cicflowmeter tool," https://github.com/GintsEngelen/CICFlowMeter.

[30] "Go-flows: Flow exporter implementation in go," https://github.com/CN-TU/go-flows.

[31] Z. Aouini and A. Pekar, "Nfstream: A flexible network data analysis framework," *Computer Networks*, vol. 204, p. 108719, 2022.

[32] "Nfstream: a flexible network data analysis framework," https://github.com/nfstream/nfstream.

[33] "Rustiflow repository," https://github.com/idlab-discover/RustiFlow, 2025.

[34] I. Pekaric and G. Apruzzese, ""We provide our resources in a dedicated repository": Surveying the Transparency of HICSS publications," in *Proc. Hawaii International Conference on System Sciences (HICSS)*, 2025.

[35] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, 2018.

[36] "Cicflowmeter v4 repository," https://github.com/ahlashkari/CICFlowMeter.

[37] R. Flood and D. Aspinall, "Measuring the complexity of benchmark nids datasets via spectral analysis," in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2024, pp. 335–341.

[38] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, "Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018," in *2022 IEEE Conference on Communications and Network Security (CNS)*, 2022, pp. 254–262.

[39] M. Sarhan, S. Layeghy, and M. Portmann, "Evaluating standard feature sets towards increased generalisability and explainability of ml-based network intrusion detection," *Big Data Research*, 2022.

[40] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *USENIX Security*, 2022.

[41] H. Chang, M. Kodialam, T. Lakshman, and S. Mukherjee, "Microservice fingerprinting and classification using machine learning," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019, pp. 1–11.

[42] J. Nam, S. Lee, P. Porras, V. Yegneswaran, and S. Shin, "Secure inter-container communications using xdp/ebpf," *IEEE/ACM Transactions on Networking*, vol. 31, no. 2, pp. 934–947, 2022.

[43] J. Gallego-Madrid, I. Bru-Santa, A. Ruiz-Rodenas, R. Sanchez-Iborra, and A. Skarmeta, "Machine learning-powered traffic processing in commodity hardware with ebpf," *Computer Networks*, vol. 243, p. 110295, 2024.

[44] J. Koch, M. Spier, B. Gregg, and E. Hunter, "Extending vector with ebpf to inspect host and container performance," https://medium.com/netflix-techblog/extending-vector-with-ebpf-to-inspect-host-and-container-performance-5da3af4c584b, 2019.

[45] M. Majkowski, "Cloudflare architecture and how bpf eats the world," Cloudflare Blog, 2019. [Online]. Available: https://blog.cloudflare.com/cloudflare-architecture-and-how-bpf-eats-the-world/

[46] M. Bachl, J. Fabini, and T. Zseby, "A flow-based ids using machine learning in ebpf," *arXiv preprint arXiv:2102.09980*, 2021.

[47] A. N. Kuznetsov, "tc(8) — linux manual." accessed on 28 May 2023. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html.

[48] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance implications of packet filtering with linux ebpf," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 1. IEEE, 2018, pp. 209–217.

[49] L. D'hooge, M. Verkerken, B. Volckaert, T. Wauters, and F. De Turck, "Establishing the contaminating effect of metadata feature inclusion in machine-learned network intrusion detection models," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2022, pp. 23–41.

[50] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi, and M. Colajanni, "Deep reinforcement adversarial learning against botnet evasion attacks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 1975–1987, 2020.

[51] "ifstat man page: handy utility to read network interface statistics," https://man7.org/linux/man-pages/man8/ifstat.8.html.

[52] "Tcpreplay repository," https://github.com/appneta/tcpreplay.

[53] "psutil: Cross-platform lib for process and system monitoring in python," https://pypi.org/project/psutil/.

[54] "Tcprewrite wiki page," https://tcpreplay.appneta.com/wiki/tcprewrite.

[55] "Argus clients program repository," https://github.com/openargus/clients.

Appendix

5.A Network Flow Exporter Details

This appendix provides more details regarding the network flow extraction tools evaluated in this chapter. For each tool, the used version, specific configuration (if any), and the number of exported features are described.

5.A.1 RustiFlow

The experiments use the *EuroS&P Workshop (WTMC)* release on GitHub [33]. Two feature set configurations are evaluated: the "Basic" and "RustiFlow" set, respectively, exporting 10 and 185 features. See Table 5.5 in Appendix 5.B for a detailed overview of the exported features.

5.A.2 Argus

The combination of the Argus network sensor and the `ra` client program (both version 5.0.2) is used for evaluation. The network sensor and client are downloaded from GitHub, corresponding to commits `b1312f5`[22] and `540ff79`[55], respectively. The Argus sensor is used to either monitor a live network interface or process a pcap file, piping the output to `ra`, which writes 75 features delimited with a comma to an output file. Listing 5.1 shows the Argus command used for offline traffic processing, along with an overview of the 75 exported features.

Listing 5.1: Argus command for offline traffic processing

```
argus -r "$PCAP_FILE" -S 3600 -w - | ra -r - -c , -s stime ltime
    trans flgs dur runtime idle mean stddev sum min max smacclass
    dmacclass senc denc saddr daddr proto sport dport stos dtos sdsb
     ddsb sttl dttl shops dhops sipid dipid autoid cause pkts spkts
    dpkts bytes sbytes dbytes appbytes sappbytes dappbytes pcr load
    sload dload loss sloss dloss ploss psloss pdloss retrans
    sretrans dretrans pretrans psretrans pdretrans sgap dgap rate
    srate drate dir state swin dwin stcpb dtcpb smss dmss tcprtt
    synack ackdat tcpopt inode offset smeansz dmeansz > "$OUTPUT_CSV
    "
```

5.A.3 Zeek

Zeek (formerly known as Bro), version 7.0.3, is installed using Ubuntu's advanced packaging tool (`apt`). Zeek differs from the other evaluated flow extractors in that it does not extract statistical features from network flows. Instead, it generates a wide range of logs based on the observed network traffic. While the documentation states that over 70 log files are supported by default, only 27 log files were produced during the experiments.

5.A.4 nProbe

nProbe version v.10.6.241202, with native `PF_RING` acceleration, is installed via Ubuntu's `apt` package manager. An educational license (nProbe Enterprise M) is used for the experiments. nProbe is configured with a custom template to export 43 network flow features. The command used for offline traffic processing, along with an overview of the exported features, is shown in Listing 5.2.

Listing 5.2: nProbe command for offline traffic processing

```
nprobe -i {pcap_file}.pcap -n none -t 3600 -d 120 --csv-separator ,
    -P {output_folder} -V 9  -T '%IPV4_SRC_ADDR %IPV4_DST_ADDR %
    L4_SRC_PORT %L4_DST_PORT %PROTOCOL %L7_PROTO %IN_BYTES %
    OUT_BYTES %IN_PKTS %OUT_PKTS %FLOW_DURATION_MILLISECONDS %
    TCP_FLAGS %CLIENT_TCP_FLAGS %SERVER_TCP_FLAGS %DURATION_IN %
    DURATION_OUT %MIN_TTL %MAX_TTL %LONGEST_FLOW_PKT %
    SHORTEST_FLOW_PKT %MIN_IP_PKT_LEN %MAX_IP_PKT_LEN %
    SRC_TO_DST_SECOND_BYTES %DST_TO_SRC_SECOND_BYTES %
    RETRANSMITTED_IN_BYTES %RETRANSMITTED_IN_PKTS %
    RETRANSMITTED_OUT_BYTES %RETRANSMITTED_OUT_PKTS %
    SRC_TO_DST_AVG_THROUGHPUT %DST_TO_SRC_AVG_THROUGHPUT %
    NUM_PKTS_UP_TO_128_BYTES %NUM_PKTS_128_TO_256_BYTES %
    NUM_PKTS_256_TO_512_BYTES %NUM_PKTS_512_TO_1024_BYTES %
    NUM_PKTS_1024_TO_1514_BYTES %TCP_WIN_MAX_IN %TCP_WIN_MAX_OUT %
    ICMP_TYPE %ICMP_IPV4_TYPE %DNS_QUERY_ID %DNS_QUERY_TYPE %
    DNS_TTL_ANSWER %FTP_COMMAND_RET_CODE'
```

5.A.5 Joy

Joy version 4.5.0 is downloaded from GitHub at commit `2177051` [25] and built following the instructions provided in the repository. The tool is configured to export bidirectional network flow features (`bidir=1`) and outputs one JSON object per flow. The number of exported features varies depending on the characteristics of the network flow. For example, in the case of a typical TCP/IP connection, the resulting JSON object contains 14 keys, corresponding to a total of 25 individual features.

5.A.6 Kitsune

The Python implementation of the Kitsune feature extractor is obtained from the GitHub repository at commit `28a654b` [27]. A custom Python script is used to initialize the feature extractor (`FE`), provide the path to the PCAP file, and configure it to process all packets in the capture. Features are extracted by repeatedly calling `get_next_vector()` until no more data is returned, and the resulting vectors are written to an output file. Kitsune produces 100 features per packet.

5.A.7 CICFlowMeter

We use the improved version of CICFlowMeter published by Engelen et al. in their WTMC 2021 paper [28], corresponding to GitHub commit `4dd5319` [29]. This version exports 93 features together with a label that is set to the same placeholder value, “NeedManualLabel”, for every flow. Since RustiFlow also supports this feature set, a detailed overview of the exported features is provided in Table 5.5 in Appendix 5.B under the “CIC” column.

5.A.8 Go-Flows

Go-Flows is downloaded from the GitHub repository at commit `9f5628c` [30] and built using Go version 1.23.3. To export features, the tool requires a JSON configuration file that specifies which features to extract. For the experiments, the `complex_v2.json` example provided in the repository is used, which defines a total of 17 features.

5.A.9 NFStream

NfStream version 6.5.3, installed using PIP, was used during our experiments. In addition to setting the idle and active timeouts and specifying the pcap file or interface, NFStream required several specific configuration options:

- Disabling the NFlow layer-7 visibility features (`n_dissections=0`).
- Enabling the statistical features (`statistical_analysis=True`).
- Enabling the internal performance metrics for the real-time robustness experiment (`performance_report=5`).

This configuration results in 28 NFlow core features and 48 post-mortem statistical features for a total of 76 features. Since RustiFlow also supports this feature set, a detailed overview of the exported features is provided in Table 5.5 in Appendix 5.B under the "NFlow" column, except for the following 6 features: source and destination MAC address, source and destination organizationally unique identifier (OUI), IP version, and VLAN ID.

5.B Rustiflow Feature Sets

Table 5.5: **Overview of the Features Exported by Different RustiFlow Feature Sets**

Feature	Basic	CIDDS	NFlow	CIC	RustiFlow	Feature	Basic	CIDDS	NFlow	CIC	RustiFlow
1. Flow Information											
Flow ID	✓		✓	✓	✓	Protocol	✓	✓	✓	✓	✓
Source IP	✓	✓	✓	✓	✓	Source Port	✓	✓	✓	✓	✓
Destination IP	✓	✓	✓	✓	✓	Destination Port	✓	✓	✓	✓	✓
Flow Expire Cause	✓		✓		✓						
2. Time-based Features											
Timestamp First	✓	✓	✓	✓	✓	Timestamp Last	✓		✓		✓
First Timestamp Fwd			✓		✓	Last Timestamp Fwd			✓		✓
First Timestamp Bwd			✓		✓	Last Timestamp Bwd			✓		✓
Fwd Duration			✓		✓	Bwd Duration			✓		✓
Flow Duration	✓	✓	✓	✓✓	✓	IAT Total					✓
IAT Mean			✓	✓	✓	IAT Std			✓	✓	✓
IAT Max			✓	✓	✓	IAT Min			✓	✓	✓
Fwd IAT Total				✓	✓	Bwd IAT Total			✓	✓	✓
Fwd IAT Mean			✓	✓	✓	Bwd IAT Mean			✓	✓	✓
Fwd IAT Std			✓	✓	✓	Bwd IAT Std			✓	✓	✓
Fwd IAT Max			✓	✓	✓	Bwd IAT Max			✓	✓	✓
Fwd IAT Min			✓	✓	✓	Bwd IAT Min			✓	✓	✓
Active Total					✓	Active Mean				✓	✓
Active Std				✓	✓	Active Max				✓	✓
Active Min				✓	✓	Idle Total					✓

Continued on next page

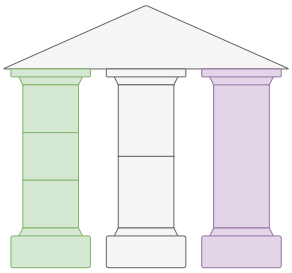
Continued from previous page

Feature	Basic	CIDDS	NFlow	CIC	RustiFlow	Feature	Basic	CIDDS	NFlow	CIC	RustiFlow
Idle Mean				✓	✓	Idle Std				✓	✓
Idle Max				✓	✓	Idle Min				✓	✓
3. Flag Features											
Fwd FIN Flag Count			✓		✓	Bwd FIN Flag Count			✓		✓
Fwd SYN Flag Count			✓		✓	Bwd SYN Flag Count			✓		✓
Fwd RST Flag Count			✓	✓	✓	Bwd RST Flag Count			✓	✓	✓
Fwd PSH Flag Count			✓	✓	✓	Bwd PSH Flag Count			✓	✓	✓
Fwd ACK Flag Count			✓		✓	Bwd ACK Flag Count			✓		✓
Fwd URG Flag Count			✓	✓	✓	Bwd URG Flag Count			✓	✓	✓
Fwd CWR Flag Count			✓		✓	Bwd CWR Flag Count			✓		✓
Fwd ECE Flag Count			✓		✓	Bwd ECE Flag Count			✓		✓
Total FIN Flag Count			✓	✓	✓	Total SYN Flag Count			✓	✓	✓
Total RST Flag Count			✓	✓	✓	Total PSH Flag Count			✓	✓	✓
Total ACK Flag Count			✓	✓	✓	Total URG Flag Count			✓	✓	✓
Total CWR Flag Count			✓	✓	✓	Total ECE Flag Count			✓	✓	✓
Flags		✓			✓						
4. Packet-based Features											
Packet Count		✓	✓		✓	Packet Len Total		✓	✓		✓
Packet Len Mean			✓		✓	Packet Len Max			✓		✓
Packet Len Min			✓		✓	Packet Len Std			✓		✓
Fwd Packet Count			✓	✓	✓	Bwd Packet Count			✓	✓	✓
Fwd Packet Len Total			✓		✓	Bwd Packet Len Total			✓		✓
Fwd Packet Len Mean			✓		✓	Bwd Packet Len Mean			✓		✓
Fwd Packet Len Std			✓		✓	Bwd Packet Len Std			✓		✓
Fwd Packet Len Max			✓		✓	Bwd Packet Len Max			✓		✓
Fwd Packet Len Min			✓		✓	Bwd Packet Len Min			✓		✓
Up Down Ratio				✓	✓						
5. Header-based Features											
Header Len Total					✓	Header Len Mean					✓
Header Len Std					✓	Header Len Max					✓
Header Len Min					✓	Fwd Header Len Total				✓	✓
Fwd Header Len Mean					✓	Fwd Header Len Std					✓
Fwd Header Len Max					✓	Fwd Header Len Min				✓	✓
Bwd Header Len Total				✓	✓	Bwd Header Len Mean					✓
Bwd Header Len Std					✓	Bwd Header Len Max					✓
Bwd Header Len Min				✓	✓						
6. Payload-based Features											
Payload Len Total				✓	✓	Payload Len Mean				✓✓	✓
Payload Len Std				✓	✓	Payload Len Max				✓	✓
Payload Len Min				✓	✓	Payload Len Variance				✓	✓
Fwd Payload Len Total				✓	✓	Fwd Payload Len Mean				✓✓	✓
Fwd Payload Len Std				✓	✓	Fwd Payload Len Max				✓	✓
Fwd Payload Len Min				✓	✓	Bwd Payload Len Total				✓	✓
Bwd Payload Len Mean				✓✓	✓	Bwd Payload Len Std				✓	✓
Bwd Payload Len Max				✓	✓	Bwd Payload Len Min				✓	✓
Fwd Non-Zero Payload Packets				✓	✓	Bwd Non-Zero Payload Packets				✓	✓
7. Re-transmission Features											
Flow Retransmission Count				✓	✓	Fwd Retransmission Count				✓	✓
Bwd Retransmission Count				✓	✓						
8. SubFlow Features											
Fwd Subflow Packets Mean				✓	✓	Bwd Subflow Packets Mean				✓	✓
Fwd Subflow Bytes Mean				✓	✓	Bwd Subflow Bytes Mean				✓	✓
Subflow Count					✓						
9. Window-based Features											
Fwd Init Window Size				✓	✓	Bwd Init Window Size				✓	✓
Window Size Total					✓	Window Size Mean					✓
Window Size Std					✓	Window Size Max					✓
Window Size Min					✓	Fwd Window Size Total					✓
Fwd Window Size Mean					✓	Fwd Window Size Std					✓
Fwd Window Size Max					✓	Fwd Window Size Min					✓
Bwd Window Size Total					✓	Bwd Window Size Mean					✓
Bwd Window Size Std					✓	Bwd Window Size Max					✓
Bwd Window Size Min					✓						
10. Rate-based Features											
Flow Bytes/s				✓	✓	Flow Packets/s				✓	✓
Fwd Bytes/s					✓	Fwd Packets/s				✓	✓

Continued on next page

Continued from previous page

Feature	Basic	CIDDS	NFlow	CIC	RustiFlow	Feature	Basic	CIDDS	NFlow	CIC	RustiFlow
Bwd Bytes/s					✓	Bwd Packets/s				✓	✓
11. Bulk Features											
Fwd Bulk Rate (s)				✓	✓	Bwd Bulk Rate (s)				✓	✓
Fwd Bulk Count					✓	Bwd Bulk Count					✓
Fwd Bulk Packets Total					✓	Bwd Bulk Packets Total					✓
Fwd Bulk Packets Mean				✓	✓	Bwd Bulk Packets Mean				✓	✓
Fwd Bulk Packets Std					✓	Bwd Bulk Packets Std					✓
Fwd Bulk Packets Max					✓	Bwd Bulk Packets Max					✓
Fwd Bulk Packets Min					✓	Bwd Bulk Packets Min					✓
Fwd Bulk Bytes Total					✓	Bwd Bulk Bytes Total					✓
Fwd Bulk Bytes Mean				✓	✓	Bwd Bulk Bytes Mean				✓	✓
Fwd Bulk Bytes Std					✓	Bwd Bulk Bytes Std					✓
Fwd Bulk Bytes Max					✓	Bwd Bulk Bytes Max					✓
Fwd Bulk Bytes Min					✓	Bwd Bulk Bytes Min					✓
Fwd Bulk Duration Total					✓	Bwd Bulk Duration Total					✓
Fwd Bulk Duration Mean					✓	Bwd Bulk Duration Mean					✓
Fwd Bulk Duration Std					✓	Bwd Bulk Duration Std					✓
Fwd Bulk Duration Max					✓	Bwd Bulk Duration Max					✓
Fwd Bulk Duration Min					✓	Bwd Bulk Duration Min					✓
12. ICMP Features											
ICMP Type				✓	✓	ICMP Code				✓	✓
Number of Features	10	10	70	93	185						



6

ConCap: Enabling Fine-Grained Network Traffic Generation for Security Assessments of Flow-based Intrusion Detection Systems

This final chapter unites two core pillars of this dissertation. It revisits the first pillar, AI for Cybersecurity, by addressing the persistent lack of high-quality, reproducible benchmark datasets for NIDS research. Simultaneously, it contributes to the last pillar, Cybersecurity for AI, by exploring realistic adversarial evaluation of ML-NIDS.

This chapter first proposes ConCap*, a framework for fine-grained network traffic generation with automated labeling, allowing researchers to generate new realistic network traffic or extend existing benchmark datasets. Next, this chapter formalizes a new practical evasion strategy against ML-NIDS using "host-space perturbations". A systematic literature review shows that such attacks have been overlooked, likely due to the practical challenges in evaluating them. Leveraging* ConCap*, we demonstrate that host-space perturbations are both effective at evading state-of-the-art classifiers and practical for real-world attackers. Through extensive experiments on academic datasets and real-world testbeds, we show that models trained on* ConCap *generated data generalize across heterogenous environments and that augmenting training with adversarial host-perturbations can improve model robustness against these attacks. These findings lay a foundation for realistic, reproducible, and security-aware evaluation of ML-based NIDS.*

⋆ ⋆ ⋆

M. Verkerken, L. D'hooge, B. Volckaert, and F. De Turck, G. Apruzzese

Submitted for review, April 2025

Abstract Network Intrusion Detection Systems (NIDS) have been studied in research for almost four decades. Yet, despite thousands of papers claiming scientific advances, a non-negligible number of recent works suggest that the findings of prior literature may be questionable. At the root of such a disagreement is the well-known challenge of obtaining data representative of a real-world network—and, hence, usable for security assessments.

We seek to tackle such a challenge in this chapter. First, we propose ConCap, a practical tool meant to facilitate experimental research on NIDS. Through ConCap, a researcher can set up an isolated network environment and configure it to produce network-related data, such as packets or NetFlows, that are automatically labeled—hence ready for fine-grained experiments. We empirically verify that ConCap produces traffic resembling that of a real-world network. Then, we turn the attention to the security of NIDS reliant on machine learning (ML). We show that, despite hundreds of papers proposing, e.g., gradient-based strategies to evade such ML-NIDS, there are certain types of "adversarial perturbations" which are (i) trivial to generate by real-world attackers, but which (ii) have not been considered before because (iii) studying their effect is—or rather, was—difficult from the standpoint of a researcher. To fix this knowledge gap and demonstrate ConCap's utility, we show that some ML-NIDS can be bypassed by changing *a single character* in the command used to launch the attack. We also validate these results in a real-world network, and consider attacks exploiting vulnerabilities never considered by prior work. We release all our resources, including a new dataset of malicious NetFlows.

6.1 Introduction

There is an anomaly in research on Network Intrusion Detection Systems (NIDS). On the one hand, a deluge of papers continues to expand the boundaries of our knowledge, as shown by [1, 2, 3]. On the other hand, a growing number of recent efforts highlight some "pitfalls" that undermine the foundations of prior research (e.g., [4, 5, 6, 7, 8]).

To portray this contrast, we can look at the panorama of open-source datasets used in NIDS-related research, some of which count thousands of citations according to Google Scholar [8]. As a concrete example, consider the paper presenting the CICIDS17 dataset [9]: published in 2018, this paper had 1600 citations in Q3 2022, which increased to 4200 in Q4 2024—indicating a substantial growth in NIDS research. Yet, in 2021, Engelen et al. [5] pinpointed glaring issues in CICIDS17—particularly in terms of *ground-truth labeling of attack samples*. The flaws of CICIDS17 (and also of its successor, CICIDS18) have been "fixed" in 2022 [6]. However, the EuroS&P'24 Best Paper Award [8] revealed that most datasets used by prior research have "bad design smells."

Simply put, the stark reality is that *(i)* data is required to support a paper's claims, but *(ii)* high-

quality data is hard to come by in the NIDS context—especially from the viewpoint of an academic researcher [2, 8, 10]. To aggravate this problem, modern NIDS increasingly rely on data-driven techniques, such as machine learning (ML). Therefore, *labeled* data is necessary to properly evaluate the pros and cons of state-of-the-art NIDS [2]. Finally, despite the benefits that open-source datasets (under the assumption that they are correctly labeled) can provide to a researcher [11], exclusive reliance on such "static benchmarks" prevents one from *generating new data*. This impairs the assessment of novel forms of attacks that do not entail (and potentially require unrealizable [12, 13, 14]) manipulations of the data points included in the benchmark dataset. We aim to rectify such a "data problem."

Our first "technical" contribution is ConCap, an open-source system to **generate network traffic mimicking that of a real-world network**. ConCap is particularly suited to generate network-related data pertaining to *malicious* activities. Such data can then be used alongside "benign" data taken from the network environment that the NIDS is meant to protect—which is a well-founded assumption [15, 16, 10]. Practically, such environment can be: *(a)* that of a benchmark dataset [11], or of *(b)* an ad-hoc network testbed for experimental research [17], or even that of *(c)* a real-world network [18]. We design ConCap so that it is *(i)* flexible enough to enable reproduction of a variety of (allegedly malicious) network activities, whose generated datapoints are *(ii)* automatically labeled at the granular level, while also enabling *(iii)* control of the network conditions (e.g., to simulate resource starvation). As such, ConCap represents a solid foundation to address the "data problem" that affects NIDS research.

Our second "conceptual" contribution is a *reflective analysis* of prior work on the security of ML-based NIDS (ML-NIDS), culminating in a (relatively) *new way to bypass such systems*. Research has shown that ML-NIDS are vulnerable to "adversarial perturbations" [14, 19, 20, 21], some of which rely on gradient-based strategies. For instance, an EuroS&P'24 paper [22] used the well-known FGSM attack [23] to evade an ML-NIDS trained on CICIDS17 having a baseline accuracy of 99%. Such perturbations, however, are applied by directly manipulating datapoints that are, in practice, generated by a dedicated appliance. For example, a host controlled by an attacker generates network traffic that is captured by, e.g., a router or the ML-NIDS itself: the perturbations of prior work have been applied to the data created by these devices—such as manipulating a PCAP trace, or a set of NetFlows, via fine-grained perturbations. To reliably do so in the real world, an attacker hence requires direct access to a highly-privileged device, which is not a very realistic threat model [14]. We posit that real attackers seeking to evade an ML-NIDS would do so by issuing slightly different commands on the host they can directly control—and not by manipulating datapoints collected by other devices. **We define such a tactic as a "host-space perturbation".** Perhaps surprisingly, such an approach has not been investigated so far: our systematic literature review of 292 papers revealed that most adversarial perturbations were crafted by manipulations of pre-collected datapoints. This finding denotes a blind spot in the security of ML-NIDS.

Our third "empirical" contribution is the combination of our previous two contributions. First, we empirically validate that ConCap produces network-related data resembling that of a real-world network. Then, we demonstrate the practical utility of ConCap to examine the effects of

exemplary "host-space perturbations" against previously proposed ML-NIDS. As a case study, we consider the well-known patator ssh brute-forcing attack [24], which is included in CICIDS17/18. We showcase that ML models having perfect detection rate against NetFlows generated by launching the command patator persistent=1 **cannot detect a single "adversarial NetFlow" generated by launching** patator persistent=0. These results hold also in an experiment carried out in a real-world network.

Contributions. We first position our work within extant literature (in Section §6.2). Then, we provide a threefold contribution to the state of the art in network security. We:

- Develop ConCap, a system for generating ad-hoc network traffic (§6.3). ConCap is particularly suited to *create and label* "malicious" datapoints—facilitating network security research. The realistic fidelity of the data produced by ConCap is validated in §6.4.
- Propose and *formalize* the concept of adversarial "host-space" perturbations, representing a convenient and practically feasible way to bypass ML-based NIDS (§6.5). Through a literature review of 292 papers, we show that "host-space" perturbations have not been studied by prior work on adversarial ML in NIDS contexts.
- Demonstrate the utility of ConCap through a security assessment of ML-NIDS (§6.6). Via experiments on benchmark datasets as well as in a *real-world network*, we show how ConCap can be leveraged to scrutinize (and improve) the robustness of well-known ML-NIDS against adversarial "host-space" perturbations.

We discuss our contributions in §6.7, wherein we also describe additional experiments. We release *all* our resources [25].

6.2 Related Work and Motivation

We summarize the domain of NIDS (§6.2.1) and we outline the challenges of acquiring data for assessing NIDS (§6.2.2), representing the root of the problem tackled by this chapter. Then, we motivate our "technical" (§6.2.3) and "theoretical" (§6.2.4) contributions. Our contributions specifically focus on NIDS analysing network flows (NetFlows [26]) due to the widespread usage of this datatype [2, 8, 27].

6.2.1 Network Intrusion Detection Systems (NIDS)

Modern networks are constantly under attack [28, 29], and NIDS represent the first line of defense against the ever-evolving cyberthreats [30]. The main goal of an NIDS[1] is to *detect* any given threat (e.g., a botnet-infected host, a remote DDoS attack, or an attacker who acquired access to an internal host) as early as possible so that proper mitigations can be enacted to mitigate the damage of an offensive campaign [33]. Fig. 6.1 shows a schematic of an NIDS deployment scenario.

[1] Borrowing from [2], we use "NIDS" in a broad sense, including also SIEM [31] or EDR [32] (which can be seen as extensions of NIDS).

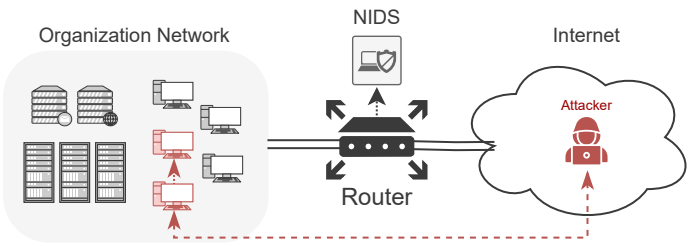


Figure 6.1: **Exemplary deployment scenario of an NIDS.** Note that the attacker *cannot* control the router or the NIDS (otherwise, it wouldn't be surprising if the NIDS cannot detect the attack).

The advent of data-driven technologies, such as AI/ML (which support both signature- and anomaly-based methods [2]), has been adopted by NIDS developers [34]. Yet, even state-of-the-art NIDSs struggle with the sheer amount of attacks that target current organizations [28]. Intriguingly, even though practitioners do appreciate the analytical capabilities of ML [35], modern security operation centers are overwhelmed with false alarms [36, 37]. Put simply, despite being an instrumental security tool, there is a constant need to improve NIDS—creating a fertile ground for research.

Since the seminal work by Denning [38], thousands of scientific papers have sought to propose (e.g., [39, 40, 41, 42]), enhance (e.g., [43, 44]), or assess (e.g., [16, 45, 46]) NIDS. Despite all such efforts, a recent SoK [2] revealed that there is skepticism among industry experts with regard to the results claimed in research. Such doubts are well-founded: various recent papers (e.g., [6, 7, 8]) identified critical issues that are becoming endemic in research. The root cause of most such critiques is the *poor quality of the data* used to test these systems—which is a problem that has been known to affect this research domain since at least 2010 [47].

6.2.2 Research Challenge: Obtaining Data for NIDS

Any security tool must be tested before its deployment, and such tests require data. In the context of NIDS, such an evaluation should be carried out on data that is representative of the environment in which the NIDS is meant to be deployed [2]. Unfortunately, carrying out the abovementioned operations is tough to accomplish from the perspective of an (academic) researcher. Let us outline the options that enable one to collect network-related data for a security assessment of a NIDS, explaining their challenges.

- *Real-world capture from the deployment environment.* This is ideally the best option since it guarantees that the data resembles the one that will be generated by the monitored network. However, such an option may not be available: the researcher may not have access to such data due to privacy reasons, and infecting/attacking physical devices may not be acceptable in some organisations (even if for research).
- *Synthetic capture from a custom environment.* This is a sensible alternative: by creating an ad-

hoc network (e.g., via virtual machines [9]), a researcher has plenty of freedom to collect and generate any sort of data. However, the *benign* data may not be representative of the deployment environment. Moreover, even in such a setup, it is currently challenging to precisely distinguish benign from malicious data points: as shown in [6, 8], there is a risk of mistakenly "label" benign samples as malicious. Such errors can skew the results of the final assessment [48].

- *Reliance on benchmark datasets.* The last option is to use publicly available data, e.g., generated by other researchers in their own environments (either from the real-world, or synthetically). This is a convenient option: it is exempt from privacy concerns and requires minimal technical expertise (since no simulation occurs). However, the validity of the corresponding evaluation will depend on whether the benchmark is a meaningful representation of the network wherein the NIDS is meant to be deployed.

For real-world deployments, it is paramount to evaluate the NIDS in the (real) network to be monitored by the NIDS: as highlighted by Sommer and Paxson [47], networks present immense variability. Hence, even if any given NIDS is shown to "work well" on data from a custom network, it is questionable whether the same NIDS works well also in other networks. This is known as the "generalisability" [49, 50] (or "transferability" [51]) problem of NIDS, which prevents the creation of plug-and-play NIDS (confirmed by practitioners [18]). However, *a research paper needs not to aim for real-world deployment to provide a significant contribution to the state of the art* [2]. Our work is rooted in this truth: we focus on improving future research on NIDS—which not necessarily requires assessments on "real" networks to be valuable.

6.2.3 Practical Generation of Network Data

Prior research on NIDS suffers from a "data" problem. Our main goal is to provide a solution to this problem by enabling future research to carry out meaningful assessments of NIDS.

To elucidate the importance of our first contribution, we present a **motivational example**. Suppose a researcher has no access to a real-world network for NIDS assessments. How can such a researcher conduct meaningful experiments? From our prior descriptions (§6.2.2) we identify three possible options.

- The researcher can create a simulated/virtual network, but doing so requires dealing with the labeling issues (for the malicious traffic), and is (likely) *limited to small-scale evaluations* due to the impossibility of recreating a large network environment [52].
- The researcher can exclusively rely on benchmark datasets, but this would *limit the evaluation to the data within the benchmark dataset* (e.g., even by manipulating the benchmark's datapoints, there is a risk of breaking domain constraints [12]), preventing exploration of new threats.
- To overcome the abovementioned limitations, the researcher can do a mix of the above [15, 16]: they can use "benign" data collected from a public dataset (potentially validated by prior work [6]), and then generate "malicious" data to use alongside the benign data to carry out a proper evaluation.

We argue that the third option is the most enticing one. As of 2025, there are various publicly

available datasets captured in large networks (see [11] for a list) whose data can be treated as "benign" for experimental purposes.[2] Hence, if one could generate "malicious" data so that *(i)* it is "correctly labeled," and *(ii)* it resembles the data generated by the host of the "benign" network, then one would be able to test a NIDS against a wide array of cyber threats—including new ones.

Research Goal #1. We seek to develop an *open-source* tool that enables the automatic *generation and labeling* of network traffic data that resembles a real network.

6.2.4 Shortcomings of Adversarial Perturbations

To set the stage for our second contribution (§6.5), let us summarize the field of ML security from a NIDS perspective.

Thousands of papers have highlighted that ML models are vulnerable to "adversarial perturbations", i.e., tiny manipulations that adversely affect an ML model's performance [53]. Unfortunately, such attacks can also work against ML models designed for NIDS [14].

There are various ways to simulate such "adversarial ML attacks" (e.g., [54, 55, 56]) to, e.g., assess the robustness of an ML model against evasion attempts at test time [57]. From the viewpoint of a researcher, it is crucial to determine where (i.e., in which "space") the perturbation is applied [58]. Historically, adversarial perturbations were applied in the *feature space*, i.e., by directly changing the feature vector provided as input to the ML model.[3] However, as pointed out by Pierazzi et al. [58], *real attackers operate in the "problem" space*—which not necessarily overlaps with the feature space [60]. The issue is that careless manipulation of the feature space risks creating "adversarial examples" that may not be physically realizable and/or which violate domain constraints [12]. (We provide in Appendix 6.C an original discussion on some key differences between problem- and feature-space in various domains.)

The paper by Pierazzi et al. [58] questioned the real-world validity of the results portrayed by previous research, since feature-space perturbations were the norm[4] in the adversarial ML domain—including the specific context of ML-NIDS [14]. As a consequence, some works (e.g., [63, 19]) began to explore scenarios wherein the perturbations were not applied to the input features, but in other spaces—allegedly being a better representation of a real attacker's operations. For instance, Han et al. [19] crafted "traffic-space perturbations" by manipulating the packets in a PCAP trace, adding junk bytes to the packets' payload: such changes would propagate to the feature space, thereby influencing the analysis of the ML-NIDS; whereas Wang et al. [63] claimed to generate "problem-space adversarial examples" by mutating network packets so that the corresponding feature representation (i.e., a NetFlow) was misclassified by the ML-NIDS. Despite the proven effectiveness

[2] Such "benign" data can also come from the specific (real) network in which the NIDS is to be deployed—but, as we argued, this is not necessary (although it would increase the soundness of an evaluation).

[3] E.g., changing the values of a "malicious" NetFlow to induce the targeted ML model to classify it as "benign", thereby evading an ML-NIDS [59].

[4] Most papers on adversarial ML focus on image recognition [55, 61], where "feature-space" perturbations entails manipulating pixels in an image [62].

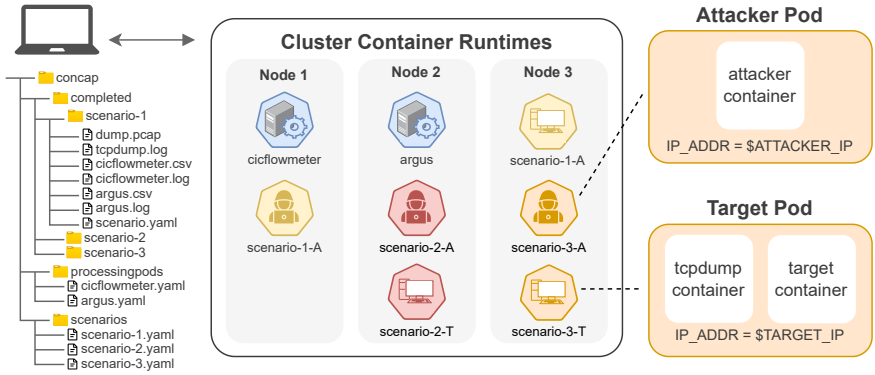


Figure 6.2: **Overview of ConCap.** [left] ConCap configured with two NetFlow extractors and three scenarios, [mid] executing all scenarios simultaneously on the cluster. [right] A view of a running scenario's attacker and target pods.

of such perturbations (which led to correct feature vectors denoting malicious samples that bypassed the targeted ML-NIDS), we argue that *such strategies may not resemble a real attacker's actions*.

Indeed, prior work is limited to applying perturbations by manipulating pre-collected datapoints—such as adding junk bytes to a packet in a PCAP trace. However, from an attacker's viewpoint, *precisely* adding that exact amount of junk bytes to the specific packet (and just that one!) sent by their controlled host may be unfeasible (especially if the payload is encrypted); besides, the packets "seen" by the router (and forwarded to the NIDS) may not correspond to those sent by the attacker-controlled hosts.[5] Put simply, to bypass an ML-NIDS, we argue that a real-world attacker may opt for more straightforward perturbation-related strategies—such as issuing different commands to their controlled hosts. We define such a class of adversarial ML attacks as "host-space perturbations" since they are physically applied to a (attacker's controlled) host.

Research Goal #2. We seek to: justify that "host-space perturbations" are a realistic (and, so far, overlooked) way to attempt evasion of ML-NIDS; and study their effects.

6.3 Our Proposed Tool: ConCap

As our major technical contribution, we present ConCap (short for "Container Capture"), our tool to generate realistic network traffic for experimental research in NIDS. After stating the overarching goals of ConCap (§6.3.1), we describe its architecture (§6.3.2), shown in Fig. 6.2, and provide low-level details (6.3.3) and comparisons with prior research on "traffic-generation" (§6.3.4). We

[5] For instance, consider the attack proposed in [64], described as follows: "a dummy packet is injected into a specified location $k \in [0, n-1]$ among the first n packets of a flow and an [universal adversarial perturbation] is injected into it." An attacker can certainly do so *in theory*—but how, *in practice*?

will empirically assess and validate the functionalities of ConCap in the next section (§6.4).

6.3.1 Overview, Goals, and Core Principles

The overarching goal of ConCap is to simplify the operations involved in the generation and labeling of *malicious* network activities that resemble those in a *physical and real-world* network. Specifically, ConCap seeks to fulfill three objectives:

- *Attack flexibility.* The tool must allow one to specify and reproduce a variety of actions that lead to network communications—including those denoting "malicious" behavior, for the purpose of security assessments.
- *Data collection and labeling.* The tool must automatically: *(i)* capture network traffic related to the specified actions; *(ii)* generate statistical metadata summarizing the communications, i.e., NetFlows; *(iii)* assign a label to these NetFlows.
- *Parallelism and control.* The tool must enable simultaneous execution of diverse experiments, and fine-grained control of the network conditions of each experiment—so to reproduce the behavior of any network, and enable "bulk" experiments.

We outline the core principles of ConCap, showing that our three objectives are embedded in its underlying design.

ConCap is rooted on the concept of "scenario", which serves as a customizable blueprint for an entire experiment. When defining a scenario, developers can specify: *(i)* the activities carried out by the involved hosts; *(ii)* the conditions of the overarching network environment; *(iii)* the fine-grained label to assign to the generated network traffic metadata.

ConCap executes each scenario in an isolated "containerized" environment that mimics the behavior of real, physical hosts, and allows for unlimited parallel execution of scenarios—subject to the resource constraints of the experimental testbed on which ConCap is executed.

A scenario in ConCap assumes a set of hosts: one designated as the "attacker", and one or more designated as the "target". These hosts are controllable by the developer. ConCap captures the network packets (as a PCAP trace) exchanged between these two hosts, automatically extracts high-level NetFlow records, and then labels such NetFlows according to the scenario definition.

6.3.2 Architecture and Design Choices

Here, we present the various elements that compose ConCap and motivate each of our design choices.

Isolated Environment. As we explained (§6.2.2), the major advantage of running network simulations is the ability to generate malicious network traffic without putting real-world networks at risk. Such simulations can be carried out by, e.g., using virtual machines (as done, e.g., in [9]) or by using *containers*. Containers are a lightweight alternative to virtual machines [65], which

are becoming very popular in related research (e.g., [52, 13, 66]) also for the ease of reproducibility [67]. For this reason, we used containers to develop ConCap. There exist many open-source solutions that can be used to deploy containerized applications [68]. For ConCap, we rely on Kubernetes [69], due to its widespread adoption (110k GitHub stars [70]) and key advantages over alternatives, such as enabling deployment and management of workflows across multiple machines [71]—which is required to achieve our design goals of simultaneous execution of scenarios. For comparison, DetGen [52] relies on Docker [72], which by default does not support parallel scenario execution across multiple machines. We stress, however, that Kubernetes alone *does not* provide the functionalities provided by ConCap (see §6.3.1): we simply use Kubernetes as the backbone.

Attacker and Target(s). In Kubernetes, a "pod" is the smallest deployable unit, functioning as a logical host that groups one or more containers with shared storage and networking resources. To ensure isolation and enable fine-grained control over each host involved in the "attack", we create ConCap so that the attacker and target(s) hosts are deployed in separate pods. This allows to configure parameters such as bandwidth and latency for each host. The attacker pod runs one container that simulates the behavior of the attacker's host. In contrast, each target pod runs two containers: one serving as the host targeted by the attacker, and another that captures the attacker-target network communications (via tcpdump).

Input, Execution, and Output of ConCap. To illustrate how ConCap operates, Fig. 6.2 presents a schematic of a typical setup of ConCap, highlighting the most relevant logical units. **[Input]** On the left, the "scenarios" folder contains three YAML configuration files, each defining a given scenario to be executed by ConCap. An example scenario configuration is shown in Listing 6.1, highlighting the various options (i.e., attacker and target definition, network conditions, and labeling) that can be configured before executing any given experiment. For added flexibility, ConCap supports using different NetFlow generation tools. This can be specified through configuration files located in the "processingpods" folder, where the details of the desired tool are defined. **[Execution]** When executing a scenario, ConCap parses the configuration files and interacts with both the host machine and the cluster, consisting of one or more nodes. If multiple scenarios are specified, ConCap supports concurrent execution by distributing them across different nodes, enabling parallelism (see the central section of Fig. 6.2). ConCap follows the instructions in the scenario YAML files to set up attacker and target pods, apply custom network configurations, initiate traffic captures, trigger attack execution, and run processing pods for feature extraction and labeling. See the rightmost section of Fig. 6.2 for a detailed view of the attacker and a target pod during scenario execution. **[Output]** For each scenario, ConCap creates a dedicated folder containing all experiment outputs, including logs, the PCAP trace, and the labeled NetFlows (see left of Fig. 6.2). The reason why we focus (also) on NetFlows for ConCap is due to their widespread popularity for network-related experiments (both in research and in practice [2, 73]); for instance, most public benchmark datasets are also released in this format [11]. Such a design choice serves to facilitate future research.

6.3.3 Implementation (and Customisability)

We provide some low-level details of ConCap, emphasizing our original contributions that enable its flexibility.

At the core of ConCap's traffic generation process is the scenario file, which serves as a blueprint for network traffic generation. We created this file so that the developer can specify the properties of the involved hosts (attacker and target(s)), of the network channel, and the fine-grained label. For instance, Listing 6.1 provides an example scenario configuration for an nmap port scan launched by the "attacker" against an Apache webserver running on the "target".

The components describing the "attacker" and "target" hosts require a *name* and the *containerImage* to deploy the containers that simulate their behavior. Additionally, we designed ConCap so that it is possible to specify the conditions of the computing runtime (e.g., *CPU* and *memory*): this is crucial to reproduce attacks that disrupt the availability of the target host (e.g., DoS). The "attacker" component also has the *atkCommand* (used to start the attack), and an optional *atkTime* parameter that controls the attack duration (in seconds): intuitively if $atkTime > 0$, then the attack will stop after the provided number of seconds; otherwise, the attack will continue until it naturally terminates. We also allow to configure the startup probe of the "target" component, which determines when the target is ready, as well as the filter used by $\mathrm{tcpdump}$ for the packet capture (PCAP).

The *network* component allows fine-grained control of the networking environment (which we implemented via Traffic Control [74]), enabling to specify, e.g., the bandwidth, latency, and packet loss of the communication channel. Finally, the *label* component describes the labeling logic applied to the NetFlows (generated after processing the PCAP file). This can be configured globally, and individually per host. Importantly, given that ConCap enables precise control of the entire attack workflow (i.e., attacker and target(s) host and network conditions), the assigned labels ensure the resulting NetFlows are associated with the correct ground truth (since they all share the same "malicious" generative process). Finally, we remark that ConCap supports any type of NetFlow generation software. In our proposed implementation of ConCap, we have integrated CICFlowMeter and Argus, which are popular in research and open source [26, 2]. Choosing a given NetFlow tool is done by editing a dedicated configuration file (shown in Listing 6.2 in the Appendix 6.A).

6.3.4 Comparison with Prior Work

We are not aware of any open-source tool that fulfills the same goals as ConCap. Closely related works are DetGen [52] and SOCBED [75]; we tried to find more works on "generation of realistic network traffic" by looking at the accepted papers in various top-tier conferences (NDSS, IEEE S&P, USENIX Sec, ACM CCS) since 2019 (similarly to [4, 55]). We found that the only related work was netUnicorn [76]. Then, we expanded our search by using the snowball method [77] and (recursively) looked for all peer-reviewed papers cited by [52, 76], and identified four related

works [78, 79, 80, 81, 82]. Let us position ConCap within these related works.[6]

Summary. Works by Beltiukov et al., PINOT [78] and netUnicorn [76], are ideal for generating benign traffic but raise security risks for generating malicious traffic (see §6.2.2). To generate benign traffic at scale, netMosaic [79, 80] harnesses public code repositories to automatically capture network traffic for a wide range of applications, but does not allow fine-grained labeling of malicious datapoints. Finally, Zhou et al. [81] propose to use foundational models to artificially "augment" any given network-traffic dataset; however, it is unclear if such methods can produce real-world data (there is no real-world assessment in [81]).

ConCap **vs DetGen.** First, DetGen [52] does not allow parallelism by design. Second, DetGen does not generate and label the NetFlows of any given experiment, thereby forcing the developer to do so manually—which is error-prone [5]. Finally, DetGen does not allow the same degree of flexibility provided by ConCap. To provide evidence of this claim, we looked at the public repository of DetGen (available at [84]), inspected its source code, and observed the following: *(i)* we did not find any way to set the available CPU/RAM of the attacker/target; *(ii)* a predefined timeout is required for every scenario, potentially stopping the scenario before successful completion; *(iii)* comments in the code state that the attack sometimes starts before traffic is captured, highlighting the lack of fine-grained control over the scenario execution by DetGen. Although we tried to compare the traffic generated by both approaches, we were unable to run any of their example scenarios (due to deprecated software/runtime errors that we could not troubleshoot with the provided documentation).[7]

ConCap **vs SOCBED.** First, SOCBED primarily targets log-data collection: even though it can create PCAPs, it does not *(a)* extract NetFlows by default, or *(b)* label them. The latter is crucial: manual post-hoc labeling of attack traffic is unreliable due to the background noise generated by the concurrent simulation of benign and malicious behaviors. Second, SOCBED relies on a *fixed* network topology designed for long-running experiments running *virtual machines* on a single host. Their sample scenario requires around one hour to complete, including a 15-minute setup time. In contrast, ConCap supports fast (startup time of seconds), parallel execution of isolated attack scenarios across multiple hosts. Finally, we found no example to configure network characteristics, such as per-host bandwidth or delay, in SOCBED.

Takeaway. ConCap is the first open-source tool for generating realistic, labeled network traffic for NIDS research. It offers attack flexibility, automated NetFlow generation and labeling, and parallel scenario execution with fine-grained control. A short demo is available in our repository [25].

[6] Some orthogonal works propose testbeds that do not provide the functionalities of ConCap: e.g., Gotham [83] cannot ensure fine-grained labeling of malicious datapoints, whereas I2DT [10] can only inject packets in a PCAP.

[7] We provide a step-by-step guide in the Appendix 6.A.3 on how to configure one of DetGen's predefined scenarios, *capture-020-nginx*, with ConCap.

Table 6.1: **Network traffic on bare-metal servers and ConCap.** For both network activities, the number of NetFlows is identical and the number of network packets has minimal variations (as expected in realistic networks).

Attack Options	Number of Packets		CICFlowMeter Flows		Argus Flows	
	Bare-metal	ConCap	Bare-metal	ConCap	Bare-metal	ConCap
-Pn -sS	5	5	2	2	2	2
-Pn -sS -sV	122	103	10	10	11	11
-Pn -sT	6	6	2	2	2	2
-Pn -sT -sV	127	107	10	10	11	11
-Pn -sU	4	4	3	3	4	4
-Pn -sU -sV	4	4	3	3	4	4
-sS	13	13	5	5	6	6
-sS -sV	137	113	13	13	15	15
-sT	14	14	5	5	6	6
-sT -sV	133	115	13	13	15	15
-sU	12	12	6	6	8	8
-sU -sV	12	12	6	6	8	8

-Pn = Treat host as online
-sS / -sT / -sU = TCP SYN, TCP Connect or UDP scan
-sV = Probe open ports to determine service and version info

(a) Nmap port scan

Attack Options	Number of Packets		CICFlowMeter Flows		Argus Flows	
	Bare-metal	ConCap	Bare-metal	ConCap	Bare-metal	ConCap
P=0 RL=0 T=1	95 194	78 309	3400	3400	3400	3400
P=1 RL=0 T=1	33 103	29 698	578	578	578	578
P=1 RL=1 T=1	38 726	35 332	578	578	578	578
P=0 RL=0 T=5	95 170	78 426	3400	3400	3400	3400
P=1 RL=0 T=5	33 554	30 293	595	595	595	595
P=1 RL=1 T=5	39 066	35 823	595	595	595	595
P=0 RL=0 T=10	94 941	78 347	3400	3400	3400	3400
P=1 RL=0 T=10	35 399	31 506	680	680	680	680
P=1 RL=1 T=10	40 772	37 384	680	680	680	680

P = Persistent, RL = Rate-Limit, T = Threads

(b) Patator SSH Bruteforce

6.4 Real-world Validation of ConCap

We demonstrate that ConCap can replicate the behavior of a real network—a validation that has not been done in most prior works (e.g., [81]). To this end, we compare the network data generated by ConCap to that of a real-world network at both the packet level (§6.4.2) and the

NetFlow level (§6.4.3).

6.4.1 Experimental Setup

Our comparisons entail two distinct environments.

- *Real-world network.* Two "bare-metal" physical hosts, each having six Intel Core i5-9400 @ 2.90GHz, 32GB RAM running Ubuntu 20.04.6 (5.4.0-67-generic x86_64) and connected by a 1 Gbit switch.
- ConCap. A Kubernetes cluster (v1.29.0) with 1 control plane and 3 worker nodes. Each node has 16GB RAM, four Intel Xeon E5-2640v4 @ 2.4GHz running Ubuntu 22.04.3 (Linux 5.15.0-91-generic). The machines are interconnected by a 10 Gbit switch.

Importantly, when configuring the scenarios run by ConCap, we will specify parameters that allow us to approximate the specification of the real-world network.

Attacks. Our experiments envision carrying out two types of (well-known) network attacks: a simple port scan (via nmap [85]) against an Apache webserver, and a more advanced brute-force attack (via ssh-patator [24]) against an OpenSSH server. Specifically, for the port scan, we will scan two ports: 79 (closed for both TCP and UDP) and 80 (open TCP port of Apache webserver, closed for UDP), using multiple scanning options provided by nmap (we test 12 combinations of -Pn -sS -sV -sT -sU, see Table 6.1a for an overview). For the SSH brute force, we configure patator to use a dictionary with 3400 combinations of username-passwords (downloaded from [86]), and then execute the attack multiple times, each with a different option (we test 9 combinations by varying: persistent=0/1 -RL=0/1 -T=1/5/10, see Table 6.1b for an overview).

Workflow and Data collection. We first execute the experiment on the real-world network. We instruct one bare-metal host to carry out the "attack", whereas the other host (acting as the "target") is configured accordingly (i.e., running the Apache and the OpenSSH server) and it also passively captures all network traffic via tcpdump. For each attack (12 for nmap and 9 for patator) we capture all the network traffic (as PCAP) and then generate the corresponding NetFlow (via CICFlowMeter and Argus). Next, we focus on the experiment on ConCap, ensuring that the scenarios are configured so that the "attacker" and "target" hosts resemble the specifics (in terms of computational power, software, and commands executed) of the "bare metal" machines. We provide the configuration files in our repository [25]. By design, ConCap will automatically capture the traffic and generate the NetFlows.

6.4.2 Network Packet Analysis

"Does ConCap generate network packets that are similar to those of a real-world network?" We address this question by carrying out qualitative and quantitative analyses.

Qualitative analysis. First, we use Wireshark [87] to manually inspect the PCAP traces. We found that, for both types of our attacks (i.e., the 12 nmap and the 9 patator variants), the bytes in

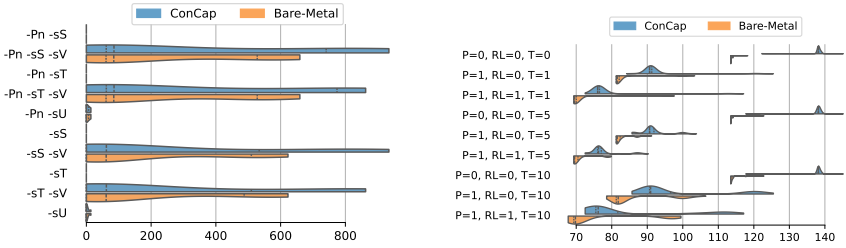


Figure 6.3: **NetFlow feature distribution of *mean packet length* for traffic generated by ConCap and a bare-metal setup (i.e., a real-world pair of physical hosts).** The leftmost plot is for nmap and the rightmost is for patator.

the individual packets generated by ConCap exactly match those of the real-world network— except for expected differences in headers (e.g., MAC and IP addresses, high ports, checksums). We also observed some variations in the *window size* and *maximum segment size* which affect the amount of data that can be handled by the receiver: such (minimal) differences are due to the physiological diversity of each network, and their existence is evidence that the packets generated by ConCap can also present a degree of uniqueness which is intrinsic to real-world networks.

Quantitative analysis. Next, we focus on the *total number of packets* exchanged for each attack. We report the results in Tables 6.1a (for nmap) and 6.1b (for patator). The leftmost column reports the specific options for each attack, whereas the second and third columns report the packets exchanged by the "target" and "attacker" host during the attack for both the real-world and ConCap network setup (we also report the NetFlows, covered in §6.4.3).

- Nmap. For the port scan, there is an almost perfect match. The differences occur only when the attacker probes the service running on the open port (option -sV). This is expected: the -sV option induces the server to provide the index HTML page of the Apache webserver, which is 10,918 bytes. In ConCap, such an exchange requires 2 packets, whereas the same payload requires 8 packets in the real-world network— due to small differences in the network conditions such as TCP window scale [88, 89].
- Patator. For the SSH brute force, we observe differences of $\sim$10% in the number of packets. In this case, the difference is due to the TCP/IP stack handling data acknowledgment on the different hardware setups. Additional testing in another replication experiment on a second real network showed similar but different deviations in the number of ACKs. The TPC RFC [90] allows for this nondeterministic behavior of "delayed ACKs" which send fewer than one ACK segment per data segment received and is even expected by the official specification [88, 91] to increase efficiency in the Internet and the hosts. However, as demonstrated by our qualitative analysis, the payload is the same.

In summary, any packet-level differences are due to inherent and unpredictable characteristics of the network, which do not affect the contents of the communication payloads.

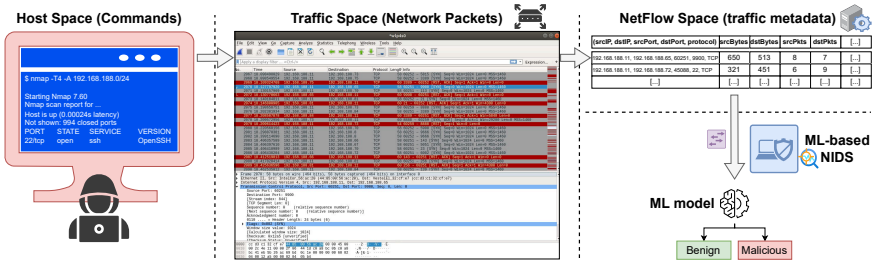


Figure 6.4: **From attackers' actions to ML inputs.** [Left] The attacker launches `nmap` on their controlled host. [Middle] This leads to the creation of multiple network packets that are captured by some dedicated network appliance (e.g., a router). [Right] Then, NetFlows are extracted from the PCAP trace, which are sent to (and analysed by) the ML-NIDS. A realistic attacker has no access to the "traffic space" (i.e., middle panel) and "NetFlow space" (i.e., right panels): the attacker can only operate at the host level (i.e., left panel).

6.4.3 Network Flow Analysis

To shed further light on ConCap-generated data, we investigate the differences at the NetFlow level using two state-of-the-art tools: Argus [92] and the fixed CICFlowMeter [93].

First, we carry out a quantitative comparison focusing on the number of NetFlows exchanged between the "attacker" and "target" host for each attack—reported in Tables 6.1a and 6.1b. Notably, there is always a perfect match: despite differences in packet counts, the number of NetFlows is consistent when the PCAP is processed by the same NetFlow software (CICFlowMeter and Argus follow a different logic to create NetFlows[8]). Then, we analyze the distributions of NetFlow features (for simplicity, we focus only on CICFlowMeter) across the different attacks. Fig. 6.3 presents side-by-side comparisons of the *mean packet length* distribution for all variations of our attacks. Additional violin plots for other features are provided in Appendix 6.E.3. We observe that all features exhibit similar distributions, or any differences can be explained via our packet-level analyses (e.g., more packets sent with empty payload lead to a decrease in the mean packet length).

> **Takeaway.** Our analyses at the packet- and NetFlow-level revealed that the network traffic generated by ConCap resembles that of a real-world network. Deviations are due to expected differences in the network channel, which are impossible to control—but do not affect the payload content.

Finally, we evaluated the determinism of ConCap-generated traffic by repeating the experiments 100 times (details in Appendix 6.E.1). The results show only minor variance, which is expected in real-world networks.

[8] **Bugfix:** we encountered an unexpected discrepancy in the NetFlows generated by Argus. We reached out to the developers, and they confirmed that there was a bug in *their* implementation, and we helped fixed their code. After fixing the code, the number of NetFlows is identical.

6.5 Host-space Perturbations

We now turn to the "conceptual" contribution of our work: *security assessments* of ML-NIDS through the formalization of "host-space" perturbations. We first justify their viability with a reflective analysis (§6.5.1), then define them and highlight their novelty (§6.5.2). Finally, we empirically demonstrate their effectiveness at creating "evasive" datapoints (§6.5.3).

6.5.1 Generic Intuition (and Threat Model)

When evaluating the security of ML-NIDS, the researcher should aim at reproducing the workflow that an hypothetical attacker would follow in the real world [55]. In the context of adversarial perturbations, a crucial question that must be answered is: "What is the problem space?" We answer this question by putting ourselves in the attacker's shoes: through *inductive reasoning* [94], we demonstrate that, in the real world, (most) attackers would (and, likely, can) only operate on the "host space", i.e., on the hosts they control.

Context. Let us assume that an attacker seeks to carry out some malicious operation within a given network. Without loss of generality, such an objective can entail any type of (malicious) network communication: external-to-internal (e.g., a remote DDoS attack); internal-to-external (e.g., data exfiltration); or internal-to-internal (e.g., reconnaissance for lateral movement). The network is protected by some NIDS, which can be deployed anywhere—but, for simplicity, we assume that the NIDS is deployed at the network's border. The NIDS integrates some ML model that analyses network-related data of any type (e.g., log graphs [95, 96], payload [97], or NetFlows [98]): without loss of generality, we assume that the NIDS analyses NetFlows, since it allows broader coverage. In this context, the attacker performs "actions" on their controlled host (external or internal); such actions lead to the generation of network packets which are preprocessed into NetFlows, and then fed to an ML model within the NIDS that determines whether the NetFlows are benign or malicious. Hence, the problem space wherein the attacker operates is defined by the *actions that the attacker can perform on their controlled host*.

Exemplary use-case. Assume the attacker, who has obtained remote access to an internal host, wants to carry out some reconnaissance: the attacker uses the nmap command, specifying any parameter (e.g., $\mathrm{-T4\ -A}$). Such actions generate some packets and NetFlows, whose values depend on the parameters specified by the attacker. This workflow is illustrated in Fig. 6.4. We observe that the packets are captured by the router/switch (to which the attacker has no access), which forwards them to the NetFlow extractor (to which the attacker has no access), which sends the NetFlows to the ML-NIDS (to which the attacker has no access). Hence, the attacker has some control on the packets sent by their controlled host, but once these packets "leave" this host, our envisioned attacker cannot modify them in any way. Simply put, such an attacker cannot manipulate the data captured by a router/switch, nor the resulting NetFlows, nor the individual feature-vectors analysed by the ML-NIDS—*all of which being ways in which prior work applied adversarial perturbations*. If the attacker suspects that the ML-NIDS is trained to detect reconnaissance attempts issued with $\mathrm{nmap\ -T4\ -A}$, the attacker may take another action, e.g., using $\mathrm{-T0}$ (instead

of -T4): doing so would lead to a slower scan, meaning that different packets and, hence, different NetFlows will be generated by their host. (The attacker can also instruct its host to add junk bytes to the payload of each packet—but this would have an effect to the network bandwidth and lead to additional changes to the PCAP trace captured by the router/switch.) Therefore, the perturbations that can be crafted by real-world attackers pertain to the actions (in terms of commands launched) performed on their *controlled hosts*—i.e., the "problem space."

Takeaway. Real attackers perform *actions* which lead to changes in the network data generated by their controlled hosts. From an adversarial ML viewpoint, this can be simulated through "host-space perturbations", which require operating at the host level—and not on the data generated by such a host (and captured by another appliance).

6.5.2 Definition, Security Considerations, Novelty

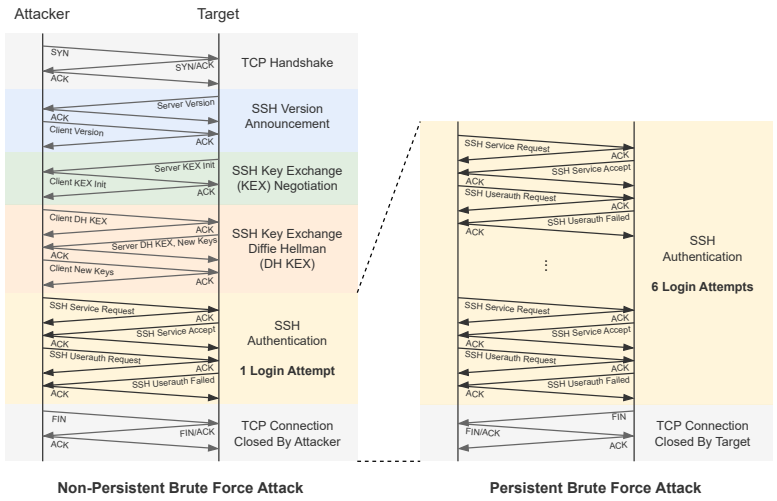
We are not aware of any existing definition of "host-space perturbation". Hence, we provide our own. Broadly, we define a host-space perturbation as "*any operation that allows an attacker to achieve the same goal by executing commands or actions on the hosts under their control (as defined by the specified threat model)*". Here, a "goal" refers to the successful execution of an attack. A more formal definition is as follows.

Assume an attacker who controls a set of hosts $\mathcal{H}$ and aims to achieve a specific goal $\mathcal{G}$ through a series of operations $\mathcal{O}$ executed on those hosts. A host-space perturbation is defined as any alternative set of operations $\mathcal{O}' \neq \mathcal{O}$ that allows the attacker to achieve the same goal $\mathcal{G}$ by operating on the same set of hosts $\mathcal{H}$. It is implicitly assumed that both $\mathcal{O}$ (and $\mathcal{O}'$) generate network traffic that is ultimately analysed by the ML model that the attacker aims to evade via the host-space perturbation—though evasion is not guaranteed.

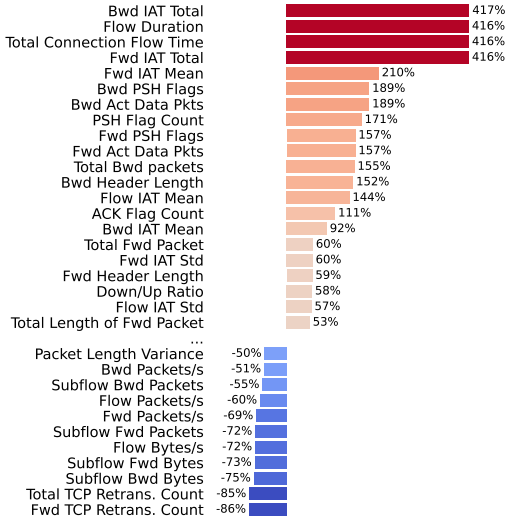
6.5.2.1 Considerations

Let us consider the simple use case (from §6.5.1) of an attacker whose goal ($\mathcal{G}$) is to "port-scan" some device. The attacker does so by issuing nmap -$\mathrm{T0}$ (i.e., $\mathcal{O}$) on their controlled host ($\mathcal{H}$). The attacker can also achieve the same goal ($\mathcal{G}$) from the same host ($\mathcal{H}$) by issuing nmap -$\mathrm{T4}$ ($\mathcal{O}'$). We use this example to make four important considerations on our host-space perturbations.

- *Certain host-space perturbations are easier-to-apply than others.* E.g., changing a parameter from $\mathrm{T0}$ to $\mathrm{T4}$ is a simple and valid host-space perturbation. The attacker could also manually probe each port using netcat; despite being more labour-intensive, it is a valid host-space perturbation.
- Some host-space perturbations may cause substantial changes in the "feature space" (or no effect at all). However, *this is not a concern for the real attacker* [55], whose objective is to evade the ML-NIDS while achieving their overarching goal (e.g., a port scan in the previous example).
- From a security standpoint, an *invalid host-space perturbation* executes a command that is not assumed to be available on the attacker-controlled host (e.g., running sudo without having root privileges), since this violates the threat model.



(a) Packet-level differences between -P=0 and -P=1



(b) NetFlow-level differences (-P=0 / -P=1)

Figure 6.5: **Low-level effects of a host-space perturbation.** We show what happens when going from patator -P=0 to patator -P=1

.

- From the viewpoint of a researcher, *it is unwise to simulate the effects of host-space perturbations via manipulations of pre-collected data*. For instance, simulating the change from -T4

to -T0 by manipulating the NetFlows (i.e., a "feature-space" perturbation, as done in [59, 99]) can lead to inconsistent adversarial examples [12] (we are not aware of a one-way function that goes from "command" to "NetFlow"); whereas manipulating the PCAP trace (as done, e.g., by [19]) is also unreliable because networks are "unpredictable" ecosystems [2]: for instance, going from -T4 to -T0 may lead to some packets being lost or retransmitted.

By definition, "host-space perturbations" satisfy the required properties of "problem-space adversarial ML attacks" [58].

6.5.2.2 Literature review

"Has prior work on the robustness of ML-NIDS considered perturbations involving launching different commands on the attacker-controlled host (i.e., host-space perturbations)?" We answer this question with a systematic literature review, for which we adopt a similar approach as recent works [4, 2]. Overall, we inspect 292 papers derived by analysing works cited, or cited by, six relevant papers: [63, 19, 2, 55, 8, 14]. The complete methodology (following PRISMA [100]) is described in Appendix 6.D.

We could not find any paper that allows to answer positively to our research question. Most perturbations are crafted by directly modifying the data—potentially before preprocessing (e.g., [101]), or via feature-space perturbations that preserve domain constraints (e.g., [102]). In §6.7.3, we constructively discuss the implications of our findings (which do not, nor seek to, invalidate prior work).

Nonetheless, our analysis elucidates a blind spot in ML-NIDS research. To the best of our knowledge, it is unknown how resilient current ML-NIDS are to host-space perturbations. We conjecture that *this gap stems from the practical challenges of studying host-space perturbations in a research setting*: it requires access to an operational networked host, executing "adversarial commands," capturing the corresponding traffic, generating the feature representation, and studying the effects on the ML-NIDS. These steps are non-trivial, especially since most prior work relies on public datasets [8] collected in networks (and, hence, hosts) not accessible by downstream researchers. Our proposed ConCap addresses this challenge, and we demonstrate its utility for this purpose in §6.6.

6.5.3 Empirical Validation

We present a proof-of-concept experiment showcasing how adversarial host-space perturbations "mutate" the resulting network traffic, producing different input samples that an ML-NIDS may struggle to classify—all with minimal effort from the attacker. For consistency, we build our case study on the patator ssh-bruteforce attack (used also in §6.4.1).

Theoretical analysis. At a high level, patator performs repeated SSH login attempts by trying various combinations of usernames and passwords. It can be configured using several options,

one of which is the "persistent" flag (-P). When enabled (-P=1), patator continues attempting logins over a single TCP connection until the SSH server terminates it (by default, OpenSSH closes the connection after six failed attempts). In contrast, when disabled (-P=0), patator initiates a new TCP connection for each login attempt. Fig. 6.5a breaks down the operations in both modes, showing the different packet-level interactions between attacker and target with a default OpenSSH setup. While some communications patterns remain unchanged (e.g., the TCP handshake and key exchange), enabling -P=1 results in many more interactions. At first glance, one might assume that -P=1 simply leads to "≈6x more packets". However, as we will show, the differences between -P=1 and -P=0 are more pronounced and less predictable.

Real-world experiment. Let us practically examine the effects of this host-space perturbation (i.e., comparing patator -P=1 with patator -P=0). We take two hosts deployed in a real, physical network encompassing ∼50 active hosts:[9] One host (the "target") runs an OpenSSH server with default configuration; whereas one host (the "attacker") executes the patator attack—first with -P=1 and then with -P=0—against the "target". We capture all network traffic on the "target" host and generate the corresponding NetFlows using CICFlowMeter, configured identically to §6.4). Fig. 6.5b visualizes the NetFlow-level differences between -P=0 and -P=1, revealing substantial differences introduced by the host-space perturbations. From these results, *we posit that replicating such differences via feature-space perturbations is (almost) impossible.* We test the impact of failing to replicate the exact effects of our host-space perturbation against an ML-NIDS in §6.7.2.

6.6 Demonstration: ConCap for research

As our last "empirical" contribution, we practically combine our previous contributions. The intention is threefold:

- demonstrate the utility of ConCap (our first "technical" contribution, §6.3) to advance the state of the art in NID;
- assess host-space perturbations (our second "conceptual" contribution, §6.5) against state-of-the-art ML-NIDS.
- show how to do the above when one *(a)* can only rely on data from public benchmarks, or *(b)* has physical access to a real-world network—i.e., all cases faced by a researcher (§6.2).

We first present the experimental setup (§6.6.1), and then discuss the experiments in the real-world network (§6.6.2) and conclude with those on benchmark datasets (§6.6.3).

[9] Unlike the testbed in §6.4.1, this network includes a variety of devices, such as IoT, gaming, laptops, and smartphones. More details in Appendix 6.B.1.

Table 6.2: **Experiments on real-world data with ConCap.** We show the results (fpr on benign and tpr on attack NetFlows) of our models for each "train set" and "test set" (averaged over 5 trials; we provide the std.dev. in our repository [25]). Boldface denotes NetFlows generated via ConCap (which are always malicious). The red columns are the "adversarial" experiments on the baseline models, and green columns are the adversarial experiments on the adversarially trained models using ConCap; arrows (↑) denote significant changes w.r.t. the baseline.

	Baseline Training						**Adversarial Training**						**Open World Training**		
Train Set	Benign + P=1			Benign + P=0			Benign + P=1 + P=0			Benign + P=0 + P=1			Benign + P=0 + P=1		
Test Set	Benign	P=1	P=0	Benign	P=0	P=1	Benign	P=1	P=0	Benign	P=0	P=1	Benign	P=0	P=1
DT	<0.001	1.000	0.000	<0.001	1.000	0.000	<0.001	1.000	1.000 ↑	<0.001	1.000	0.988 ↑	<0.001	1.000	1.000
RF	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000	1.000 ↑	0.000	1.000	1.000 ↑	0.000	1.000	1.000
XGB	0.000	>0.999	0.000	<0.001	1.000	0.000	<0.001	1.000	1.000 ↑	<0.001	1.000	1.000 ↑	<0.001	1.000	1.000
SVM	0.000	1.000	0.490	0.000	1.000	1.000	<0.001	1.000	1.000 ↑	0.000	1.000	1.000 =	<0.001	1.000	1.000
DNN	<0.001	1.000	0.000	<0.001	1.000	1.000	<0.001	1.000	1.000 ↑	<0.001	1.000	1.000 =	<0.001	1.000	1.000

6.6.1 Experimental Setup and Methodology

The experiments in this section share some traits with those discussed in the previous parts of this chapter (§6.4 and §6.5.3).

Workflow. We adopt a similar experimental protocol across both of our evaluations, organised in three phases.

- *Baseline.* We start with network data (i.e., NetFlow with known ground truth—benign and malicious) captured in a given environment. This data is used to train ML models for NID, ensuring they achieve state-of-the-art performance.
- *Adversarial.* Then, we launch attacks entailing our adversarial host-space perturbations and show that the resulting NetFlows evade our baseline ML models. Next, we show that by "adversarially training" [103] our baselines using adversarial data generated with ConCap, the models become more resilient to such evasion attempts.
- *Open World.* We show that malicious NetFlow generated via ConCap can be used to train ML-NIDS that generalize to real-world "variants" of the same attacks.

More low-level details are found in the following subsections.

Data and Networks. We describe the data collection process for our experiments (we make all data available in our repository [25]) and more dataset details in Appendix 6.B.

- *Real-world network.* We use the same environment as in the experiment described in §6.5.3. Recall that, in this network, we captured ∼5GB of packets generated by ∼50 physical hosts; among these, one launching the patator ssh-bruteforce attack (with both the -P=1 and -P=0 options) against another host running a vanilla OpenSSH server. We configure ConCap to execute similar scenarios, ensuring that the "attacker" and "target" host, as well as the underlying conditions, resemble those of the real-world network.
- *Benchmark dataset.* We consider CICIDS17 and its extension CICIDS18 [9], due to the widespread popularity of these datasets in research, representing a valid setup for benchmarking. Both of these datasets contain traffic related to the patator attack, enabling a fair comparison with the real-world experiment. Importantly, we use the fixed version of these datasets (see [6]).

We set up ConCap to carry out the same attack scenarios as in the real-world experiment (i.e., by launching $\mathrm{patator}$ with -$\mathrm{P{=}0}$ and -$\mathrm{P{=}1}$), and also replicate the specifications of the corresponding hosts in CICIDS17/18 (according to the documentation).

We will also consider "completely new" attacks (in §6.7.2).

Developing the ML-NIDS. We develop our ML models by following best practices established in recent research [2, 4, 5]. To ensure fair comparisons between the real-world and benchmark experiments, we use NetFlows generated by the fixed version of CICFlowMeter [5], since the CICIDS17/18 datasets are provided in this format. Then, for each experiment, we train various ML models reliant on well-known classification algorithms used by prior work [2, 8, 22]: XGBoost (XGB), Random Forest (RF), Logistic Regression (LR), Deep Neural Network (DNN), Support Vector Machine (SVM). We use an 80:20 split for train:test and repeat the experiments 5 times to mitigate biased selection of samples and ensure statistical robustness [2].[10] We measure the performance via the true positive rate (tpr) to assess detection capability and the false positive rate (fpr) to gauge false alarm frequency. We also record the operational runtime of each model (see Appendix 6.E). More details on the composition of the training and test sets are provided in the corresponding section.

6.6.2 Using ConCap with Real-world Data

We begin with the experiments in the real world since we will use their results (reported in Table 6.2) as a scaffold for the experiments discussed in the next subsection.

Baseline. We take the PCAP trace captured in the real-world network and generate the corresponding NetFlows: we are certain that no attack occurs during this capture, and hence can safely label all such NetFlows as benign. Then, we take the PCAP traces captured on the "target" host, containing the malicious traffic generated by the "attacker" host (via both the $\mathrm{patator}$ -$\mathrm{P{=}1}$ and $\mathrm{patator}$ -$\mathrm{P{=}0}$); we remove any packet unrelated to this "attacker" host or directed at a port different from 22 (the one used by OpenSSH by default). We generate the corresponding NetFlows and label them as malicious. We train and test all our models (DT, RF, XGB, SVM, DNN) on benign samples and on either set of malicious samples (with $\mathrm{patator}$ -$\mathrm{P{=}1}$ or $\mathrm{patator}$ -$\mathrm{P{=}0}$). From Table 6.2, we can see that these models exhibit a near-perfect tpr and tnr.

Adversarial. We test the baseline models on malicious samples stemming from our host-space perturbations. In other words, we test the models trained on $\mathrm{patator}$ -$\mathrm{P{=}1}$ against NetFlows generated by issuing $\mathrm{patator}$ -$\mathrm{P{=}0}$, and vice-versa. The results in Table 6.2 (red cells) show that only two baseline models (out of five) trained on -$\mathrm{P{=}0}$ can still detect -$\mathrm{P{=}1}$, whereas models trained on -$\mathrm{P{=}1}$ cannot detect -$\mathrm{P{=}0}$ (albeit the SVM still has a suboptimal tpr=0.49). In other words, *by changing a single digit, it is possible to perfectly evade eight out of 10 of our baseline*

[10] Across all our experiments, we do not "data snoop' [4] to prevent inflating the performance of our ML models—we have no incentive to do so, as our goal is not to outperform prior work. We maintain a clear separation between training and test data, never evaluating on samples seen during training. Moreover, since our data is collected over a short timeframe, a temporal split is unnecessary. Prior empirical studies on similar testbeds have shown it yields no significant benefit [2], and it may even degrade performance [104].

models which exhibited perfect detection performance against the same variant of the corresponding attack. Then, we enhance the baseline models by training them on the "adversarial" variants of the malicious samples but generated by ConCap (e.g., we take the models trained on data from patator -P=1 in the real-world network, and retrain them on data stemming from patator -P=0 generated by ConCap). We then test such "adversarially trained" models on the corresponding real-world samples. The results in Table 6.2 (green cells) show that, *by using data generated by* ConCap, our models are now robust against the real-world attack.

Open world. We repeat the same procedure as in the "baseline" assessment, but instead of using malicious samples from the real world in the training set, we only use samples by ConCap. For instance, we take 80% of the benign NetFlows (from the real-world), all of the malicious NetFlows (stemming from patator -P=1) generated on ConCap, and test the resulting model on all of the malicious NetFlows of the same attack (i.e., patator -P=1) generated in the real-world setup (and on the remaining 20% benign samples from the real world). The results in Table 6.2 show a remarkable performance, indicating that ConCap can be used to develop ML models that withstand the corresponding attack—without having seen any sample "from the real world" of such an attack.[11]

6.6.3 Using ConCap with Benchmark Datasets

We show that ConCap allows to produce "new knowledge" by relying only on benchmark datasets. To this end, we follow a similar procedure as that of the real-world experiments (§6.6.2), showing that we reach the same conclusions—as it can be gleaned by the results in Table 6.6 (in Appendix 6.E).

Baseline. We treat our two considered datasets as two separate environments (following the recommendation of [2]). Notably, CICIDS17/18 have (labeled) data pertaining to a variety of malicious activities (13 malicious classes for CICIDS17, and 15 for CICIDS18). To align these experiments with those in §6.6.2, we only consider the patator NetFlows (generated with -P=1). Therefore, for each dataset, we train and test our ML models on its benign and malicious NetFlows. The results in Table 6.6 show that all our ML models perform well, and achieve performance that matches that of prior work [2, 6, 8], thereby validating our setup. Moreover, our baselines can correctly detect patator -P=1 generated via ConCap.

Adversarial. The CICIDS17 or CICIDS18 datasets do not include any traffic related to patator -P=0, and we are unable to generate new traffic within their original network environments. Hence, for these experiments, we rely exclusively on ConCap—a valid approach given our earlier demonstration (§6.4) showing that ConCap can produce traffic closely resembling that of real-world networks. We begin by testing our baseline ML models, originally trained on patator

[11] **Validation with additional real-world benign data.** Given that ML-NIDS are known to generate false positives [36], we collected another PCAP trace (of ∼17GB) having (different) traffic from the same network used in the real-world experiment in §6.6.2. We tested our models (both baseline and adversarially trained) on the NetFlows of this trace: our models still exhibited near-zero fpr, demonstrating their quality.

-P=1, against NetFlows generated by ConCap for patator -P=0. The results (highlighted in the red in Table 6.6) show that across both datasets only one baseline model—XGB trained on CICIDS17—is "robust" against to this host-space perturbation. All other models suffer a significant drop in tpr. Next, we adversarially retrain each model using the patator -P=0 NetFlows generated by ConCap and evaluate them on slightly different patator -P=0 samples (also generated by ConCap) to avoid train-test overlap. The results (green cells in Table 6.6) match our findings from the real-world experiment: all ML models are now robust against the patator -P=0 variants.

Open world. We train new ML models using 80% of the available benign data (for each dataset) and on all the malicious NetFlows generated through ConCap by launching patator with both -P=0 and -P=1 options. We test such models on the malicious NetFlows generated by running patator -P=1 in the real-world network setup of CICIDS17 and CICIDS18 and on the remaining 20% of benign data. The results (in Table 6.6) align with the real-world experiment's results.

Takeaway. We derive two lessons learned. **(1)** ConCap can be used as a "sandbox" to generate network data for realistic experiments on NIDS—including host-space perturbations. **(2)** Perturbations in the "host space" can have a high impact against state-of-the-art ML models for NIDS—and, in practice, attackers only require to change some options.

Further Validation: experiments on larger models. We have carried out additional experiments using all classes of CICIDS17 (leading to "larger" baseline models). Our findings remain consistent in this setting as well.

Specifically, we first train our models (DT, HGB, DNN, SVM, RF) on 80% of the benign and all malicious samples from CICIDS17, evaluating them on the remaining 20%. Their performance matches that of state-of-the-art detectors [5] with fpr <0.1% and tpr >99.9%, except for SVM which has tpr of 99.7%. Then, we test these models on samples of patator P=0 (not included in CICIDS17) generated by ConCap. The majority of these samples evade detection: the tpr is <1% for all models (except 7.3% for RF and 21.4% for DT). Then we adversarially train these models using the ConCap generated samples: the resulting models maintain a high tpr and low fpr and, importantly, all models now have tpr=100% for patator P=0. Finally, we test models trained on benign samples from CICIDS17 and malicious patator P=1 and P=0 generated by ConCap (i.e., the same ones used for the open world assessment) on all malicious samples of CICIDS17. We find that only the samples of patator are correctly classified as malicious out of all 13 malicious classes. This is expected because all other malicious classes entail attacks that differ substantially from patator (e.g., DoS attacks).

6.7 Discussion and Critical Analyses

We review our contributions (§6.7.1), describe additional experiments (§6.7.2), and discuss our major findings (§6.7.3).

6.7.1 Scope, Limitations and Future Work

We advance the state of the art by: *(i)* providing a tool, ConCap, to foster future research on NID; and *(ii)* revealing a blind spot within extant research in ML-NIDS' security. Our contributions are aimed at *research*: real-world deployments are outside our scope (as we clearly remarked in §6.2).

Our extensive experiments demonstrated that ConCap allows a researcher to carry out meaningful evaluations in the NID context. Through ConCap it is possible to produce new knowledge (or test new hypotheses) without the need (and risk) to carry out experiments in real-world networks: although we did rely on real-world assessments, such assessments primarily served to prove the utility of ConCap for this specific purpose. **Future research** can use ConCap to investigate open problems in NIDS [105, 54], such as: robustness to concept drift [104, 98], explainability [106, 35], false alarms [36, 107, 37, 108], or development of novel detection techniques (not necessarily relying on NetFlows, such as [39, 109, 110, 111]) that improve current NIDS. ConCap can also be used for fine-grained generation of *benign* labeled traffic—useful for applications orthogonal (or ancillary) to network security (e.g., network traffic classification [112, 113, 114, 115, 116])

Nonetheless, ConCap presents some **limitations**. For instance, its traffic is not fully deterministic (see Appendix 6.E.1): while this property can be a strength (e.g., by running the same scenario multiple times, it is possible to collect more "realistic" data) it can also be a weakness (e.g., for reproducibility of some experiments). However, we observe that such a limitation is expected due to the non-deterministic nature of modern networks/protocols (§6.4.2): even the authors of DetGen [52] have acknowledged that their tool is not deterministic [84].

6.7.2 Extra Experiments (New Attacks & Data)

We carry out experiments to expand our contributions—including generating a completely new labeled dataset.

Approximating host-space perturbations. We posited that some host-space perturbations are tough to reproduce by operating in the feature space (§6.5.3). We tested this with an experiment. We considered the use-case shown in Fig. 6.5, suggesting that launching patator with -P=1 leads to 6x more packets (w.r.t. patator -P=0) being exchanged. We simulated this by taking the NetFlows generated by patator -P=0 (and patator -P=1) in the real-world network, and multiplying (dividing) the number of packets by 6; we also did so for the number of bytes and updated all other dependent features, ensuring a correct feature vector. We use these NetFlows, allegedly representing patator -P=1 (and -P=0) in two ways:

- Submit them to ML models trained on the "real" patator.
- Use them to "adversarially train" ML models, and then test such ML models on "real" samples of patator.

In theory, if these "fictitious" NetFlows resemble that of the "real" patator, we should obtain

the same results achieved in §6.6.2: a perfect tpr (see Table 6.2). However, we found that this is not the case: for three models (DT, HGB, RF) out of five, the tpr is *always* 0. This indicates that such NetFlows do not resemble those of the "real" $\mathrm{patator}$ (either -$\mathrm{P{=}0}$ or -$\mathrm{P{=}1}$), showcasing that using such "adversarial" NetFlows for research experiments would lead to misleading results. This is why we recommend using ConCap to carry out such simulations—instead of trying to approximate the effects of a host-space perturbation by directly operating in the feature space. (This is just an attempt at approximating our host-space perturbation: future work may use ConCap to find ways to precisely map a host-space perturbation in the feature space.)

Additional attack-simulations with ConCap. In our "adversarial" experiments, we focused on two variants of $\mathrm{patator}$ by changing between $\mathrm{P{=}1}$ and $\mathrm{P{=}0}$. However, there are other ways to introduce host-space perturbations that lead to the same goal (i.e., an ssh-bruteforce attack). We hence expanded our assessment: we took the baseline models trained on CICIDS17 and CICIDS18, and tested them against attacks carried out via different ssh-bruteforcing tools, such as Hydra [117] and Medusa [118]; we also considered changing the conditions of the testbed (e.g., changing OS, adding network delays, or packet loss) by relying on the customisability of ConCap. These experiments are described in Appendix 6.E.2 (where we also provide additional realistic considerations), and the results are provided in Table 6.5. We find that these additional host-space perturbations can be quite effective (e.g., the tpr against Medusa is above 0.33 for only one classifier on each dataset); whereas using the same attack ($\mathrm{patator}$) in different network conditions can also lead to different responses from the detectors (albeit the tpr is still high).

Creating new data with ConCap. Among the benefits of ConCap is the possibility of creating entirely new malicious and correctly labeled network traffic data. We leveraged this property to create a new dataset containing NetFlows conforming to three recent network-related attacks *that are not contained in any existing benchmark dataset* (according to [8]). Specifically, we considered three exploits (from VulnHub): CVE-2024-47177 [119] (related to OpenPrinting Cups), CVE-2024-36401 [120] (related to GeoServer), CVE-2024-2961 [121] (related to GNU C Library). We setup ConCap accordingly, so that the target host can be "exploited" via the commands included in the CVE and executed on the attacker host. To provide broader coverage, we reproduce these attacks by simulating various network conditions, and we also introduce different host-space perturbations (by slightly changing the payload of the attack). More details are in Appendix 6.B.3. We report the statistics of our dataset in Table 6.3: future work can use our labeled NetFlows (or the corresponding PCAP traces) and use them to assess/develop novel NIDS (not necessarily reliant on NetFlows).

6.7.3 Implications of our Findings

A recent work claimed that "real attackers do not compute gradients" [55]. Our findings in §6.6 support this claim, suggesting it may be well founded.

It is true that, by using gradient-based strategies, it is possible to evade an ML-NIDS. Indeed, in our literature review (§6.5.2.2), hundreds of papers used similar strategies to bypass ML-NIDS.

However, as we have argued (§6.5), applying such perturbations is tough from the perspective of an attacker who does not have admin access to the highly-secure hosts (router/switch or the ML-NIDS itself) of the targeted network—which is the common case [14]. Yet, our results show that attackers can bypass ML-NIDS simply by adjusting a command-line option. We posit attackers are more likely to rely on such simple strategies to bypass real-world NIDS.[12]

Nonetheless, we highlight the approach proposed by Catillo et al. [13], which explores "problem-space perturbations" by introducing delays through modifications to the attacker's host network configuration using Traffic Control. However, this approach—reproducible via ConCap (§6.3.3)—requires the attacker to have admin power on the host, and has the drawback of affecting *all* traffic between this specific host and the target—including the "benign" communications.

Critical remarks. A reader may raise the three following points, which we address in Appendix 6.F. **(1)** *The experiments only involve simple and well-known attacks, such as brute-force or port-scans.* **(2)** *One can easily manipulate and send "perturbed" packets via scapy and tcpreplay, therefore host-space perturbations are not the only way that attackers can use to evade ML-NIDS.* **(3)** *The technical contribution of* ConCap *is only a trivial application of Kubernetes.*

6.8 Conclusions

This chapter is a stepping stone for future research in NID.

With ConCap, we enable researchers—especially those without access to real-world networks—–to conduct realistic security assessments in networking contexts. Our new dataset is proof of this, since it was created by running recent exploits in a safe environment which mimics a realistic network.

The potential impact of our envisioned "host-space adversarial perturbations" should serve as a call to action. Real attackers favor cheap and simple strategies. Future research should shed more light on this yet another security risk of ML-NIDS. Such analyses are now accessible to any researcher—thanks to our proposed ConCap.

[12] **Disclaimer.** This is not a finger-pointing exercise to invalidate prior research: the perturbations of prior work *can* represent valid attacks. Our intention is to cast light on a different (practical) way to bypass ML-NIDS which had not been investigated. Thanks to ConCap, this can now be tested.

Bibliography

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, 2019.

[2] G. Apruzzese, P. Laskov, and J. Schneider, "Sok: Pragmatic assessment of machine learning for network intrusion detection," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2023.

[3] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, 2021.

[4] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *USENIX Security*, 2022.

[5] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 7–12.

[6] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, "Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018," in *IEEE Conference on Communications and Network Security*. IEEE, 2022.

[7] M. Catillo, A. Pecchia, and U. Villano, "Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS?" in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume*, 2023.

[8] R. Flood, G. Engelen, D. Aspinall, and L. Desmet, "Bad design smells in benchmark nids datasets," in *IEEE 9th European Symposium on Security and Privacy (EuroSP)*, 2024.

[9] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, 2018.

[10] C. G. Cordero, E. Vasilomanolakis, A. Wainakh, M. Mühlhäuser, and S. Nadjm-Tehrani, "On generating network traffic datasets with synthetic attacks for intrusion detection," *ACM Transactions on Privacy and Security*, 2021.

[11] P. Bönninghausen, R. Uetz, and M. Henze, "Introducing a comprehensive, continuous, and collaborative survey of intrusion detection datasets," in *Cyber Security Experimentation and Test Workshop*, 2024.

[12] R. Sheatsley, B. Hoak, E. Pauley, Y. Beugin, M. J. Weisman, and P. McDaniel, "On the robustness of domain constraints," in *ACM SIGSAC conference on computer and communications security*, 2021.

[13] M. Catillo, A. Pecchia, A. Repola, and U. Villano, "Towards realistic problem-space adversarial attacks against machine learning in network intrusion detection," in *International Conference on Availability, Reliability and Security*, 2024.

[14] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, "Modeling realistic adversarial attacks against network intrusion detection systems," *ACM Digital Threats: Research and Practice*, 2021.

[15] I. Arnaldo and K. Veeramachaneni, "The holy grail of "systems for machine learning" teaming humans and machine learning for detecting cyber threats," *ACM SIGKDD Explorations Newsletter*, 2019.

[16] G. Apruzzese, L. Pajola, and M. Conti, "The cross-evaluation of machine learning-based network intrusion detection systems," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 5152–5169, 2022.

[17] J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, "A survey on network simulators, emulators, and testbeds used for research and education," *Computer Networks*, 2023.

[18] G. Apruzzese et al, "The role of machine learning in cybersecurity," *ACM Digital Threats: Research and Practice*, 2022.

[19] D. Han, Z. Wang, Y. Zhong, W. Chen, J. Yang, S. Lu, X. Shi, and X. Yin, "Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors," *IEEE Journal on Selected Areas in Communications*, 2021.

[20] J. Wang, L. Qixu, W. Di, Y. Dong, and X. Cui, "Crafting adversarial example to bypass flow-&ml-based botnet detector via rl," in *International symposium on research in attacks, intrusions and defenses*, 2021.

[21] D. Wu, B. Fang, J. Wang, Q. Liu, and X. Cui, "Evading machine learning botnet detection models via deep reinforcement learning," in *IEEE International Conference on Communications*, 2019.

[22] D. Bhusal, M. T. Alam, M. K. Veerabhadran, M. Clifford, S. Rampazzi, and N. Rastogi, "Pasa: Attack agnostic unsupervised adversarial detection using prediction & attribution sensitivity analysis," in *IEEE European Symposium on Security and Privacy*, 2024.

[23] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *ICLR*, 2017.

[24] "ssh-patator," https://github.com/lanjelot/patator.

[25] "Repository of this paper," https://anonymous.4open.science/r/Host-Space-Adversarial-Attacks-923B/.

[26] G. Vormayr, J. Fabini, and T. Zseby, "Why are my flows different? a tutorial on flow exporters," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2064–2103, 2020.

[27] L. Dias, S. Valente, and M. Correia, "Go with the flow: Clustering dynamically-defined netflow features for network intrusion detection with dynids," in *IEEE International Symposium on Network Computing and Applications*, 2020.

[28] ENISA, "Enisa threat landscape," ENISA, Tech. Rep., 2023. [Online]. Available: https://www.enisa.europa.eu/topics/cyber-threats/threats-and-trends

[29] CloudFlare, "DDoS threat report for 2023 Q4," CloudFlare, Tech. Rep., 2024. [Online]. Available: https://blog.cloudflare.com/ddos-threat-report-2023-q4/

[30] C. Crowley, "Sans 2024 soc survey: Facing top challenges in security operations," SANS Research Program, Tech. Rep., 2024. [Online]. Available: https://newsletter.radensa.ru/wp-content/uploads/2024/07/SANS-2024-SOC-Survey.pdf

[31] Comodo, "Difference between siem and ids," Tech. Rep., 2023. [Online]. Available: https://web.archive.org/web/20240829025823/https://www.comodo.com/difference-between-siem-and-ids.php

[32] PaloAlto, "What is the difference between edr vs. siem?" Tech. Rep., 2024. [Online]. Available: https://web.archive.org/web/20240829025838/https://www.paloaltonetworks.com/cyberpedia/what-is-edr-vs-siem

[33] E. Bertino and I. Karim, "Ai-powered network security: Approaches and research directions," in *Proceedings of the 8th International Conference on Networking, Systems and Security*, 2021.

[34] N. Kshetri, "Economics of artificial intelligence in cybersecurity," *IEEE IT Professional*, 2021.

[35] J. Mink, H. Benkraouda, L. Yang, A. Ciptadi, A. Ahmadzadeh, D. Votipka, and G. Wang, "Everybody's got ml, tell me what else you have: Practitioners' perception of ml-based security tools and explanations," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2068–2085.

[36] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of {SOC} analysts' perspectives on security alarms," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2783–2800.

[37] M. Vermeer, N. Kadenko, M. van Eeten, C. Gañán, and S. Parkin, "Alert Alchemy: SOC Workflows and Decisions in the Management of NIDS Rules," in *ACM CCS*, 2023.

[38] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, pp. 222–232, 1987.

[39] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *NDSS*, 2018.

[40] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *NDSS*, 2021.

[41] M. Piskozub, F. De Gaspari, F. Barr-Smith, L. Mancini, and I. Martinovic, "Malphase: Fine-grained malware detection using network flow data," in *Proceedings of the 2021 ACM Asia conference on computer and communications security*, 2021, pp. 774–786.

[42] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[43] R. Sommer and V. Paxson, "Enhancing byte-level network intrusion detection signatures with context," in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003.

[44] F. Araujo, G. Ayoade, K. Al-Naami, Y. Gao, K. W. Hamlen, and L. Khan, "Improving intrusion detectors by crook-sourcing," in *Annual Computer Security Applications Conference*, 2019.

[45] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, "Ai/ml for network security: The emperor has no clothes," in *ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[46] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Information sciences*, 2013.

[47] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*, 2010.

[48] T. Krauß, J. Stang, and A. Dmitrienko, "Verify your labels! trustworthy predictions and datasets via confidence scores," in *USENIX Security Symposium*, 2024.

[49] M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Towards model generalization for intrusion detection: Unsupervised machine learning techniques," *Journal of Network and Systems Management*, vol. 30, pp. 1–25, 2022.

[50] M. Sarhan, S. Layeghy, and M. Portmann, "Evaluating standard feature sets towards increased generalisability and explainability of ml-based network intrusion detection," *Big Data Research*, 2022.

[51] M. Catillo, A. Del Vecchio, A. Pecchia, and U. Villano, "Transferability of machine learning models learned from public intrusion detection datasets: the cicids2017 case study," *Software Quality Journal*, 2022.

[52] H. Clausen, R. Flood, and D. Aspinall, "Traffic generation using containerization for machine learning," in *Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security*, 2019.

[53] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, 2018.

[54] A. E. Cinà, K. Grosse, A. Demontis, B. Biggio, F. Roli, and M. Pelillo, "Machine learning security against data poisoning: Are we there yet?" *Computer*, 2024.

[55] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, and K. Roundy, ""Real Attackers Don't Compute Gradients": Bridging the Gap Between Adversarial ML Research and Practice," in *SaTML*, 2023.

[56] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2018.

[57] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *ECML PKDD*, 2013.

[58] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *IEEE Symposium on security and privacy (SP)*, 2020.

[59] G. Apruzzese and M. Colajanni, "Evading botnet detectors based on flows and random forest with adversarial samples," in *IEEE International Symposium on Network Computing and Applications (NCA)*, 2018.

[60] G. Apruzzese, M. Conti, and Y. Yuan, "Spacephish: The evasion-space of adversarial attacks against phishing website detectors using machine learning," in *Proc. ACSAC*, 2022.

[61] F. Suya, A. Suri, T. Zhang, J. Hong, Y. Tian, and D. Evans, "Sok: Pitfalls in evaluating black-box attacks," in *2024 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE, 2024, pp. 387–407.

[62] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, 2019.

[63] N. Wang, Y. Chen, Y. Xiao, Y. Hu, W. Lou, and Y. T. Hou, "Manda: On adversarial example detection for network intrusion detection system," *IEEE TDSC*, 2022.

[64] A. M. Sadeghzadeh, S. Shiravi, and R. Jalili, "Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification," *IEEE Transactions on Network and Service Management*, 2021.

[65] A. M. Potdar, D. Narayan, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020.

[66] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, 2015.

[67] D. Moreau, K. Wiebels, and C. Boettiger, "Containers for computational reproducibility," *Nature Reviews Methods Primers*, vol. 3, no. 1, p. 50, 2023.

[68] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Concurrency and Computation: Practice and Experience*, 2020.

[69] T. L. Foundation, "Kubernetes - production-grade container orchestration," [Online]. Available: https://kubernetes.io/.

[70] Kubernetes, "Kubernetes github," [Online]. Available: https://github.com/kubernetes/kubernetes.

[71] O. Bentaleb, A. S. Belloum, A. Sebaa, and A. El-Maouhab, "Containerization technologies: Taxonomies, applications and challenges," *The Journal of Supercomputing*, vol. 78, no. 1, pp. 1144–1181, 2022.

[72] D. inc., "Docker: Accelerate how you build, share, and run applications," 2024, [Online]. Available: https://docker.com.

[73] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright, "Deepcoffea: Improved flow correlation attacks on tor via metric learning and amplification," in *IEEE S&P*, 2022.

[74] A. N. Kuznetsov, "tc(8) — linux manual." accessed on 28 May 2023. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html.

[75] R. Uetz, C. Hemminghaus, L. Hackländer, P. Schlipper, and M. Henze, "Reproducible and adaptable log data generation for sound cybersecurity experiments," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 690–705.

[76] R. Beltiukov, W. Guo, A. Gupta, and W. Willinger, "In search of netunicorn: A data-collection platform to develop generalizable ml models for network security problems," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2217–2231.

[77] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014.

[78] R. Beltiukov, S. Chandrasekaran, A. Gupta, and W. Willinger, "Pinot: Programmable infrastructure for networking," in *Proceedings of the Applied Networking Research Workshop*, 2023, pp. 51–53.

[79] P. I. Khan, S. Guthula, R. Beltiukov, R. Schmid, T. Bühler, A. Gupta, L. Vanbever, and W. Willinger, "Harnessing public code repositories to develop production-ready ml artifacts for networking," in *Proceedings of the 2024 Applied Networking Research Workshop*, 2024, pp. 100–102.

[80] T. Bühler, R. Schmid, S. Lutz, and L. Vanbever, "Generating representative, live network traffic out of millions of code repositories," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, 2022, pp. 1–7.

[81] G. Zhou, X. Guo, Z. Liu, T. Li, Q. Li, and K. Xu, "Trafficformer: An efficient pre-trained model for traffic data," in *IEEE Symposium on Security and Privacy 2025*, 2025.

[82] S. Guthula, N. Battula, R. Beltiukov, W. Guo, and A. Gupta, "netfound: Foundation model for network security," *arXiv preprint arXiv:2310.17025*, 2023.

[83] X. Sáez-de Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Gotham testbed: a reproducible iot testbed for security experiments and dataset generation," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[84] R. F. David Aspinall, Henry Clausen, "DetGen GitHub repository," https://github.com/detlearsom/DetGen.

[85] "nmap," https://nmap.org/.

[86] "danielmiessler/SecLists," https://github.com/danielmiessler/SecLists/tree/master.

[87] "wireshark," https://www.wireshark.org/.

[88] R. 813, "WINDOW AND ACKNOWLEDGEMENT STRATEGY IN TCP," MIT Laboratory for Computer Science, Tech. Rep., 1982. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc813

[89] R. 7323, "TCP Extensions for High Performance," Internet Engineering Task Force (IETF), Tech. Rep., 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7323

[90] R. 793, "Transmission Control Protocol," DARPA, Tech. Rep., 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc793

[91] R. 1122, "Requirements for Internet Hosts – Communication Layers," Network Working Group, Tech. Rep., 1989. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1122

[92] QoSient, "Argus NetFlow," https://qosient.com/argus/argusnetflow.shtml, 2024.

[93] G. Engelen, "Fixed CICFlowMeter," https://github.com/GintsEngelen/CICFlowMeter, 2012.

[94] T. E. Carroll, D. Manz, T. Edgar, and F. L. Greitzer, "Realizing scientific methods for cyber security," in *Proceedings of the 2012 Workshop on Learning from Authoritative Security Experiment Results*, 2012.

[95] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "E-graphsage: A graph neural network based intrusion detection system for iot," in *IEEE/IFIP Network Operations and Management Symposium*, 2022.

[96] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos:: Practical intrusion detection and investigation using whole-system provenance," in *IEEE Symposium on Security and Privacy*, 2024.

[97] W. Song, M. Beshley, K. Przystupa, H. Beshley, O. Kochan, A. Pryslupskyi, D. Pieniak, and J. Su, "A software deep packet inspection system for network traffic analysis and anomaly detection," *Sensors*, 2020.

[98] X. Wang, "Enidrift: A fast and adaptive ensemble system for network intrusion detection under real-world drift," in *Annual Computer Security Applications Conference*, 2022.

[99] M. Schneider, D. Aspinall, and N. D. Bastian, "Evaluating model robustness to adversarial samples in network intrusion detection," in *IEEE International Conference on Big Data (Big Data)*, 2021.

[100] M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, L. Shamseer, J. M. Tetzlaff, E. A. Akl, S. E. Brennan *et al.*, "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews," *BMJ*, vol. 372, 2021.

[101] M. Abdelaty, S. Scott-Hayward, R. Doriguzzi-Corin, and D. Siracusa, "Gadot: Gan-based adversarial training for robust ddos attack detection," in *IEEE Conference on Communications and Network Security*, 2021.

[102] G. Severi, S. Boboila, A. Oprea, J. Holodnak, K. Kratkiewicz, and J. Matterer, "Poisoning network flow classifiers," in *Annual Computer Security Applications Conference*, 2023.

[103] A. Shafahi, M. Najibi, M. A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!" *Advances in neural information processing systems*, 2019.

[104] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection," in *ACM workshop on artificial intelligence and security*, 2021.

[105] F. Ceschin, M. Botacin, A. Bifet, B. Pfahringer, L. S. Oliveira, H. M. Gomes, and A. Grégio, "Machine learning (in) security: A stream of problems," *ACM DTRAP*, 2024.

[106] D. Bhusal, R. Shin, A. A. Shewale, M. K. M. Veerabhadran, M. Clifford, S. Rampazzi, and N. Rastogi, "Sok: Modeling explainability in security analytics for interpretability, trustworthiness, and usability," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, 2023, pp. 1–12.

[107] T. Van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna, "Deepcase: Semi-supervised contextual analysis of security events," in *IEEE Symposium on Security and Privacy (SP)*, 2022.

[108] C. Fu, Q. Li, K. Xu, and J. Wu, "Point cloud analysis for ml-based malicious traffic detection: Reducing majorities of false positive alarms," in *ACM CCS*, 2023.

[109] M. Sharif, P. Datta, A. Riddle, K. Westfall, A. Bates, V. Ganti, M. Lentzk, and D. Ott, "Drsec: Flexible distributed representations for efficient endpoint security," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3609–3624.

[110] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 139–139.

[111] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "{PROGRAPHER}: An anomaly detection system based on provenance graph embedding," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[112] A. Azab, M. Khasawneh, S. Alrabaee, K.-K. R. Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," *Digital Communications and Networks*, 2024.

[113] B. Cebere and C. Rossow, "Understanding web fingerprinting with a protocol-centric approach," in *RAID*, 2024.

[114] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *USENIX NSDI*, 2022.

[115] V. Rimmer, T. Schnitzler, T. Van Goethem, A. Rodríguez Romero, W. Joosen, and K. Kohls, "Trace oddity: Methodologies for data-driven traffic analysis on tor," *PETS*, 2022.

[116] S. Schäfer, A. Löbel, and U. Meyer, "Accurate real-time labeling of application traffic," in *IEEE LCN*, 2022.

[117] "Hydra," https://github.com/vanhauser-thc/thc-hydra.

[118] "Medusa," https://github.com/jmk-foofus/medusa.

[119] VulHub, "CVE-2024-47177," https://github.com/vulhub/vulhub/blob/master/cups-browsed/CVE-2024-47177, 2024.

[120] ——, "CVE-2024-36401," https://github.com/vulhub/vulhub/blob/master/geoserver/CVE-2024-36401, 2024.

[121] ——, "CVE-2024-2961," https://github.com/vulhub/vulhub/tree/master/php/CVE-2024-2961, 2024.

[122] G. Apruzzese, P. Laskov, and A. Tastemirova, "Sok: The impact of unlabelled data in cyberthreat detection," in *EuroS&P*, 2022.

[123] L. D'hooge, M. Verkerken, B. Volckaert, T. Wauters, and F. De Turck, "Establishing the contaminating effect of metadata feature inclusion in machine-learned network intrusion detection models," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2022, pp. 23–41.

[124] F. Woitschek and G. Schneider, "Physical adversarial attacks on deep neural networks for traffic sign recognition: A feasibility study," in *IEEE Intelligent vehicles symposium (IV)*, 2021.

[125] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *IEEE conference on computer vision and pattern recognition*, 2018.

[126] R. Song, M. O. Ozmen, H. Kim, R. Muller, Z. B. Celik, and A. Bianchi, "Discovering adversarial driving maneuvers against autonomous vehicles," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 2957–2974.

[127] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, 2014.

[128] K. Lucas, W. Lin, L. Bauer, M. K. Reiter, and M. Sharif, "Training robust ml-based raw-binary malware detectors in hours, not months," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 124–138.

[129] R. Liu, Y. Lin, X. Yang, S. H. Ng, D. M. Divakaran, and J. S. Dong, "Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach," in *USENIX Security Symposium*, 2022.

[130] Q. Hao, N. Diwan, Y. Yuan, G. Apruzzese, M. Conti, and G. Wang, "It doesn't look like anything to me: Using diffusion model to subvert visual phishing detectors," in *USENIX Sec*, 2024.

[131] P. Kotzias, K. Roundy, M. Pachilakis, I. Sanchez-Rola, and L. Bilge, "Scamdog millionaire: Detecting e-commerce scams in the wild," in *ACSAC*, 2023.

Appendix

6.A ConCap Configuration

We provide information for practical use of ConCap. First, the full configuration files for a ConCap scenario and NetFlow extractor are given with all the possible configuration options. Then, a step-by-step guide is provided to replicate a scenario from DetGen [52].

6.A.1 Scenario Configuration File

A ConCap scenario is the blueprint for the traffic generation process and allows for flexible configuration. Listing 6.1 presents an exemplary scenario definition file for a full Nmap port scan against an Apache webserver with all the possible configuration options.

6.A.2 NetFlow Exporter Configuration

Automatic NetFlow extraction in ConCap is set-up by processing configuration files. A flow extractor is created for each file, which exports NetFlows from the scenario's network traces. An example configuration file in Listing 6.2 for *Argus* has 3 configuration options: a "name", "containerImage", and "command". The name and container image are used to deploy the NetFlow exporter container. The command is responsible for processing the network capture file and outputting the extracted network flows as a CSV file.

Listing 6.1: A ConCap scenario configuration file describing a full port scan via nmap against an Apache webserver.

```
attacker:
  name: nmap
  image: instrumentisto/nmap:latest
  atkCommand: nmap $TARGET_IP -A -T4
  atkTime: 30s
  cpuRequest: 100m
  memRequest: 100Mi
  cpuLimit: 500m
  memLimit: 500Mi
target:
  name: httpd
  image: httpd:latest
  filter: "((dst host $ATTACKER_IP and src host $TARGET_IP) or (dst
       host $TARGET_IP and src host $ATTACKER_IP)) and not arp"
  cpuRequest: 100m
  memRequest: 100Mi
  cpuLimit: 500m
  memLimit: 500Mi
network:
  bandwidth: 100mbit
  queueSize: 100ms
  limit: 10000
  delay: 0ms
  jitter: 0ms
  distribution: normal
  loss: 0%
  corrupt: 0%
  duplicate: 0%
  seed: 0
labels:
  label: 1
  category: port-scan
  subcategory: nmap
  scenario: nmap_A_T4
```

6.A.3 Configuration of a DetGen Scenario in ConCap

To demonstrate the flexibility of ConCap, we have created a step-by-step guide to implement one of the predefined scenarios from DetGen [52]. We selected the well-documented "capture-020-nginx" scenario, which uses "siege", an HTTP load testing and benchmarking tool, to target the "nginx" HTTP server and reverse proxy. In this guide, we show how to replicate the "capture-020-nginx" scenario in ConCap using processing pods and scenario definitions.

NetFlow Configuration DetGen does not support automated NetFlow generation, thus we skip the processing pods.

Scenario Configuration The ConCap scenario is detailed in Listing 6.3. Here, the attacker is configured to use *siege* for 10 seconds with 10 simulated users, each making 1,000 requests to

Listing 6.2: Argus Processing Definition for ConCap

```
name: argus
containerImage: ghcr.io/anonymous/concap/argus:latest
command: "argus -r $INPUT_FILE -S 60s -w - | ra -r - -c, >
    $OUTPUT_FILE"
```

Listing 6.3: The replicated capture-020-nginx scenario in ConCap.

```
attacker:
  name: siege
  image: ghcr.io/anonymous/concap/siege:ubuntu18
  atkCommand: siege -c 10 -r 1000 -v http://$TARGET_IP
  atkTime: 10s
target:
  name: nginx
  image: nginx:1.13.8-alpine
```

the reverse proxy's index page. The IP address of the target is assigned through environment variable expansion. Since there is no official Docker image for *siege*, we could have opted for one of the many community-built images. However, to demonstrate the simplicity of creating a custom image, we provide a minimal Dockerfile in Listing 6.4, which we then push to our GitHub Container Registry. For the target, we use the official *nginx* DockerHub image. To execute the scenario twice, we duplicate the scenario definition file.

6.B Datasets (existing and new ones)

We provide details on the various "datasets" considered in this chapter. Specifically, we first provide information on the real-world capture (used in §6.5.3 and §6.6.2), then we describe how we preprocessed the benchmark datasets CICIDS17 and CICIDS18 (used in §6.6.3), and finally we provide low-level details of our "new" dataset containing labeled NetFlows related to attacks not contained in currently available benchmarks (mentioned in §6.7.2).

6.B.1 Real-word Network Capture: Description

For some of our experiments (in §6.5.3 and §6.6.2), we used data captured in a real-world network. Here, we provide more details of this network environment, and the captured PCAP trace and corresponding NetFlow data.

Network Overview The network environment encompasses 40–50 physical devices. Such devices entail: smartphones, laptops / desktops, gaming consoles; as well as various IoT devices (smart speakers, lightbulbs) and media appliances (e.g., smart TV). All these devices are connected to a router through a WiFi 5 or 2.4 interface. The router is connected to the internet through a 50Mbps

Listing 6.4: A minimal container image for running *siege*.

```
FROM ubuntu:18.04
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update && \
    apt-get -y install siege && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists

ENTRYPOINT ["siege"]
```

download speed and 5Mbps upload speed. The devices within the network are kept up-to-date with security patches and their owners are security experts, hence it is safe to assume that the traffic is "benign" (and even if some traffic is "malicious", it is of a different class than patator meaning that our conclusions are not affected by such circumstances). With regards to the devices used to simulate the attack in this network, they were two laptops both running Ubuntu 18.04; the "target" mounts an Intel i7 7700H with 32GB of RAM; the "defender" mounts an Intel N4100 with 8GB of RAM.

Network Traffic (PCAP). Three sets of network captures were performed, two for benign background traffic and one related to the patator ssh-bruteforce attacks. The background traffic was first captured on the 6th of November 2023 and a second time on the 26th of August 2024, the same day the malicious attacks were executed and captured. The total file size for the benign traffic of the August 26th capture is of 6GB for 8M packets, whereas the attack captures only measured 45MB for 241k packets. For the (benign) trace captured on November 2023, the size is of 17GB for 17M packets

NetFlow. NetFlows were extracted from the network captures using CICFlowMeter. The benign capture from November 2023 contained 30,304 unique NetFlows, while the benign capture from August 2024 contained 49,188 unique NetFlows. The malicious capture, on the other hand, included a total of 7,990 unique NetFlows.

6.B.2 Preprocessing of Benchmark datasets

In this appendix, we provide an overview of the data preprocessing steps undertaken to prepare the network traffic data for analysis. This includes the extraction of NetFlow features using the latest version of CICFlowMeter, data cleaning procedures, and a description of the real world network captures used in our experiments.

NetFlow Extraction The fixed versions of the datasets CICIDS17 and CICIDS18 were released in October 2022. Since then, CICFlowMeter, the tool used to extract the NetFlow features from the network traffic traces, has received over 30 new commits fixing, changing, and adding NetFlow features. To benefit from these updates over the last two years, we replicated the work done by

Liu et al. [6] by extracting the NetFlows using the current version (Aug '24) of CICFlowMeter and labeling the flows based on their fixed logic. The "attempted" NetFlows are removed from our dataset.

Data Cleaning. The dataset cleaning steps on the NetFlows performed in our experiments follow the best practices described in previous work [122, 4, 123]. First, the meta-data and spurious features are removed: "id", "Flow ID", "Src IP", "Dst IP", "Timestamp", "FWD Init Win Bytes", and "Bwd Init Win Bytes". Second, the source and destination ports are mapped to their IANA port categories and subsequently encoded as 0, 1, or 2 for respectively *well-known*, *registered*, and *dynamic*. Third, all NetFlows with missing values are removed. Last, all duplicate NetFlows are removed.

Implementation. The features and hyperparameters used to develop our ML models are provided in our repository.

6.B.3 New Data with ConCap

We generated new labeled "malicious" data using ConCap by executing three recent CVE-based attack scenarios, ensuring none of them overlap with the datasets mentioned in Flood et al. [8]. These CVEs represent diverse vulnerabilities, each exploited to demonstrate the flexibility of ConCap in handling modern attacks.

- **CVE-2024-47177** [119]: This vulnerability affects OpenPrinting Cups-Browsed versions 2.0.1 and earlier, where improper handling of the FoomaticRIPCommandLine parameter in PostScript Printer Description (PPD) files allows remote code execution. Attackers can exploit this by creating a malicious IPP (Internet Printing Protocol) server that sends crafted printer information to a vulnerable Cups-Browsed instance, enabling arbitrary command execution on the affected system.
- **CVE-2024-36401** [120]: In GeoServer versions prior to 2.25.1, 2.24.3, and 2.23.5, unauthenticated remote code execution is possible through unsafely evaluated property name expressions. Specially crafted input can exploit these unsafe evaluations in multiple OGC request parameters, allowing attackers to execute arbitrary code on a vulnerable GeoServer installation.
- **CVE-2024-2961** [121]: This vulnerability in the GNU C Library (versions 2.39 and earlier) affects the iconv() function, which may overflow the output buffer by up to 4 bytes when converting strings to the ISO-2022-CN-EXT character set. Attackers can exploit arbitrary file read vulnerabilities in PHP applications to escalate to remote code execution by leveraging the iconv() issue, potentially crashing the application or executing code.

The target environments were constructed using Vulhub, an open-source collection of pre-built vulnerable docker environments. The attacker environments were built using official Docker images, combined with the necessary tools and exploit code for each CVE. These environments were then defined into a ConCap scenario together with the attack command and assigned a unique label with the corresponding CVE for automatic traffic labeling. The three scenarios, one for each CVE, are then repeated 320 times in different network environments with a unique set of values for *delay, loss, corrupt,* and *duplicate*. Each of these scenarios is repeated for five, two, and four host-space perturbations respectively for CVE-2024-47177, CVE-2024-36401, and CVE-2024-2961.

Table 6.3: **Summary of the new dataset generated via ConCap.** For each attack, we report: the number of packets and NetFlows generated; the number of network states considered (for generalizability) and the number of Host-space Perturbations (HsP).

Attack	#Packets	# NetFlows	States	# HsP
CVE-2024-47177	44 658	4800	320	5
CVE-2024-36401	10 372	640	320	2
CVE-2024-2961	391 350	1280	320	4

6.C Considerations on the "Spaces" of Perturbations

To further illustrate the major differences between "feature" and "problem" space, we describe three exemplary scenarios. These scenarios envision domains that, despite being orthogonal to NIDS, enable a better understanding of the crux tackled by our work.

Attacks vs Autonomous Driving Cars

Let us assume a modern car that relies on ML models to analyse the surrounding environment so that proper driving decisions can be made. An exemplary use case is an ML model that must recognize *traffic signs*. In this case, the "feature space" is represented by the pixels of the images acquired by the sensors of the car (e.g., cameras), which are provided as input to the ML model to discern whether *(i)* the image contains a traffic sign, and *(ii)* if so, which sign it is.

To introduce a perturbation in such a feature space (i.e., direct pixel manipulation of the digitally-acquired image), the attacker must tamper with the software embedded in the car's system—which is doable, but impractical unless the attacker has compromised the systems of the car's manufacturer, or deliberately manipulated the system's of the targeted car [124].

The attacker can, however, also attempt to manipulate the traffic sign *in the physical world*. To do this, the attacker can, e.g., tamper with a specific traffic sign, so that whenever some sensors capture images of such a traffic sign, the resulting image will (ideally) fool the classification of the ML model [125, 126].[13]

Attacks vs Malware Detectors

Let us assume a malware detector that relies on ML to analyse some applications and determine whether they are malicious or not. An exemplary use case is the DREBIN detector for Android malware [127]: DREBIN receives as input a feature vector of thousands of binary features, each related to a specific functionality of the corresponding app (e.g., permissions).

[13] This example is useful to explain a "host-space" perturbation: instead of manipulating the image, which is acquired and processed by devices that are not under the attacker's control (i.e., a perturbation to a piece of data), the attacker creates the "perturbation" by interacting with the physical world (i.e., which, in our case, is the attacker's host).

To introduce perturbations in such a feature space, the attacker must have access to the preprocessing mechanism that extracts the feature vector (which will be fed as input to the ML model, i.e., DREBIN). Doing so means that the attacker has root access of either *(a)* the Android device executing the anti-malware app, or *(b)* of a remote security system. Both of these cases are possible, but assume a very powerful attacker who has already compromised the security system.

Alternatively, the attacker can *manipulate the source-code* of the (malicious) Android app, so that the resulting feature vector will have values that bypass the ML model [58, 128]. This is easier to do by an attacker, since they have complete control on the source code of the piece of malware that they develop; however, in doing so, the attacker must be careful not to implement changes that would hinder the malicious functionality of the resulting app.

Attacks vs Phishing Website Detectors

To counter phishing websites mimicking benign websites, state-of-the-art solutions rely on ML models for visual recognition. For instance, PhishIntention [129] extracts the logos from the screenshot of a given webpage, and then uses ML to determine if such logos resemble those of a well-known brand (e.g., PayPal); then, if a match is found, the systems checks if the domain of the analysed webpage matches the domain of the well-known brand (ideally, phishing webpages resemble well-known brands, but are hosted under different domains). In this case, the feature space is represented by the image of a logo (extracted from the screenshot of a webpage).

To introduce perturbations in such a feature space, the attacker simply needs to visually alter the logo in the phishing webpage—on which they have, by definition, complete control. For instance, this can be done by taking the logo image, change some pixels, and stitch the new "adversarial" logo onto the phishing webpage [130].

This is an exemplary case in which *the attacker may have complete control on the feature space*. However, there are other types of phishing website detectors (reliant, e.g., on the analysis of information extracted from the HTML) that do not allow an attacker to tamper with the feature space—unless the attacker has access to the internal workflow of the targeted phishing detection system. We point the interested reader to [131, 60].

Takeaway. Adversarial perturbations can be applied in many ways. Yet, depending on the specific domain of application, introducing some perturbations may be challenging for real-world attackers. It is hence crucial to specify the "problem space" in which the attacker is allowed to operate. In such a way, it is possible to identify what actions an attacker can take to reach their underlying goal—and, hence, simulate realistic attack strategies.

6.D Literature Review: Method

For our systematic literature review, we proceed by following three steps (inspired by [4, 55, 2]). We describe these steps by following the well-known PRISMA guidelines [100].

- *Paper Collection.* First, we consider: the 38 works analysed by [8]; the 46 papers reviewed in [2]; and the 88 papers in [55]. We do so because all these works scrutinised prior work related to ML-NIDS (for [2, 8]) or ML security (for [55]) published in top-tier venues within 2017–2023 Then, we consider the three works [19, 14, 63], and apply the snowball method [77], collecting all papers that are cited by either of these works [19, 14, 63]. We do so because these three works [19, 63, 14] are well-known in the ML-NIDS domain and consider "problem space" adversarial ML attacks that resemble real-world scenarios. After removing duplicates, we obtain a set of 292 papers.
- *Automated Filtering.* We review the text of these 292 papers and determine whether there is an evaluation of "adversarial ML attacks" against ML-NIDS. We do so by means of a keyword search with the terms "adversarial perturbation" or "adversarial attack" or "adversarial example". We filter the papers mentioning any of the keywords for at least three times (many papers simply mention similar terms for future work, e.g. [40]), increasing the likelihood that the paper is indeed about adversarial ML; this gives us 163 papers.
- *Manual Analysis.* We analyse the remaining 163 papers which carry out a "adversarial" assessment of ML-NIDS. We scrutinise "where" the perturbation is applied: this is done with a qualitative analysis focused on inspecting the experimental methodology followed in the paper, gauging whether the perturbations stem from attackers issuing different commands on their controlled hosts (i.e., host-space perturbations); we may also check the source code (if provided).

These operations were performed in Aug. 2024 by two researchers (having [5–8] years of experience in the ML-NIDS domain) who frequently interacted and, in case of doubt, discussed their findings to reach a consensus (as in [4, 55]).

6.E Additional Experimental Details and Results

We provide additional experimental results obtained to validate the statements made in this chapter. Appendix 6.E.1 evaluates the degree of determinism in the traffic generated by ConCap compared to a real network. Appendix 6.E.2 discusses additional host-space perturbations. Appendix 6.E.3 shows supplementary results that could not be put in the main chapter.

6.E.1 Determinism of ConCap's Traffic

The traffic generated by ConCap closely resembles that of a real network. However, we wonder: is the traffic generated by ConCap *deterministic*—or is ConCap's traffic also affected by some noise? Answering such a question allows one to measure the extent to which ConCap can truly approximate the "unpredictable" behavior of real-world networks.

Setup. To answer our question, we created four scenarios with increasing complexity in the traffic exchanged between the attacker and target: a simple ping scan (10 ICMP requests), a basic port scan ($\mathrm{nmap\ -sS}$), a full port scan ($\mathrm{nmap\ -A\ -T4}$), and an ssh brute-force attack ($\mathrm{patator\ -P{=}1\ -RL{=}0\ -T{=}10}$). We capture the packets and generate the NetFlows for each scenario. We repeat each scenario 100 times. The intention is to study the degree of similarity across all

Table 6.4: **Traffic generated by ConCap is deterministic, but the nondeterministic nature of networking results in small variations in packets and bytes.** The main reason for the variation is how data is acknowledged, using a separate TCP packet or as a header in the next packet with a payload.

Environment	Attack	Packets		Number of Flows	
		Count	Sum of Bytes	CICFlowMeter	Argus
Bare-Metal	Ping scan	20	1960	1	10
	Basic Port Scan	13	736	5	6
	Full Port Scan	2751 ± 88	$271\,551 \pm 6235$	1091 ± 43	1093 ± 43
	SSH Bruteforce	$30\,935 \pm 37$	$5\,060\,797 \pm 2417$	680	679
ConCap	Ping scan	20	1960	1	10
	Basic Port Scan	13	694	5	6
	Full Port Scan	2504 ± 6	$239\,401 \pm 4631$	1098 ± 1	1092 ± 1
	SSH Bruteforce	$26\,960 \pm 354$	$4\,759\,703 \pm 23\,353$	680	679

of these repetitions. We also carry out the exact same operations on the bare-metal servers to compare with a real-world setup (we repeat these experiments 10 times), which should not be deterministic.

Results. Table 6.4 reports the results (mean and std) across our trials for both setups, detailing the number of packets, byte count, and the number of NetFlows extracted by CICFlowMeter and Argus. We analyse these results by focusing on ConCap.

- We see no variation for the ping and basic port scan (nmap -sS) at both the packet- and NetFlow level. This shows that *the network connection is reliable*: these scenarios involve sending a single request and receiving a single response (or none). Variation would only occur if the network between the attacker and target were unreliable, causing packet duplication, loss, or corruption.
- In the complex scenarios (nmap -A -T4 and patator -P=1 -RL=0 -T=10), some variation is observed at the packet level, and in the case of the full port scan (nmap -A -T4), even at the NetFlow level. Similar to the realistic traffic assessment, packet-level variation arises from differences in how much data can be transmitted in a single packet and how this data is acknowledged. The flow-level variation in the full Nmap scan is caused by the aggressive mode (-T4) used by Nmap, which can overload the target.

To conclude, the network traffic generated by ConCap is not fully deterministic. We see this as an advantage: ConCap does not just simulate network traffic, it generates it in a way that resembles a real-world network—whose behavior is unpredictable [47] and, hence, not deterministic (as is evident by observing the results for the bare-metal setup in Table 6.4).

6.E.2 Additional Demonstrations of ConCap

The results in the chapter focus on the host-space perturbation for an ssh brute-force attack against an OpenSSH server. Specifically, we emphasized the impact of the “persistent” option of

Table 6.5: **Additional security assessments.** We report the tpr achieved by the baseline models trained on benchmarks (CIC17, CIC18) against the NetFlows generated via ConCap that conform to different host-space perturbations, networks, and OS.

Model	Hydra		Medusa		Network change		OS change	
	CIC17	CIC18	CIC17	CIC18	CIC17	CIC18	CIC17	CIC18
DT	0.019	0.000	0.000	0.000	0.743	0.729	0.750	0.750
RF	0.018	0.000	0.658	0.000	0.993	0.000	0.849	0.000
XGB	0.021	0.000	0.005	0.000	0.883	0.729	0.956	0.750
SVM	0.890	0.895	0.214	0.334	0.749	0.733	0.750	0.750
DNN	0.941	0.902	0.221	0.879	0.822	0.749	0.768	0.750

patator to adversarially evade an ML-NIDS with *a single character* perturbation. Naturally, there are many more potential actions a realistic attacker can take in the host space. To shed more light on these phenomena and further showcase the potential of ConCap, we tried using different brute-force attack tooling and adversarially tested them on the CICIDS17 and CICIDS18 benchmark datasets. The results of these experiments are presented in Table 6.5, together with additional security assessments ConCap allows, such as switching the attacker's controlled host environment. The subsequent paragraphs describe these security assessments.

Changing the bruteforcing-attack tool. patator is an all-round brute force tool written in Python. There exist many more brute forcing tools an attacker could use to successfully attack its target. We evaluated an attacker using *Hydra* [117] and *Medusa* [118]. *Medusa* has an interesting and unique parameter to set the client banner during the SSH Version Announcement. The default "SSH-2.0-MEDUSA_1.0", a minimal legitimate "SSH-2.0-a" with a single "a", and a long banner with the "a" repeated 100 times are evaluated. These are all examples of host space perturbations.

Changing the Network resources. The impact of the network environment is analyzed by configuring the network parameters in the ConCap scenario definition. Multiple variations for the *latency* (10, 25 or 100ms), *loss* (0 or 5%), *corrupt* (0 or 5%), and *duplicate* (0 or 5%) are evaluated. From the perspective of host-space perturbations, these cases represent "strong" threat models: to be practically viable, the attacker requires to have root access to the controlled host and manipulate the connectivity (as done, e.g., in [13]).

Changing the Base Version OS. The attacker in the CIC datasets performed the brute force ssh attack from an Ubuntu 18 machine. We evaluated if switching from one OS to another as an attacker can evade detection. We stress, however, that this can be an exemplary case of an *invalid host-space perturbation*: an attacker can unlikely "change" the OS of their controlled host—unless the attack is initiated from outside the network. Nevertheless, investigating this scenario is useful to assess the "stability" of the ML-NIDS against attacks launched by machines running different OSes.

Table 6.6: **Experiments on benchmark data with ConCap.** We show the results (fpr/tpr on benign/malicious NetFlows) of our models for each "train set" and "test set", for both the CICIDS17 and CICIDS18 benchmark datasets. Boldface denotes NetFlows generated via ConCap (which are always malicious). The red columns are the "adversarial" experiments on the baseline models, and green columns are the adversarial experiments on the adversarially trained models using ConCap; arrows (↑) denote significant changes w.r.t. the baseline models. We report the std in our repository [25].

	Train Set	Baseline Training Benign + P=1				Adversarial Training Benign + P=1 + **P=0 + P=1**				Open World Training Benign + **P=0 + P=1**	
	Test Set	Benign	P=1	**P=0**	**P=1**	Benign	P=1	**P=0**	**P=1**	Benign	P=1
CICIDS17	DT	<0.001	0.997	0.224	0.891	<0.001	0.996	1.000↑	1.000↑	<0.001	0.980
	RF	0.000	0.998	0.490	0.999	0.000	0.998	1.000↑	1.000↑	0.000	0.986
	XGB	<0.001	0.998	0.986	0.891	0.000	0.999	1.000↑	1.000↑	0.000	1.000
	SVM	<0.001	0.993	0.000	0.897	<0.001	0.991	1.000↑	1.000↑	<0.001	0.993
	DNN	0.000	0.998	0.000	0.914	0.000	0.998	1.000↑	1.000↑	<0.001	0.996
CICIDS18	DT	0.000	1.000	0.000	0.859	<0.001	1.000	1.000↑	1.000↑	<0.001	1.000
	RF	0.000	1.000	0.000	0.430	0.000	1.000	1.000↑	1.000↑	<0.001	1.000
	XGB	0.000	1.000	0.000	0.859	<0.001	1.000	1.000↑	1.000↑	<0.001	0.984
	SVM	0.000	1.000	0.000	0.907	<0.001	1.000	1.000↑	1.000↑	<0.001	1.000
	DNN	0.000	1.000	0.000	0.907	<0.001	1.000	1.000↑	1.000↑	<0.001	1.000

6.E.3 Additional Results

We presents detailed results for the interested reader.

First, we provide the results of our assessment on the benchmark datasets in Table 6.6. Then, we provide the training time of the models evaluated on the real-world network in Table 6.7. Finally, Fig. 6.6 shows an additional 30 NetFlow features compared for the patator brute force attack between traffic generated by ConCap and on a real network as described in §6.4.

We provide the tables with the standard deviation of our results (useful to carry out statistical tests) in our repository [25].

6.F Critical Remarks Addressed

6.F.1 The experiments only involve simple and well-known attacks, such as brute-force or port-scans.

This is true. We mostly (in §6.4 and §6.6) focused on simple, well-known attacks (e.g., brute force or port scans) that are widely understood, using them to **demonstrate the functionality** of both ConCap and host-space perturbations (HsP). However, they are by no means limited to these examples. In addition to the main experiments, **we conducted further evaluations involving recent, more sophisticated attacks**—specifically CVE-2024-47177, CVE-2024-36401, and CVE-2024-2961 (see §6.7.2). These attacks, which are not present in any existing benchmark datasets, highlight ConCap's capability to generate realistic network traffic and produce labeled data for complex, real-world exploits. We also applied host-space perturbations to these CVE-based attacks, showing that changes in host actions can significantly influence the resulting network

Table 6.7: **Training times for the models for the assessment of the real-world network.** Avg time (sec) and std. dev. over 5 training folds.

Train Set	Baseline		Adversarial		Open World
	Benign + P=1	Benign + P=0	Benign + P=1 + **P=0**	Benign + P=0 + **P=1**	Benign + **P=0 + P=1**
DT	0.3	0.2	0.4	0.5 ± 0.1	0.4
RF	3.1 ± 0.1	4.1 ± 0.2	5.4 ± 0.2	5.4 ± 0.2	5.4 ± 0.1
XGB	3.0 ± 0.4	3.9 ± 0.6	4.5 ± 0.3	3.9 ± 0.2	5.1 ± 0.6
SVM	0.5	0.9 ± 0.1	1.2 ± 0.2	1.1	1.1 ± 0.1
DNN	9.2 ± 0.5	12.3 ± 0.5	13.9 ± 0.8	11.7 ± 0.2	11.8 ± 0.8

(a) Real-world

	Baseline	Adversarial	Open World
DT	6.7 ± 1.3	8.2 ± 1.2	5.3 ± 0.6
RF	40.8 ± 4.2	36.9 ± 1.4	38.0 ± 3.6
XGB	15.1 ± 3.8	21.9 ± 3.4	27.1 ± 6.6
SVM	4.1 ± 0.9	12.5 ± 1.4	25.3 ± 6.5
DNN	17.2 ± 2.0	22.6 ± 0.5	20.0 ± 1.5

(b) CICIDS17

	Baseline	Adversarial	Open World
DT	40.4 ± 1.8	38.6 ± 3.3	35.1 ± 3.2
RF	547.8 ± 6.3	618.5 ± 55.9	504.3 ± 41.9
XGB	118.8 ± 7.4	120.8 ± 7.6	47.8 ± 30.8
SVM	101.3 ± 10.9	131.1 ± 18.3	389.6 ± 58.2
DNN	574.5 ± 45.8	384.3 ± 28.4	389.3 ± 46.6

(c) CICIDS18

traffic, all without directly manipulating packet contents. For more advanced threats, including multi-stage attacks like lateral movement, researchers can define each phase as a separate scenario, allowing for precise control and modular experimentation. These results (in Appendix 6.E) demonstrate that both ConCap and HsP go far beyond the simple examples presented in the main text, offering powerful tools for studying modern and complex network threats.

6.F.2 One can easily manipulate and send "perturbed" packets via scapy and tcpreplay, therefore host-space perturbations are not the only way that attackers can use to evade ML-NIDS.

This is true. Attackers can manipulate packet structures directly, but that approach often requires higher threat models and risks breaking protocols or leaving obvious artifacts (e.g., unnatural packet headers/payloads). Besides, even if the manipulation is functionally correct, there is **no guarantee that the packet(s) sent by the attacker-controlled host will be received by the router** exactly as they are sent, and exactly in the same "order" as that of an original PCAP trace (containing no adversarially-manipulated packets). In other words, our argument is that manipulating the packets included in a PCAP trace (assumed to be collected by the router/switch that is connected to the ML-NIDS) cannot perfectly simulate the actions that an attacker would perform on their controlled hosts. Nonetheless, instructing an host to reproduce (via tcpreplay) an adversarially-manipulated PCAP trace is an operation that falls into the definition of an host-space perturbation—and which can be reproduced via ConCap. Finally, and more generally, we do not claim that gradient-based perturbations are impossible—just that they may not be the first choice of an attacker when attempting evasion of an ML-NIDS.

6.F.3 The technical contribution of ConCap is only a trivial application of Kubernetes.

While ConCap leverages Kubernetes for container orchestration, its technical contribution extends far beyond simply deploying containers. Configuring and executing reproducible, labeled network experiments end-to-end—especially in the context of security research—is not supported by Kubernetes out of the box. ConCap builds a complete, automated pipeline that includes scenario definition, multi-node container deployment, traffic shaping, packet capture, NetFlow extraction, and labeling. For example, Kubernetes relies on a container networking interface (CNI) plugins for networking, but support for traffic shaping (e.g., controlling bandwidth or latency) is limited and experimental across CNIs. Rather than tying the system to a specific CNI implementation, ConCap uses Linux's Traffic Control to provide fine-grained control over container-level networking, ensuring consistent behavior across clusters. Additionally, Kubernetes does not natively support packet capture, NetFlow generation, or any form of labeling—essential components for generating high-quality datasets in NIDS research. While some of these steps may seem trivial in isolation, executing them reliably and reproducibly across scenarios is a known challenge. ConCap addresses the data-inconsistency issues found in prior work (e.g., flawed configurations, mislabeling, incomplete captures [8]) by automating the entire process and enforcing consistency. It allows researchers to specify attacker-target interactions using containers, resource constraints, trigger timelines, and networking conditions in a structured, reusable format. As a result, experiments can be shared and rerun with minimal setup, supporting reproducibility and fair comparison. A short demo of ConCap is available in our repository [25].

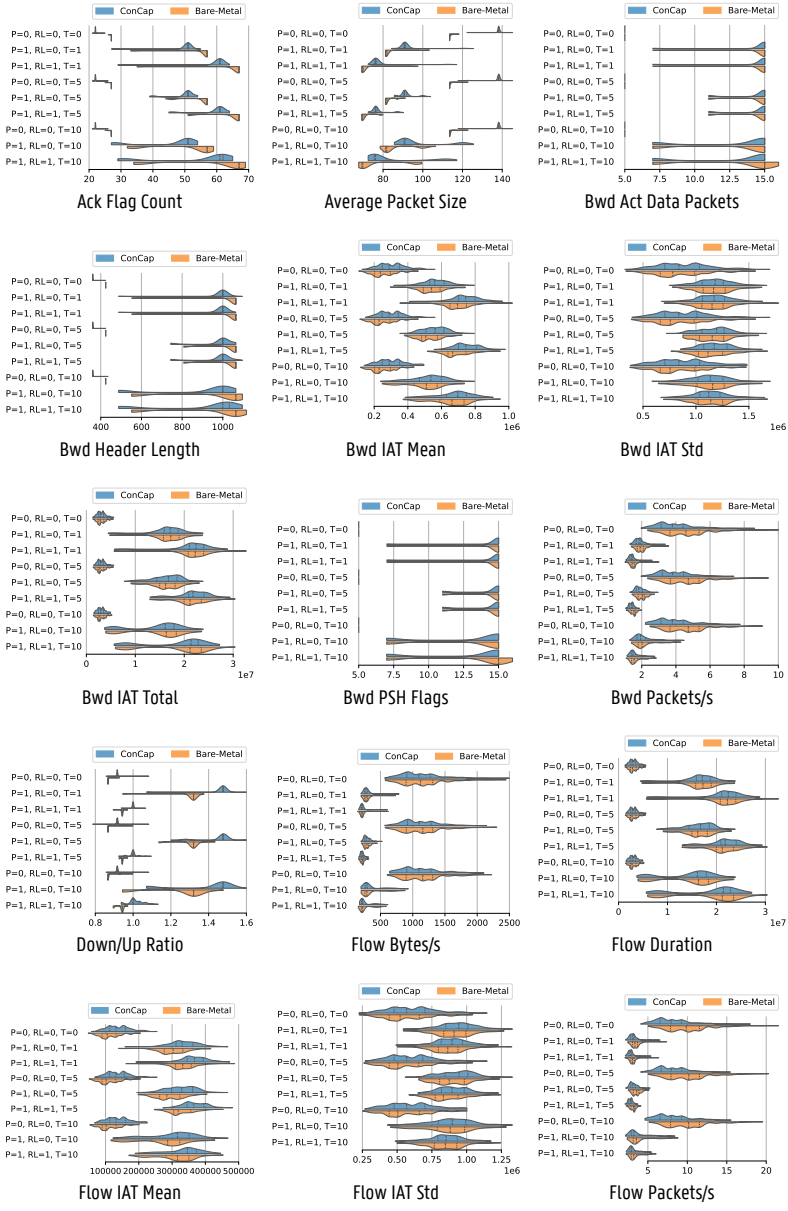


Figure 6.6: Comparison of NetFlow feature distributions of a Patator brute-force SSH attack between ConCap and a real network.

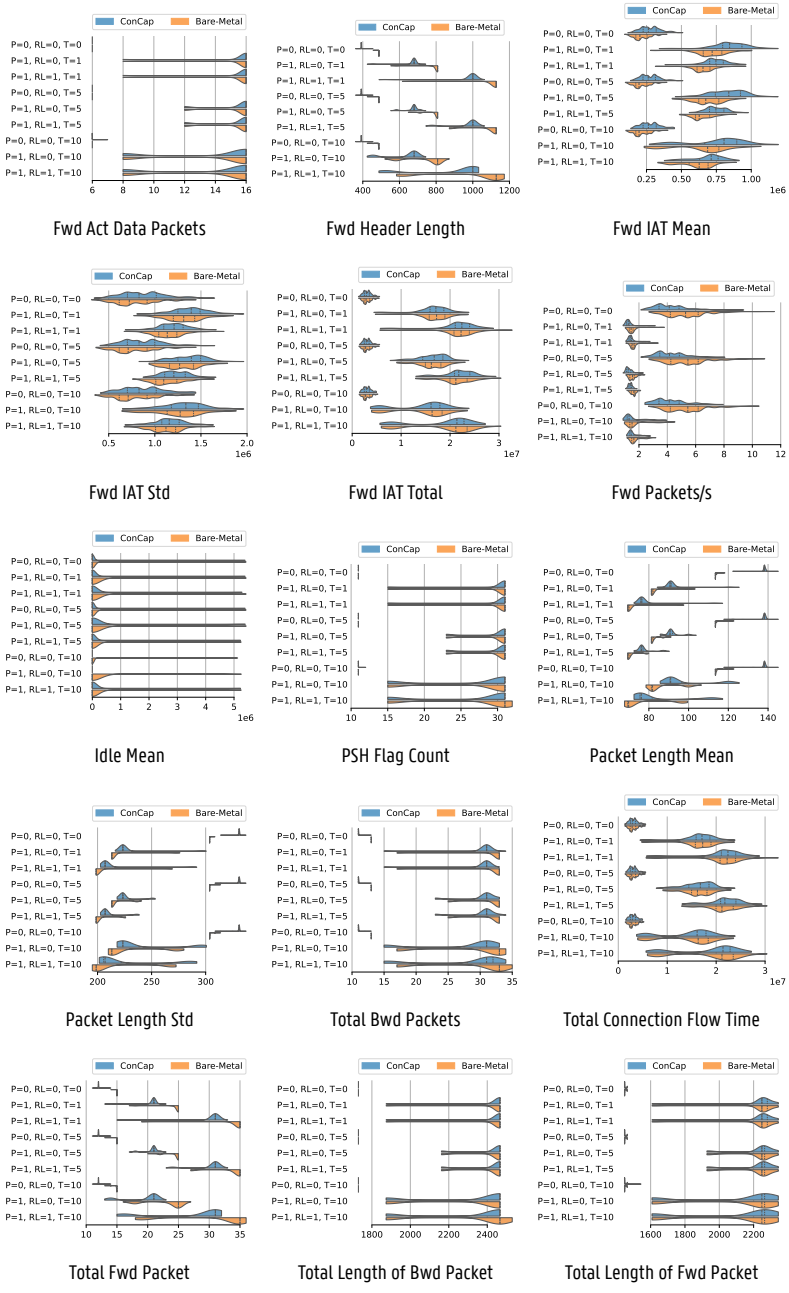


Figure 6.6: Comparision of NetFlow feature distributions of a Patator brute-force SSH attack between ConCap and a real network. (cont.)

7

Conclusions and Future Research Directions

This concluding chapter reflects on the proposed solutions in the broader context of this dissertation and identifies open challenges and future research directions regarding generalizable, secure, and practical ML-NIDS.

7.1 Summary of Contributions

This dissertation made several contributions over the previous chapters across the three thematic pillars, introduced in Chapter 1, towards more generalizable, practical, and secure ML-NIDS. Each contribution addresses one or more of the research questions formalized in the first chapter.

7.1.1 AI for Cybersecurity

This pillar targets the open challenges of the application of AI to Cybersecurity, in particular, the machine learning challenges in the development of intrusion detection systems.

At the beginning of this PhD, the research field was characterized by the release of a new wave of "modern" benchmark datasets, such as CIC-IDS-2017 and CSE-CIC-IDS-2018 [1]. Initial studies using these datasets reported near-perfect classification performance using predominantly supervised ML approaches, often without addressing known limitations or considering realistic operating environments [2]. As a result, it seemed that ML-NIDS was a solved problem. Early in this dissertation, we began to question these claims [3]. Chapter 2 benchmarks unsupervised and self-supervised

learning techniques on these modern datasets, confirming that these approaches can achieve similarly high classification performance as supervised models (RQ1). However, instead of stopping there, we introduced an inter-dataset evaluation strategy to assess the generalization capabilities of these models across heterogeneous but compatible datasets. Specifically, we trained models on CIC-IDS-2017 and tested them on CSE-CIC-IDS-2018, and vice versa (RQ2). The results revealed a significant performance drop of up to 37% AUROC in inter-dataset evaluations, confirming that generalization remains a fundamental challenge—regardless of whether models are trained in a supervised or unsupervised manner. As a result, we advocated for the adoption of more realistic evaluation strategies by researchers proposing new approaches for ML-NIDS.

RQ1. *Can unsupervised ML-NIDS models achieve near-perfect classification performance on par with supervised methods when trained and evaluated on academic benchmark datasets?*
Yes, unsupervised and self-supervised models can achieve near-perfect classification performance when evaluated on the same datasets they are trained on. However, this performance is misleading, since these models fail to generalize, showing drops up to 37% in AUROC using an inter-dataset evaluation strategy.

Motivated by the generalization limitations discovered in state-of-the-art single-model approaches, we collaborated with the high-speed networking research group at NYCU, Taiwan, to develop and evaluate a multi-stage ML-NIDS architecture designed for real-world deployment. This work, in Chapter 3 addresses both the generalization challenges from a model-centric perspective (RQ2) and the practical requirements for scalable, real-world ML-NIDS deployments (RQ3). The proposed architecture consists of three sequential stages: an anomaly detector, a multi-class classifier, and a zero-day detector—leveraging a combination of supervised and unsupervised ML techniques. Experimental results demonstrate that the multi-stage approach not only improves overall classification performance compared to single-model baselines but also robustly detects zero-day attacks. Additionally, the design supports hierarchical, bandwidth-efficient deployments with improved privacy preservation and enables threshold-based tuning to balance performance trade-offs without requiring model retraining.

After numerous unsuccessful attempts to address the generalization challenge using existing benchmark datasets [4, 5, 6, 7], we reconsidered the problem as one rooted in data quality rather than model design. This shift in perspective led to the development of ConCap, a framework for fine-grained, controlled network traffic generation with automated labeling. ConCap allows researchers to define attacker-target interactions in containerized environments, configure network characteristics and resource constraints, and reproduce experiments with minimal effort. As demonstrated in Chapter 6, ML-NIDS trained on traffic generated by ConCap demonstrate strong generalization performance across both real-world networks and academic testbeds—provided the configured network characteristics reflect the ones of the target environment. These promising results open up new research opportunities in dataset generation, reproducibility, and evaluation strategies, as more extensively discussed in Section 7.2. With this, the dissertation brings the generalization story full circle—starting from doubts about overly optimistic results on flawed datasets and concluding with a practical framework to address the root cause.

RQ2. *Do ML-based NIDS generalize across heterogeneous network environments?*
No, current ML-NIDS models trained on academic benchmark datasets show poor generalization to different network environments, even when the datasets share similar traffic generation methodologies. However, we demonstrate that it is possible to train models that do generalize using data generated with ConCap, provided the training traffic reflects the target network's characteristics.

7.1.2 Research to Practice

This pillar focuses on the adoption by practitioners of proposed ML-NIDS approaches by researchers into practical, scalable, and reproducible systems that are ready for real-world deployment. While the first pillar highlighted generalization as a key limitation in academic ML-NIDS, this second pillar tackles the often-overlooked practical challenges that prevent promising research prototypes from being adopted in practice.

To address these challenges, we first developed *ChronosGuard*, a hierarchical, containerized implementation of the multi-stage ML-NIDS introduced in Chapter 3. Designed for modern cloud-native environments, ChronosGuard enables flexible and scalable deployments using container orchestration platforms such as Kubernetes. Chapter 4 presents the results of thousands of deployment experiments, systematically exploring how factors such as deployment strategy, network topology, and orchestration algorithms affect performance and resource efficiency. These experiments provide empirical insights into practical deployment trade-offs and demonstrate that hierarchical deployments and network-aware schedulers can significantly improve the operational performance of ML-NIDS systems (RQ3).

RQ3. *How can we design and deploy a distributed ML-NIDS suitable for large-scale, real-world deployment in modern cloud infrastructures?*
We proposed ChronosGuard, a scalable, multi-stage ML-NIDS architecture designed for deployment in containerized cloud environments. Empirical evaluation shows that hierarchical deployments, combined with network-aware scheduling, can significantly improve detection performance, reduce bandwidth, and optimize resource usage—making large-scale deployment feasible and effective.

However, deploying ML-NIDS at scale also requires efficient and robust preprocessing pipelines—particularly for extracting statistical flow features from high-speed network traffic. Chapter 5 addresses this challenge by conducting a comprehensive literature review and empirical benchmark study of network flow exporters most commonly used in ML-NIDS research. The review reveals a fragmented ecosystem characterized by a reliance on custom tools that often lack standardization, performance, and reproducibility. These observations are further substantiated through a systematic benchmark of both academic and industry-standard flow exporters in real-time and offline processing scenarios. The results highlight substantial limitations in terms of throughput, resource efficiency, and feature compatibility—reinforcing the need for standardized, high-performance tooling to support practical ML-NIDS deployments (RQ4).

To bridge this gap, we introduced RustiFlow, a high-performance, open-source network flow exporter built in Rust and powered by eBPF. RustiFlow supports both offline and real-time modes, multiple standard feature sets, and extensibility for custom features, making it a versatile tool for both researchers and practitioners. Its design addresses the core limitations identified in the benchmarking study and enables reproducible, high-throughput ML-NIDS pipelines.

RQ4. *What are some challenges, besides the lack of generalization, that may impair the integration of ML-NIDS research prototypes into practice?*
Yes. We tackled two key challenges: (1) Academic prototypes often lack support for real-world scalability, cloud-native infrastructure, or efficient resource usage. (2) Statistical network flow feature extraction remains a bottleneck. Our benchmark showed that most tools used in ML-NIDS research are custom-built and not designed for high performance or reproducibility. We addressed this by developing RustiFlow, a high-throughput, extensible, and standardized flow exporter suitable for both academic and operational use.

7.1.3 Cybersecurity for AI

This pillar explores the adversarial nature of the cybersecurity domain and investigates how ML-NIDS can be targeted and potentially evaded. While prior work in adversarial machine learning has primarily focused on domains such as computer vision [8]—where attackers can easily manipulate input features—ML-NIDS presents a more constrained and realistic threat model. In practice, attackers have no access to the internals of the ML system or the flow feature extraction process and thus cannot directly manipulate the input feature vectors [9, 10].

Chapter 6 addresses this gap by introducing and formalizing a new class of adversarial attacks known as host-space perturbations. Unlike traditional gradient-based approaches in the feature-space, host-space perturbations modify attacker behavior at the host level—such as changing a single character in a command-line instruction—thereby altering the generated network traffic. These modifications are trivial for real-world attackers to perform but have remained largely unexplored in the ML-NIDS literature due to the lack of practical tools for studying them. We developed and released ConCap, a containerized network traffic generation framework capable of producing labeled, realistic attack traffic in controlled environments to enable such experimentation. ConCap facilitates reproducible experimentation and allows researchers to simulate and evaluate the impact of host-space perturbations on ML-NIDS performance. Empirical results—both on academic datasets and in real-world networks—demonstrate that state-of-the-art ML-NIDS can be practically evaded by an attacker using minimal host-space perturbations (RQ5).

RQ5. *How feasible is it for a realistic attacker to evade an ML-NIDS?*
Highly feasible. We introduced a new class of host-space perturbations, where minimal changes to an attacker's behavior—such as changing a single command parameter—can evade detection by state-of-the-art ML-NIDS. While trivial for attackers to perform, studying the effect of such perturbations has traditionally been difficult for researchers due to tooling limitations. We addressed this gap by enabling reproducible experimentation with ConCap.

7.2 Future Research Directions

The ML-NIDS domain remains an active area of research. Despite the contributions made in this dissertation toward building more secure, practical, and generalizable ML-NIDS solutions, several challenges remain open. Below, we outline several promising directions for future research. In particular, the final chapter of this dissertation, Chapter 6, which introduces ConCap—a tool for fine-grained network traffic generation with automated labeling–has shown promising results in enabling generalization across heterogeneous networks. This advancement opens up new research opportunities that were previously difficult to explore.

7.2.1 Improving the Quality of NIDS Datasets

As discussed throughout this dissertation—particularly in Chapter 6—there remains a significant lack of publicly available, high-quality, and well-labeled NIDS datasets. Numerous studies [11, 12, 13, 14] have extensively analyzed the limitations of current academic benchmark datasets and identified multiple flaws. These datasets often lack compatibility, a standardized evaluation protocol for consistent comparison across studies, reproducibility, and detailed documentation of the network traffic generation process.

Flood et al.[14] identified seven "bad design smells" that weaken the utility of benchmark datasets and downstream NIDS research. The development of ConCap, presented in Chapter 6, presents promising new opportunities for generating high-quality network traffic datasets that address these shortcomings.

- First, ConCap's end-to-end pipeline—including traffic generation, feature extraction, and automated labeling—substantially reduces the manual effort from researchers to set up network traffic generation experiments. This reduction in setup overhead allows researchers to dedicate more time to include a wider range of attack scenarios. In addition, the use of reproducible scenario files—serving as blueprints for the traffic generation experiments—enables the straightforward creation of numerous *host-space perturbations*. These properties collectively not only increase the diversity of the resulting datasets but also support their iterative extension over time, directly addressing the first design smell: "poor data diversity".
- The second design smell, "highly-dependent features" that empower shortcut learning on features unrelated to the underlying attack mechanisms, can be mitigated through the increased dataset diversity. By systematically varying elements such as network characteristics, operating system versions, attack tools and parameters, and target configurations, we can obtain a realistic distribution for these spurious features and reduce the risk of models overfitting to artifacts introduced by the network traffic generation process.
- The third and fourth design smells, "unclear ground truth" and "wrong labels", are addressed by ConCap's controlled and minimalistic containerized environments. By isolating attacker-target interactions and removing unrelated background services, ConCap ensures precise and reliable labeling of the generated network traffic.

- Finally, the last two design smells—"traffic collapse" and "artificial diversity"—typically result from misconfigurations or hard-to-control parameters, such as hardware failures that impact the generated network traffic. While such issues can never be entirely eliminated, ConCap's standardized and reproducible pipeline simplifies their detection, inspection, and correction. Unlike current approaches that restrict the quality analysis to the generated output, ConCap also enables the inspection of the entire traffic generation process itself, providing more reliable quality control.

In conclusion, ConCap provides a promising foundation for creating diverse, high-quality, and well-labeled network traffic datasets. As a first step, it could be used to reproduce existing state-of-the-art datasets. Addressing the data quality issues identified by Flood et al. [14] not only advances NIDS development but also supports research in related areas such as traffic classification, fingerprinting, and other applications that depend on labeled network traffic.

7.2.2 Adversarial Attacks against ML-NIDS

Augmentation of Existing Datasets for Security Assessments. Beyond generating entirely new datasets, ConCap also enables the augmentation of existing benchmark datasets or real-world traffic captures with realistic network traffic that reflects how it would have been observed in the original target network. Previously, this was only possible with physical access to the environment in which the original traffic was captured. In practice, such access is often restricted—or entirely prohibited—due to security and privacy constraints that prevent conducting attacks in real-world target environments. However, with ConCap, researchers can now mimic these environments and generate network traffic that generalizes to the characteristics of the target network. This capability opens the door to new types of experiments–such as the security assessments presented in Chapter 6—which were previously impossible to conduct.

Addressing the Inverse Feature-Mapping Problem. A fundamental challenge in adversarial ML for NIDS is the inverse feature-mapping problem[15]. While extracting statistical flow features from network traffic is straightforward, the reverse process—reconstructing packet-level traffic from a given feature vector—is not possible, as the feature-mapping function is neither invertible nor differentiable. Additionally, multiple packet traces can produce identical flow features, further highlighting the inherent ambiguity of the problem. Pierazzi et al. [16] proposed a novel formalization for adversarial ML evasion attacks in the problem-space, differentiating between perturbations in the feature-space and problem-space, while arguing that the latter are more realistic. In Chapter 6, we extended this formalization in the NIDS context by further dividing the problem-space into traffic-space and host-space perturbations, providing a more fine-grained understanding of feasible adversarial transformations under a realistic threat model. Since there exists no inverse mapping from flow features to network packets, it is questioned if feature-space attacks are realizable in practice [17]. ConCap now offers a brute-force workaround to this challenge by enabling the systematic generation of a wide range of host-space perturbations. If a host-space perturbation is found that results in the desired adversarial feature vector, it demon-

strates the feasibility of the corresponding feature-space attack. On the contrary, if no such perturbation is found, it suggests—but does not definitively prove—that the attack is not practically realizable. Given the vastness of the search space, exhaustive exploration is infeasible. To make this brute-force process more efficient, optimization techniques such as Bayesian optimization or reinforcement learning could be used to guide the search.

Evaluating the Impact of Feature Extractors. Another underexplored area is understanding how different feature extractors influence ML-NIDS robustness. The recent SoK [18] notes that most studies rely on a single feature extractor during both training and evaluation. In rare cases where multiple extractors are used, they are usually tied to different datasets—meaning the same raw traffic is never analyzed with multiple extractors. This raises important research questions: Are some flow exporters or feature sets inherently more robust to adversarial manipulation? Do certain feature configurations better capture attack behavior or improve generalization? Researchers can evaluate multiple feature extractors on the same underlying raw network traffic, enabling systematic comparisons. This opens the door to benchmarking feature sets and exporters not just on classification performance but also on resilience to adversarial attacks.

7.2.3 Improving Generalization of ML-NIDS

The steps discussed in Section 7.2.1 for generating new NIDS datasets can also be applied to augment existing datasets during training, validation, or testing phases. Expanding the test set with additional diverse network traffic enables a more realistic evaluation of generalization strength compared to the traditional intra-dataset evaluations, as discussed in Chapter 3. Similarly, augmenting the training and validation set with more diverse data can significantly improve the generalization strength of ML-NIDS models. Chapter 6 demonstrated that models trained on a single variant of an attack failed to detect the same attack implemented using different attack tools or configurations reliably. Even small changes in network characteristics between the training and test dataset can be sufficient to prevent the model from reliably detecting the attack. However, when the second variant was included in the training set, the model achieved perfect detection performance for both variants. This highlights the role of dataset diversity.

Augmenting datasets with every possible host-space perturbation of an attack is practically infeasible due to the vastness of the data space. However, increasing the diversity of current academic benchmark datasets by even a few orders of magnitude may already be sufficient to train ML-NIDS models that can generalize across different attack variants. As discussed in Chapter 6, it is important to note that not all host-space perturbations lead to changes in network traffic. To make the data collection process more efficient, it would be valuable to prioritize those perturbations that are most likely to affect the resulting network traffic. Automated approaches that can identify such perturbations—based on, for example, source code analysis or documentation—could help guide data generation without the need to execute all possible perturbations.

Chapter 6 emphasizes the importance of mimicking the characteristics of the target network when

generating network traffic. However, in real-world scenarios, the origin of an attack is often unpredictable—it might come from within the internal network, from a previously compromised host, or from an external network with unknown properties. To develop ML-NIDS that are invariant to network conditions, one approach is to augment the training dataset using a variety of potential network configurations. Yet, due to the immense size of the configuration space and the need to repeat this for every attack variant, it is impractical to cover it fully using data alone. More intelligent strategies may be needed to achieve generalization across network environments. Instead of relying purely on data augmentation, research could focus on learning higher-level, network-invariant representations of traffic. Such representations would allow models to maintain robustness without needing to sample the full range of possible network conditions. I believe that combining both approaches is critical for success.

7.2.4 Adopting State-of-the-Art Models

Another promising direction for future research lies in adopting more advanced and specialized models for ML-NIDS. Rather than relying on a single general-purpose model, the use of multiple specialized models—each tailored to detect specific types of attacks or operating in different phases of a detection pipeline—can significantly improve detection performance. As demonstrated in Chapter 2, chaining multiple models improved the classification performance on current benchmark datasets compared to a single model.

Specialized models also present practical benefits: they are often easier to train using limited amounts of high-quality labeled data and can leverage domain-specific representations to enhance learning. For instance, using multi-flow representations or time-windowed input structures can help models better capture sequential and contextual patterns within the network traffic [19]. Similarly, models built on graph-based representations [20, 21] are better suited for detecting complex attack behaviors that span multiple hosts and time periods, such as lateral movement after the initial infection of a host.

Furthermore, as more high-quality, diverse NIDS datasets become available—through tools like ConCap—researchers will be better positioned to explore models with increased complexity and capacity. Currently, performance comparisons on academic benchmark datasets often show that simple traditional models (e.g., random forests) trained on only 5–20% of the available features and a small fraction of the data (e.g. 5%) can achieve similar performance to more complex deep learning models using all available data [22]. This suggests that existing datasets are too limited to exploit the learning capacity of state-of-the-art models fully. These models, when paired with sufficient high-quality data and representative input structures, may yield substantial improvements in generalization and robustness.

Bibliography

[1] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006639801080116

[2] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Classification hardness for supervised learners on 20 years of intrusion detection data," *IEEE Access*, vol. 7, pp. 167 455–167 469, 2019.

[3] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Inter-dataset generalization strength of supervised machine learning methods for intrusion detection," *Journal of Information Security and Applications*, vol. 54, p. 102564, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214212619310415

[4] L. D'hooge, M. Verkerken, B. Volckaert, T. Wauters, and F. De Turck, "Establishing the Contaminating Effect of Metadata Feature Inclusion in Machine-Learned Network Intrusion Detection Models," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, L. Cavallaro, D. Gruss, G. Pellegrino, and G. Giacinto, Eds. Cham: Springer International Publishing, 2022, pp. 23–41.

[5] L. D'hooge, M. Verkerken, T. Wauters, B. Volckaert, and F. De Turck, "Discovering non-metadata contaminant features in intrusion detection datasets," in *2022 19th Annual International Conference on Privacy, Security & Trust (PST)*. IEEE, 2022, pp. 1–11.

[6] L. D'hooge, M. Verkerken, T. Wauters, F. De Turck, and B. Volckaert, "Investigating generalized performance of data-constrained supervised machine learning models on novel, related samples in intrusion detection," *Sensors*, vol. 23, no. 4, p. 1846, 2023.

[7] L. D'hooge, M. Verkerken, T. Wauters, F. De Turck, and B. Volckaert, "Castles built on sand: Observations from classifying academic cybersecurity datasets with minimalist methods," in *8th International Conference on Internet of Things, Big Data and Security (IoTBDS)*. SciTePress, 2023, pp. 61–72.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[9] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *IEEE Symposium on security and privacy (SP)*, 2020.

[10] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, and K. A. Roundy, ""Real Attackers Don't Compute Gradients": Bridging the Gap Between Adversarial ML Research and Practice," Dec. 2022, arXiv:2212.14315 [cs]. [Online]. Available: http://arxiv.org/abs/2212.14315

[11] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study," in *2021 IEEE Security and Privacy Workshops (SPW)*, May 2021, pp. 7–12.

[12] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, "Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018," in *IEEE Conference on Communications and Network Security*. IEEE, 2022.

[13] M. Catillo, A. Pecchia, and U. Villano, "Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS?" in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume*, 2023.

[14] R. Flood, G. Engelen, D. Aspinall, and L. Desmet, "Bad design smells in benchmark nids datasets," in *IEEE 9th European Symposium on Security and Privacy (EuroSP)*, 2024.

[15] R. Sheatsley, N. Papernot, M. Weisman, G. Verma, and P. McDaniel, "Adversarial examples in constrained domains," *arXiv preprint arXiv:2011.01183*, 2020.

[16] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing Properties of Adversarial ML Attacks in the Problem Space," in *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020, pp. 1332–1349, iSSN: 2375-1207. [Online]. Available: https://ieeexplore.ieee.org/document/9152781/?arnumber=9152781

[17] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, "Modeling Realistic Adversarial Attacks against Network Intrusion Detection Systems," *Digital Threats: Research and Practice*, vol. 3, no. 3, pp. 1–19, Sep. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3469659

[18] G. Apruzzese, P. Laskov, and J. Schneider, "Sok: Pragmatic assessment of machine learning for network intrusion detection," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2023.

[19] V. Rimmer, A. Nadeem, S. Verwer, D. Preuveneers, and W. Joosen, "Open-World Network Intrusion Detection," in *Security and Artificial Intelligence: A Crossdisciplinary Approach*, L. Batina, T. Bäck, I. Buhan, and S. Picek, Eds. Cham: Springer International Publishing, 2022, pp. 254–283. [Online]. Available: https://doi.org/10.1007/978-3-030-98795-4_11

[20] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos:: Practical intrusion detection and investigation using whole-system provenance," in *IEEE Symposium on Security and Privacy*, 2024.

[21] I. J. King, X. Shu, J. Jang, K. Eykholt, T. Lee, and H. H. Huang, "Edgetorrent: Real-time temporal graph representations for intrusion detection," in *International Symposium on Research in Attacks, Intrusions and Defenses*, 2023.

[22] L. D'hooge, M. Verkerken, T. Wauters, F. De Turck, and B. Volckaert, "Characterizing the impact of data-damaged models on generalization strength in intrusion detection," *Journal of Cybersecurity and Privacy*, vol. 3, no. 2, pp. 118–144, 2023.