THE UNIVERSITY *of* EDINBURGH

Enhancing Trust in NIDS: From Flawed Benchmarks to Formal Guarantees

Robert Flood

Doctor of Philosophy

THE UNIVERSITY OF EDINBURGH

2025

Abstract

Networks serve as the critical backbone for our increasingly digital world, supporting everything from personal communications to essential infrastructure. Securing these networks against malicious activities is paramount for maintaining the confidentiality, integrity, and availability of data. Machine learning (ML) has been extensively applied to network intrusion detection, with researchers particularly interested in ML models' ability to detect generalise to new attack patterns. However, research into Network Intrusion Detection Systems (NIDS) can be marred by their opaque nature. These models often lack transparency on two fronts: they are trained on large datasets with flaws that compromise benchmarking validity, and they function as black boxes, making critical security decisions via mechanisms that resist straightforward interpretation. These issues fundamentally undermine trust in their capabilities. This thesis addresses this trust gap via two complementary strands of research: the systematic interrogation of benchmark NIDS datasets and the application of neural network verification techniques to NIDS.

We begin by introducing the concept of 'Bad Data Design Smells' as indicators of flaws in the design of synthetic datasets that undermine their suitability as evaluation benchmarks. Through a systematic literature overview and detailed case studies, we demonstrate how these flaws significantly impact downstream research, such as dataset artefacts degrading classification accuracy by over 90%. We develop a two-pronged approach to identify these smells, combining systematised manual analysis with automated heuristic measurements. We then extend this analysis by contextualising the complexity of network data. We introduce a novel metric based on spectral clustering that allows us to compare NIDS benchmarks with datasets from other fields. Despite their ubiquity, our measurements consistently reveal that benchmark NIDS datasets exhibit minimal complexity compared to even simple benchmarks in other domains, limiting their utility in research.

Building upon these insights, we use neural network verification techniques to improve our ability to reason about ML-based NIDS training and inference. First, we encode domain knowledge as formal specifications that models must adhere to, enforcing these properties through adversarial training. We then formally verify model adherence, which substantially improves cross-dataset generalisation accuracy by over 35%. In addition, we use these specifications as a form of explainability, gaining insight into model decision boundaries and failure modes, such as ranking model fragility to feature perturbations. Our approach outperforms model certification techniques, which we show fails in likely settings. Second, we leverage the counter-examples produced by verification frameworks to generate constrained, realisable adversarial examples for NIDS, addressing a notable gap in the field between feature-space and problem-space adversarial attacks and improving on standard adversarial attacks in some cases.

By adopting a strategy of both scrutinising dataset quality and formally defining model behaviour, this thesis improves the trustworthiness of NIDS models. We address trust at both training and inference, by establishing clear links between dataset flaws and model outcomes, as well as formally specifying model behaviour to resist evasive attacks and concept drift. Altogether, these contributions introduce greater rigour into NIDS evaluation processes, advancing both theoretical and practical aspects of ML applied to network security.

Lay Summary

While most internet connections are harmless, a subset are malicious *intrusions*: unauthorised connections aimed at causing harm or havoc. To keep computer networks secure, we ask: '*Can we analyse network traffic to determine whether a specific connection is normal or an intrusion?*' This is the task of software known as *Intrusion Detection Systems* (IDS). However, standard IDSs are notoriously unreliable and laborious. Their manually written rules often fail to generalise, requiring frequent human checks and allowing intrusions to slip through with minor changes.

Thus, we ask: '*Can we build better IDSs that automatically infer rules that generalise?*' To answer this, researchers turn to *machine learning* (ML) and *neural networks*, areas of artificial intelligence focused on creating algorithms that generalise to unseen data — a seemingly natural fit.

In practice, ML-based IDSs present new research challenges. While ML has thrived in areas such as image recognition, this success relies on large datasets. However, internet packets are often contain sensitive information and collecting a large amount of network data would almost certainly violate privacy laws. Additionally, labelling this data is complex due to its intricate nature. To circumvent these issues, researchers rely heavily on synthetic network data, created using some artificial process to look like real-world data, to test proposed ML-based IDSs. In recent years, several synthetic IDS datasets have been released and used widely, forming a foundational aspect of the research field.

Yet, creating synthetic datasets that truly reflect real-world threats is time-consuming, arduous, and often neglected. This forms a major focus of this thesis: ***how do we rigourously analyse and evaluate the quality of synthetic network data?*** We present new methods which find that these datasets are often oversimplified, potentially affecting the validity of previous research and leading to experimental bias.

This thesis also asks ***how do we maximise the benefits of using synthetic network data?*** Unlike image or text generation, where progress involves larger ML systems, we argue for a different approach in intrusion detection. Instead of complex models, we enhance model performance by creating targeted, bespoke synthetic data. In later chapters, we use this data in conjunction with *neural network verification*, a set of techniques for ensuring that ML algorithms learn appropriate behaviours via strong mathematical guarantees.

Acknowledgements

First and foremost, I would like to express my profound gratitude to my PhD supervisor, David Aspinall, whose unwavering guidance has been instrumental throughout this journey. His patience in meticulously reviewing countless paper drafts, his commitment to our (always interesting!) meetings, and his understanding of my less-than-ideal email habits have been invaluable to my academic development. I would not have pursued a PhD if it wasn't for his excellent supervision during my MSc.

I am deeply grateful to Arm Ltd., and my industry supervisor Hugo Vincent, for their generous support through their PhD scholarship program. Equally, I extend my sincere appreciation to the School of Informatics and the Laboratory for Foundations of Computer Science for providing an intellectually stimulating environment with exceptional resources.

This research has benefited immensely from the insights and feedback of many brilliant people. I would like to thank — though this list is far from complete — Gints Engelen, Vera Rimmer, Gudmund Grov, Marco Casadio, Ekaterina Komendantskaya, Luca Arnaboldi, Michio Honda, Stuart Wray, Henry Clausen, Marc Juarez, and Tariq Elahi. Special recognition goes to the reviewers at EuroS&P, whose detailed and constructive feedback transformed my borderline-rejected paper into work worthy of the Distinguished Paper award.

The PhD journey is as challenging as it is rewarding, and my tendency to work against tight deadlines certainly added to that intensity and stress. Throughout these years, I have been blessed with a support system that kept me grounded and determined. My deepest appreciation goes to my partner, Jianyue Wang, whose unwavering encouragement and understanding has been vital to my sanity. To my parents, Patricia and Richard, and to my brother, Killian, thank you for your lifelong support, belief in my capabilities, and for instilling in me the values that guided me through this PhD. Each of you has contributed uniquely to my growth, and I am forever grateful for your presence in my life.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification. Most work is based on conference papers, stated in the introduction of the relevant chapter:

- Robert Flood, Gints Engelen, David Aspinall, et al. Bad Design Smells in Benchmark NIDS Datasets. In 2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P). [81]
- Robert Flood and David Aspinall. Measuring the Complexity of Benchmark NIDS Datasets Via Spectral Analysis. In 2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) [79]
- Robert Flood, Marco Casadio, David Aspinall, and Ekaterina Komendantskaya. Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models. In Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing [82] (Awaiting Publication)
- Robert Flood, Marco Casadio, David Aspinall, Ekaterina Komendantskaya. Generating Traffic-Level Adversarial Examples from Feature-Level Specifications. In European Symposium on Research in Computer Security (ESORICS) [80] (Awaiting Publication)

Robert Flood

Contents

Nomenclature

C&C Command and Control

DNS Domain Name System

DSL Domain Specification Language

GAN Generative Adversarial Network

IAT Inter-arrival Time

ICMP Internet Control Message Protocol

ML Machine Learning

NIDS Network Intrusion Detection System

PGD Projected Gradient Descent

TCP Transmission Control Protocol

UDP User Datagram Protocol

VAE Variational Autoencoder

Chapter 1

Introduction

"How smart's an AI, Case?" "Depends. Some aren't much smarter than dogs. Pets. Cost a fortune anyway."

William Gibson
Neuromancer

Networks are vitally important in our increasingly digital world, serving as the backbone for everything from personal communications to critical infrastructure. Securing these communications against malicious activities, such as Distributed Denial of Service attacks or ransomware, is essential to maintain their privacy, integrity, and availability. Network Intrusion Detection Systems (NIDS) play a crucial role in this security landscape by identifying and mitigating unauthorised access and attacks.

Machine learning (ML) applied to network intrusion detection is a well-studied area. The field emerged in the mid-1980s following work by Denning and Neumann [61] and, and since then, ML techniques have evolved to enhance the detection capabilities by learning dynamic and complex patterns of network traffic. Although research interest has ebbed and flowed, network intrusion detection applications of ML frequently appear in top security and machine learning conferences alike. Rather than manually define good-and-bad network behaviour, ML-based NIDS use large corpuses of data to learn probabilistic models of network traffic. These models can then be used in place of a traditional signature-based NIDS, aiming to achieve better performance with less maintenance. Although this data is hard to collect due to privacy concerns, a set of synthetic public datasets, discussed in Section 2.3, have bolstered ML-based NIDS

research by providing a series of common benchmarks that can be used to compare and contrast model efficacy. Importantly, researchers are deeply interested in the *generalisation* capabilities of machine learning models i.e., the ability of ML-based NIDS to detect *out-of-distribution* or *never-before-seen* attacks.

However, despite continuous research interest in ML-based NIDS, commercial applications of the technology are rare. Currently, organisations hesitate to adopt these systems because they are uncertain about their reliability and ability to perform effectively under diverse and evolving network conditions. Additionally, ML-based NIDS are often less interpretable than traditional rule-based systems, making it difficult for users to understand and have faith in their decision-making processes. In other words, the issue is one of ***trust***. There are many causes of this, but this thesis investigates two factors: the limitations of benchmark research NIDS datasets and complex model architectures, which are difficult to interpret. Critically, in order for ML-based NIDS research to be applied in a trustworthy and dependable manner, we must be able to identify and mitigate weaknesses both in the benchmark datasets and in proposed model architectures.

The trust gap between research methodologies and real-world adoption can be seen in the above example of generalisation. Although generalisation is frequently referenced as a driving motivation for ML-based NIDS, research rarely concretely demonstrates, measures or evaluates this. Instead, since machine learning models have successfully generalised beyond their training data in other fields, it is often assumed that network security can benefit similarly. However, unlike other domains, the dynamic nature of network environments, coupled with continuously evolving threat vectors and diverse attack surfaces, complicates the evaluation process. Simple train/test splits are not sufficient to demonstrate meaningful model generalisation, rendering the experimental results untrustworthy. This thesis addresses these challenges by enhancing the understanding and development of more dependable ML-based NIDS.

By scrutinising existing benchmark datasets for flaws and proposing methods to measure their impact, this work lays a foundation for more accurate and relevant data usage in model training in research, providing better benchmarks for understanding model effectiveness. Furthermore, by employing neural network verification techniques, this thesis advances how we can design smaller, simpler models that are not only robust in learning from these datasets but also resilient under adversarial conditions. Through these efforts, this research contributes to crafting more reliable intrusion detection systems.

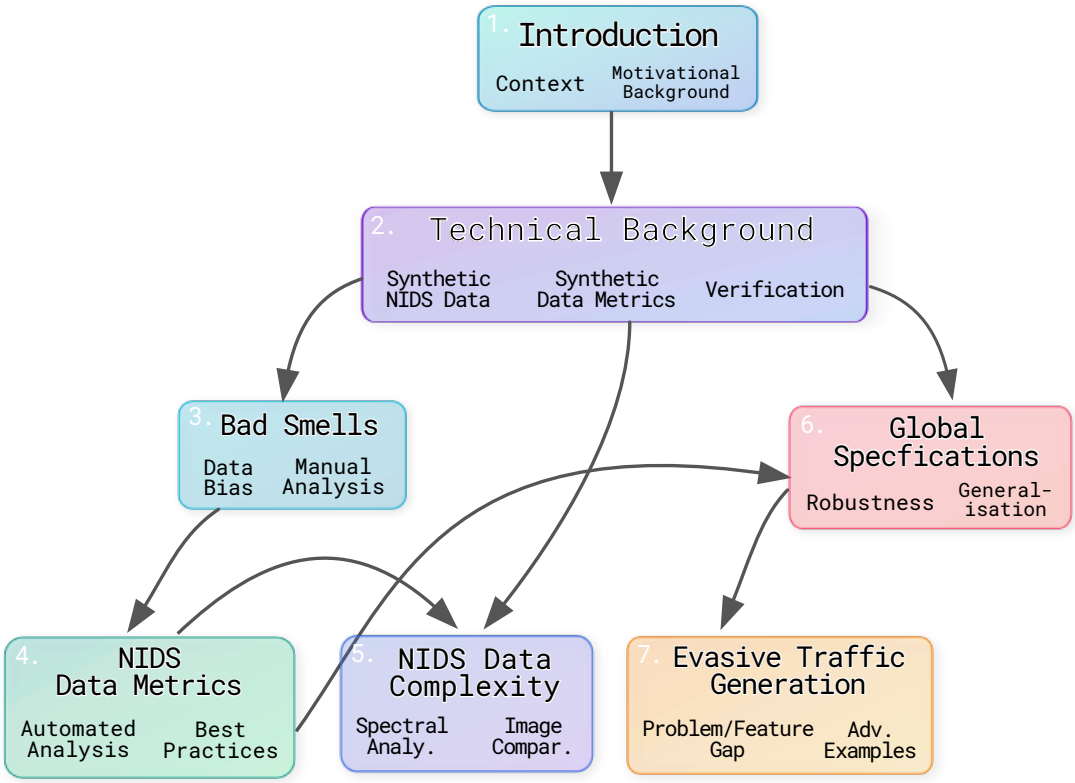


Figure 1.1: Thesis Structure and Chapter Relations

In Chapter 3, we present our initial criticism of benchmark NIDS datasets, defining the concept of *Bad Data Design Smells*. Bad data design smells — such as *Highly Repetitive Data* — stem from the improper generation/processing of network data, introducing experimental bias. We demonstrate the negative impact of bad smells on downstream research via a combination of a systematic literature overview of 40+

top conference papers and four in-depth case studies. We continue this discussion in Chapter 4. Here, we uncover bad data design smells in NIDS datasets via a two-pronged approach, combining a systematised manual analysis process with six automated, heuristic measurements, each related to a bad design smell. We highlight concrete issues with these datasets that, despite their ubiquity, have been undiscovered thus far, such as a malicious class of CTU-13 consisting of 99.9% malformed flows. We also demonstrate that bad data design smells are more common in the NIDS datasets than other anomaly detection datasets by applying these measures to domains. Together, these chapters present an overview into how the ML techniques are impacted when synthetic network data does not adequately resemble a real-world threat environment, complicating research conclusions that rely on such datasets. While we highlight dataset realism deficiencies, measuring it concretely remains challenging. As a proxy, in Chapter 5, we measure the complexity of network data via spectral clustering. We compare benchmark NIDS datasets with simple benchmarks in other areas, such as image recognition, placing the complexity of these NIDS datasets into a cross-disciplinary context. We show that, according to our measure, the input complexity of NIDS data is trivial compared with datasets understood to be simple, such as MNIST, further confusing research outcomes relying on these datasets. In Chapters 6 & 7, we use neural network verification to improve our ability to reason about the training and evaluation of ML-based NIDS. We encode domain knowledge via a set of *global specifications* and enforce that our models adhere to these properties via an adversarial training procedure. We formally verify model adherence, improving the cross-dataset generalisation of our models by 35%, while establishing adversarial robustness, improving trust during inference. We carry out eight experiments to show the benefits of globally specifying model behaviour, such as showing how prior work in certified robustness [221] fails in some settings. Second, we exploit the counterexamples produced by verification frameworks to produce constrained, realisable adversarial examples for NIDS. We then confirm the validity of these counterexamples by characterising them as perturbations of a Markov

chain representation of the source traffic. Doing this, we produce valid, evasive traffic at the PCAP level and show the benefits of verification-based adversarial attacks over PGD, a standard gradient-based approach. We diagram the relationships between these chapters in Figure 1.1.

Ultimately, this research seeks to bridge the trust gap, contributing to the advancement of ML-based NIDS that are both interpretable and reliable, supporting their widespread adoption in critical network security applications.

1.1 Motivational Background: A Comparison with Benchmark Image Datasets

Interrogating benchmark datasets and evaluating their bias is well-established in other fields, particularly image recognition. For instance, despite the fact that benchmark datasets such as PASCAL [74], LabelMe [173] and ImageNet [60] all contain images of cars, the specific characteristics of what constitutes a 'car sample' varies between each set. A dataset that, say, contains commercial photographs of individual passenger cars versus amateur photographs of groups of cars in everyday driving scenarios may lead to different biases when generalised beyond their training distribution. Here, to motivate our research into NIDS dataset flaws, we juxtapose it with prior research into dataset bias in computer vision. Although this will provide insight into the issues with benchmark NIDS datasets, we will see several reasons why the exact lessons learned from computer vision cannot be applied directly to network security data. Note that we focus on *experimental* bias inherent to datasets; despite similar terminology, this is distinct from, say, racial or class-based bias.

In *An Unbiased Look at Dataset Bias* [215], Torralba & Efros quantify dataset bias using several measures: dataset classification, cross-dataset generalisation and negative set bias. Despite the high overlap between class labels, each of these criteria found strong dataset bias, suggesting that training on a given dataset produces a model with strong built-in biases. Generally, Torralba & Efors recommend minimising bias by relying on multiple datasets in a thoughtful manner, a sentiment we echo in

Chapter 4. In contrast, Khosla et al. [116] propose a framework to mitigate dataset bias by training a model that jointly learns a common “visual world" weight vector and individual dataset-specific bias vectors. This is achieved within a max-margin learning framework similar to support vector machines. The objective function is composed of two main terms: one for the visual world model — which is trained using samples from all datasets to capture common features — and another for the dataset-specific biases — which are adapted using samples from their respective datasets to account for unique dataset-specific characteristics. This approach allows for effective generalisation on novel datasets while retaining specific adaptations for seen datasets.

Surprisingly, dataset bias can differ between datasets produced in near identical ways. 10 years after the original paper, Liu & He [136] revisit the Torralba & Efros dataset classification experiment on a new ‘generation’ of image datasets, largely sourced via web crawling. Again, despite the common approach to data collection, supervised learning methods far outperform random guessing. Liu & He also use a feature extraction process based on masked autoencoders as a front-end to a simple linear model, a process they call *linear probing*, to provide evidence that their models are learning generalisable dataset features, rather than rely on memorisation, achieving classification accuracies in excess of 80%.

Unfortunately, it is difficult to apply these insights directly to NIDS datasets as there are few agreed-upon standards in the field. For instance, although multiple datasets may contain ‘*backdoor*’ traffic, poor provenance and bespoke feature sets make it is unclear whether these refer to the same attack. Furthermore, for image datasets, once bias is accounted for there should be a strong relationship between classes with the same label. In NIDS, this assumption does not hold: aspects specific to each dataset such as bandwidth, network topology and background services, means that there are multiple valid representations of, say, SSH traffic and differences between datasets cannot be automatically categorised as bias.

1.1.1 The Iterative Development of Datasets

Alongside their experiments, Torralba & Efros provide an opinionated view of the history of image datasets, remarking that each subsequent dataset can be viewed as a reaction against the biases of the previous cohort:

> *Any good revolution needs a narrative of struggle against perceived unfairness and bias, and the history of dataset development certainly provides that. From the very beginning, every new dataset was, in a way, a reaction against the biases and inadequacies of the previous datasets in explaining the visual world . . . Caltech-101 was partially a reaction against the professionalism of Corel's photos, and an embrace of the wilderness of the Internet. MSRC and LabelMe, in their turn, were a reaction against the Caltech-like single-object-in-the-center mentality, with the embrace of complex scenes with many objects. PASCAL Visual Object Classes was a reaction against the lax training and testing standards of previous datasets. Finally the batch of very-large-scale, Internet-mined datasets – Tiny Images, ImageNet, and SUN09 – can be considered a reaction against the inadequacies of training and testing on datasets that are just too small for the complexity of the real world.*
>
> *On the one hand, this evolution in the development of datasets is perhaps a sign of progress. But on the other hand, one could also detect a bit of a vicious cycle . . . What seems missing, then, is a clear understanding of the types and sources of bias, without which, we are doomed to repeat our mistakes.*

This pattern has continued with very-very-large-scale datasets such as LAION-5B [182] labelled using natural language techniques, an acknowledgement that simple object categories are not enough to accurately describe images.

Examining the development of synthetic NIDS datasets in Section 2.3, there is a similar, albeit much briefer, process of reactionary development: KDD Cup 1999 [3] can be seen as a reaction to the private, unpublished datasets that were commonplace in intrusion research. In turn, NSL [210] emerged as a stop gap solution to the simplicity of KDD Cup 1999, intentionally introducing artificial bias to complicate the benchmark. Finally, the bulk of popular NIDS datasets released between 2012 and

2019 are a reaction to age of KDD Cup, which was increasingly seen as outdated and obsolete. Naturally, given how rare network datasets are and how short this reactionary development process has been, we can expect benchmark NIDS datasets to be less mature than their image counterparts.

However, this only partially explains the failings of NIDS datasets. Importantly, for image datasets, each layer of this reactive process is grasping towards some notion of '*realism*'. Although this realism may have been ill-defined at first, enough iterations of dataset development has honed in on complex benchmarks that generalise to real-world tasks directly. Nothing of the sort can be said for NIDS datasets. NIDS dataset authors often focus on iterative changes that are entirely tangential to the realism of the underlying data. NIDS dataset documentation often boasts about the size of the dataset, the number of attack classes, the recency of the attack data and the number of hosts. Meanwhile, concrete statistical evidence measuring the dataset's relationship to real-world data is scant. By focusing on a narrow understanding of realism, more fundamental questions about what intrusions look like in the wild, how features should be extracted and processed, how multi-stage attacks function and, most importantly, what best research practises should be followed are ignored.

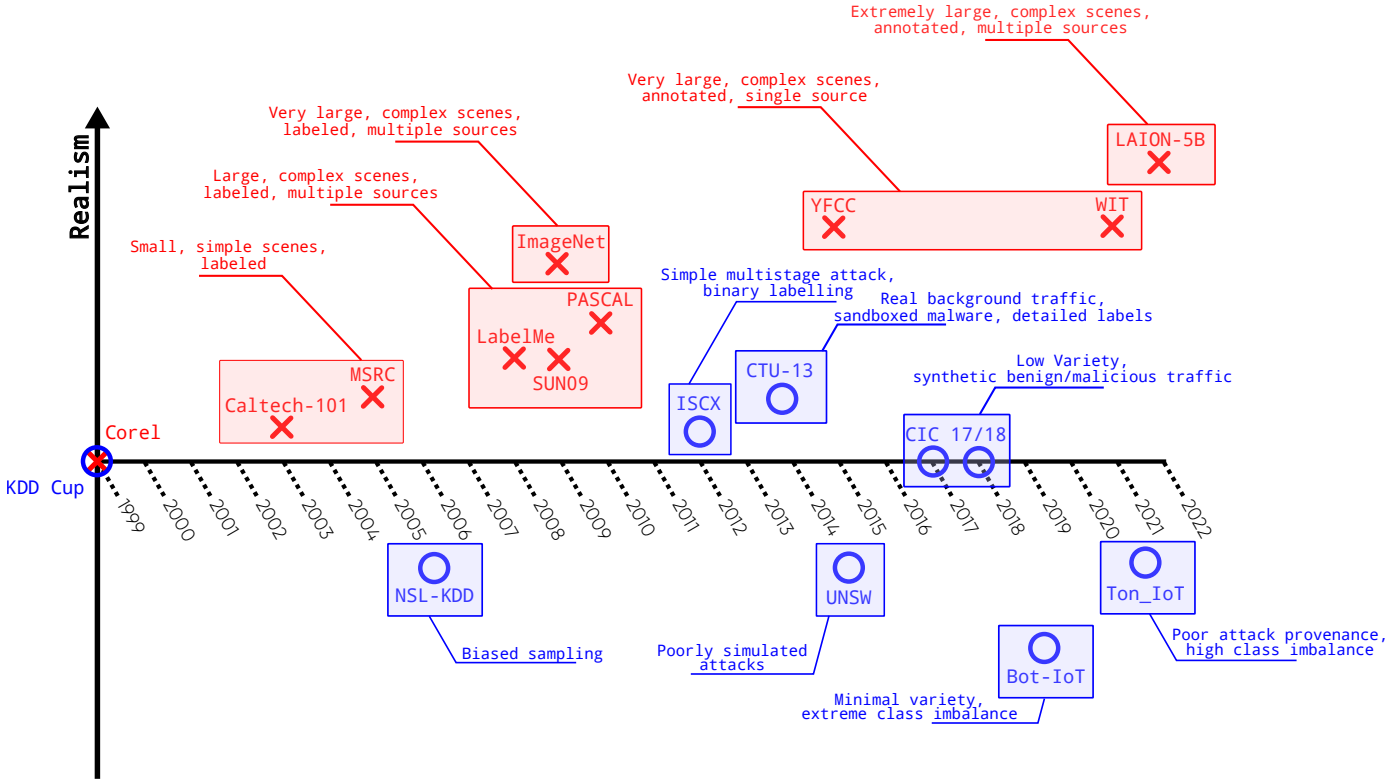


Figure 1.2: A comparison between image and NIDS datasets, detailing the progression/regressions relative to benchmark datasets released in 1999. Image datasets continually progress as the limitations of previous datasets are identified and resolved. As time progresses, the number of samples, scene complexity, source diversity and label granularity all increase, corresponding to improved realism. In contrast, the flaws of prior benchmark NIDS datasets are not resolved by subsequent datasets and are often exacerbated. Furthermore, improvements, such as CTU-13's granular labelling scheme, are not capitalised on and fall by the wayside. Despite 22 years of dataset development, it is unclear how Ton_IoT is a better benchmark than KDD Cup.

1.2 Motivational Background: Model-First NIDS, Biased Data and Trust

A significant thread in modern NIDS research is the pursuit of increasingly sophisticated machine learning architectures. Inspired by breakthroughs in other domains, such as the progression from variational autoencoders (VAEs) to generative adversarial networks (GANs) to diffusion models, *model-first* methodologies in NIDS have spawned a plethora of specialised solutions, including recurrent neural network approaches for multi-stage attacks [70, 194], graph neural network–based systems [138, 219], and adversarial training frameworks aimed at countering evasive threats [105, 190, 220]. While these innovations undeniably push technical boundaries, they also tend to heighten architectural complexity without addressing a more fundamental concern: the questionable quality of underlying datasets and the mistrust this can foster in real-world deployments.

As noted in other fields, it appears that "*Everyone wants to do the model work, not the data work*" [176], and many NIDS papers pay only cursory attention to the input data itself. This contrast is striking given the relationship observed in computer vision where model complexity and benchmark data evolved together. In network security, however, high-performance results on synthetic or poorly curated datasets risk creating a false sense of progress. Issues like encryption, partial network visibility, concept drift, and the rarity of malicious events are rarely captured by standard benchmarks. As a result, **model-first methodologies are currently difficult to justify in NIDS**: despite notional accuracy gains, real-world utility remains questionable when data is unrepresentative or riddled with artefacts. Indeed, as noted above, **the success of model-first approaches in other fields has grown in tandem with improved dataset standards**. This has yet to transpire in NIDS research.

Moreover, the rapid proliferation of model-first solutions, such as self-learning approaches [10, 18] or elaborate ensemble frameworks [149, 224], can further erode user confidence. Each new architecture may yield incremental improvements on a fixed set of benchmarks, yet remains opaque and poorly interpretable [157, 227]. As

a result, patterns learned by models may be incidental to the data they are trained on — a common issue which we measure in Chapter 4 — and may be tangential to the underlying operation of an attack — which we ameliorate in Chapter 6. In a security-critical environment, such opacity severely undermines operational trust. While researchers rightly explore innovative techniques, absent rigorous and realistic data, these systems lack a solid empirical foundation for wider adoption. Indeed, the attempts to mitigate adversarial examples by imposing network constraints [190] or applying packet-level manipulations [105, 189] reiterate the same challenge: the model may look robust on paper, yet it is unclear how faithfully the dataset and experimental setups capture real-world intrusions. While complex, domain-specific architectures may one day produce highly capable NIDS, their current value is hindered by the paucity of comprehensive benchmarks and the lack of widely accepted standards for data collection and labelling.

Against this backdrop, this thesis makes the case for a data-first approach. Rather than continuously stacking layers of complexity, we emphasise the foundational role of credible, high-quality datasets and transparent evaluation practices. We demonstrate this principle in Chapters 3 and 4. Under this paradigm, even simple, shallow models can yield strong performance, while offering a more suitable foundation for applying neural network verification, an approach that does not scale readily to large or complex architectures. By using verification in conjunction with targetted training methods, we improve model adversarial robustness, cross-dataset generalisation and interpretability.

1.3 Main Contributions

The main contributions of this thesis include:

- A systematic review process for evaluating design flaws in network intrusion datasets. To our knowledge, this is the first critique and analysis of *design* issues in network intrusion datasets. Furthermore, we concretely link dataset design decisions to questionable practises that degrade experimental quality, highlighting case studies where, for instance, classification accuracy can degrade by over 90% when minor artefacts are accounted for.

- Several novel metrics for evaluating the complexity and quality of network flow data. We employ these metrics to strengthen the above criticism, as well as compare network intrusion data to datasets in other fields, including real-world network traffic. We consistently find that network intrusion data has minimal complexity compared to data from other fields.

- A methodology for encoding network traffic properties as global specifications for neural networks, improving model cross-dataset generalisation accuracy by over 35%. Again, to our knowledge, we are the first to apply neural network verification to the network intrusion domain or to network traffic more generally.

- A novel approach to generating realisable adversarial examples corresponding to evasive network traffic while adhering to domain constraints. Due to the gap between the feature-space and problem-space in network traffic, feature-level perturbations are difficult to translate back to the packet-level. This has been highlighted as an issue for the field, which we tackle by producing realistic, realisable adversarial examples at the traffic-level with greater accuracy than state-of-the-art feature-level attacks.

Chapter 2

Background

In this chapter, we cover the relevant background for this thesis. First, we summarise foundational information in Sections 2.1– 2.4, including network data formats, relevant benchmark datasets and details about NIDS. We include the historical origins of the field, which provide context for the long-lasting difficulties with NIDS research. As most data used in this thesis is synthetic, in Section 2.5 we describe NIDS data generation techniques alongside synthetic data metrics. This provides a backdrop for the measurements used in Chapters 4 and 5. Finally, we introduce neural network verification techniques in Section 2.7 which are used extensively in Chapters 6 and 7.

2.1 Network Data

Network traffic is commonly recorded in two formats: *pcap* data and *network flows*. *pcap* data, or *packet capture* data, is the raw, unprocessed network packets, each containing protocol-dependent *header* data, such as IP addresses, TCP flags or ICMP type, as well as an optional *payload* containing user data, as seen in Figure 2.1. Packets can be recorded, and potentially replayed, using tools such as tcpdump or tcpreplay.

In contrast, network flows compress this raw data by grouping together packets with the same source/destination IPs, source/destination ports and network protocol — typically within some time window — which correspond to a single network connection. High-level statistics can then be calculated from each packet sequence, such as, say, the flow's duration. Flow statistics aim to simplify analysis by summarising data over time. Whilst the shared attributes of network flows are fixed across implementations, different frameworks represents flows using a unique set of statistics. For instance,

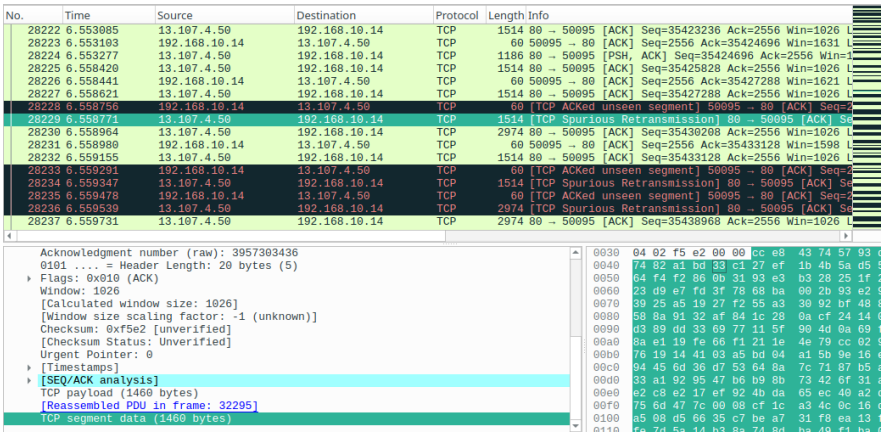


Figure 2.1: A pcap file showing a sequence of TCP packets (center), packet metadata (bottom left) and packet payload (bottom right).

Cisco NetFlows [206] differ from CICFlowMeter [1] flows. The former is optimised for speed, sampling packets intermittently with minimal post-processing, while the latter contains a more detailed set of processed statistics, such as the *Standard Deviation of Forward Bytes*[1]. Unlike raw network traffic, flow statistics can be fed into machine-learning pipelines with minimal preprocessing and these statistics are often used directly as the '*features*' for these algorithms to automatically learn classification criteria. To gauge the accuracy of these models, a 'label' indicating the threat nature of the flow is also provided. The criteria for labelling flows can be opaque or unclear, which we discuss Chapter 3.

With the exception of Cisco NetFlows, the reasoning behind extracting a given set of features from flows are poorly documented. It appears that tools like CICFlowMeter calculate as many statistical features as possible in order to thoroughly summarise each flow. Since these are likely intended near-exclusively for training machine learning models, it is left to the algorithms to assess their relevance for a given task.

[1]The standard deviation of the size of packets from the source to destination direction

2.2 Network Intrusions

There are many classification systems and taxonomies for categorising network intrusions, such as Mitre ATT&CK [52]. As discussed in Chapters 3 and 4, depending on the attacks included, the design characteristics of a NIDS dataset can vary highly which, in turn, implicitly defines qualities such as generalisation criteria or performant model architectures. In this setting, an important distinction is the difference between *volumetric* attacks and *non-volumetric* attacks.

Volumetric Volumetric attacks necessitate large quantities of traffic and typically scale with the amount of traffic produced. Due to the relative ease of automating these attacks, simple tooling can produce highly repetitive traffic, leading to highly skewed datasets. Real-world examples include the Mirai botnet attacks of 2016, which overwhelmed DNS provider Dyn with millions of flows originating from compromised IoT devices [12]. These attacks typically appear on the network as sudden surges of traffic from numerous distributed hosts.

- **DoS Attacks**: Volumetric Denial-of-Service (DoS) attacks are a primary category within the broader DoS attack spectrum, designed to exhaust the bandwidth of the targeted network or system. These attacks send a massive volume of seemingly legitimate requests that may exploit idiosyncrasies in the server or network specification, causing the victim to use a disproportionate amount of computational power or network bandwidth. Tools such as LOIC [156] can rapidly generate high volumes of requests, often resulting in predictable patterns, e.g., a massive spike in identical packets, making rudimentary forms of detection feasible. Terminology varies slightly depending on the number of malicious hosts: attacks that rely on large numbers of attack devices working in tandem are referred to as Distributed Denial-of-Service (DDoS) or Botnet attacks.

- **Enumeration Attacks**: Also referred to as fuzzing attacks, enumeration involves probing a network or server for hidden resources (e.g., usernames, directories, or APIs). In practice, tools like dirb [2] or GoBuster [167] can produce tens of thousands of attempts within minutes, triggering distinctive patterns of rapid, systematic requests. Detection signatures may include suspicious regular intervals of requests, repeated 404 errors, or large bursts of failed connection attempts.

- **Reconnaissance Attacks**: Reconnaissance denotes information-gathering activities, though some variants, such as repeated port scans, are highly volumetric. Tools like nmap [140] can send large numbers of packets in a short period, creating easily recognisable spikes in connection attempts across multiple ports. While signature-based NIDS can flag these abrupt bursts, more advanced scans remain harder to detect, as attackers often randomise timing or traffic volume to evade simpler rules, with tools providing a wide breadth of configuration options.

Non-volumetric By contrast, non-volumetric attacks do not rely on immense traffic volume but are more targeted, often leveraging knowledge of system vulnerabilities. These attacks often involve nuanced traffic signatures or payloads, making them trickier to detect with purely volume-based heuristics, such as standard flow-based features. Here, we summarise a selection of non-volumetric attacks that are seen in NIDS datasets.

- **XSS Attacks**: Cross-Site Scripting (XSS) embeds malicious scripts into web pages viewed by other users. These attacks exploit vulnerabilities in web applications that fail to adequately sanitise user input, allowing attackers to inject unauthorised, executable code (usually JavaScript) into a website's output. This is challenging for flow-based NIDS, as detection usually requires inspecting packet contents or metadata (e.g., unusual parameters in HTTP payloads) for the presence of malicious scripts.

- **SQL Injection**: SQL injection attacks manipulate backend database interaction by embedding malicious SQL statements into an input query to exfiltrate, modify, or delete data. While volumetric scanning may precede the actual injection attempt [57], the final exploit is often a single crafted packet or series of queries. Signs of SQL injection on the network include suspicious sequences of characters (e.g., "OR 1=1" or "UNION SELECT") passing through HTTP or TCP streams. Again, this is difficult for flow-based NIDS to detect.

- **'Zero-day' attacks**: 'Zero-days' are a class of attacks that rely on vulnerabilities that are — at the time of use — unknown with no corresponding patch, i.e., vendors have 'zero days' to fix the issue. Although there is no overarching statistical similarities between different zero-days, network intrusion detection research sometimes places particular emphasise on them as motivation: if machine learning classification can generalise in domains such as computer vision, then it is possible that network security could benefit similarly. Chapter 4 discusses how this assumption is not necessarily justified and is poorly supported by existing dataset design, as highlighted by other work [14].

2.3 Benchmark NIDS Datasets

To evaluate the performance of NIDS at detecting malicious traffic, agreed-upon benchmark datasets are needed, as is the true for all ML domains. For instance, in computer vision, researchers can test methods on comparatively simple datasets such as MNIST [123] or CIFAR10 [120] before scaling to more involved datasets such as ImageNet [60] or LAION [182]. Benchmark datasets allow researchers to compare methods on a like-by-like comparison and measure incremental progress.

Unfortunately, building a dataset using real-world traffic is difficult; network traffic gathered from an enterprise network would almost certainly contain both privacy- and security-sensitive information which, if released, would violate data privacy laws or provide malicious actors with footholds to the network. Moreover, network traffic is

noisy and difficult to automatically process, requiring tedious, manual labelling from domain experts. This situation is made more difficult by the fact that network intrusions are inherently secretive, making it difficult to source up-to-date information about malicious behaviour. Real-world datasets are thus rare and heavily anonymised.

As a result, the majority of network datasets contain **synthetic** data, defined by Jordan et al. [109] as '*data that has been generated using a purpose-built mathematical model or algorithm, with the aim of solving a (set of) data science task(s)*'. Several *model*-based synthetic data generators, such as VAEs, GANs and Diffusion models, have been used to generate synthetic network data. However, these methods are limited. Thus, NIDS datasets are typically generated in an *emulation*-based manner by scripting benign/malicious behaviours. These behaviours can then be played across a generation testbed — often a series of virtual machines — and the resultant data collected and processed. We provide a fuller treatment of synthetic data and its utility in Section 2.6.

In this section, we provide an overview of the NIDS datasets used or referenced throughout this thesis. We show a timeline in Figure 2.2.

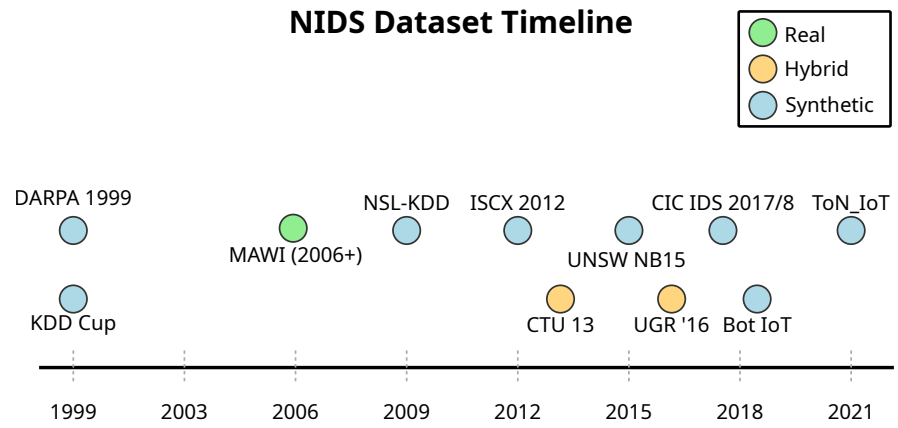


Figure 2.2: A timeline of the approximate release dates of NIDS datasets discussed in this thesis.

DARPA 1999 [131], KDD Cup [3] & NSL-KDD [210] Released by the MIT Lincoln Laboratory in 1999, *DARPA 1999* consists of both background and malicious traffic captured from a simulated airforce base network. Attacks are subdivided into four classes: Probe, DoS, R2L (*Remote-to-Local*, i.e., attacks which allow initial entry into the network) and U2R (*User-to-Root*, i.e., privilege escalation attacks).

Needless to say, a 25 year old dataset does not represent a modern network or threat environment; almost all of the attacks are outdated, targeting operating systems such as Windows NT, and much of the background traffic relates to protocols that are now obsolete, including telnet and finger. Despite this, modified versions of DARPA 1999 are still used as benchmarks in contemporary NIDS papers.

KDD Cup is a modified version of DARPA 1999 originally released as part of a data mining challenge accompanying the SIGKDD 1999 and likely was not intended to become a de facto benchmark dataset. As a result, there is no paper describing its contents or how it differs from DARPA 1999. Furthermore, research quickly highlighted flaws with the dataset [144]. Despite these issues — as well as the dataset's age — it is still used as a benchmark for research NIDS today.

NSL-KDD is, in turn, a modified version of KDD Cup that attempts to ameliorate its issues and present a more challenging NIDS benchmark. NSL KDD removes redundant records and oversamples difficult-to-classify malicious flows, based on the classification results of 21 baseline algorithms. However, many problems with KDD Cup stem from its design and are difficult to correct following its release.

ISCX 2012 [196] & CIC IDS 2017/8 [186] ISCX 2012, CIC IDS 2017 and CIC IDS 2018 are a series of similar datasets released by the University of New Brunswick, the latter two developed in conjunction with the Canadian Institute of Cybersecurity (which we call the '*CIC datasets*'). These datasets are ubiquitous in NIDS research, with over 3000 citations [2]. Despite this, the datasets are known to be flawed, containing serious mislabelling issues and endemic feature artefacts [73, 133].

[2] according to Google scholar, 26/01/25

The CIC datasets share several similar design aspects, such as attacks and generation methodology. The authors of these datasets emphasize their role as '*benchmark*' datasets and prioritize the '*realism*' of the background traffic and the '*diversity*' of attacks. In both datasets, the benign and malicious traffic is generated via B- and M-profiles respectively, which aim to mimic properties of the CIC network. However, it is unclear what attributes are replicated exactly. The feature set is also shared between the datasets, consisting of 80 flow statistics calculated using CICFlowMeter [1]. The main difference between the datasets is size: CIC IDS 2017 has approximately 14 hosts, whereas CIC IDS 2018 has over 500.

UNSW NB15 [151] Released by the University of New South Wales, UNSW NB15 is a dataset consisting of both synthetic benign and malicious traffic. Importantly, the attack data was generated using the IXIA PerfectStorm tool, a network traffic generator primarily aimed at testing network infrastructure and load tolerance. Although advertising copy for PerfectStorm claims that it can generate intrusion data — which IXIA call 'strikes' — it is unclear how realistic these are. The feature set is divided into four categories: basic, content, time and additional features, consisting of protocol specific flags and connection rate-based features. The dataset's design emphasises its recency; the authors say that it contains 'contemporary synthesized attack activities' and they claim that the dataset is more complex than *KDD Cup* [3, 152].

CTU 13 [85] CTU 13 is a dataset containing network traces of malicious bots, including the Neris botnet and the Murlo malware, from intentionally infected hosts. CTU 13 differs from the other datasets listed here as it contains 13 'scenarios', each containing distinct malware flows, which have been labelled in a highly granular manner.

ToN_IoT [150] Ton_IoT is a group of datasets collected from a network testbed containing several internet of things sensors. Alongside the telemetry data from these sensors, ToN_IoT contains a standard intrusion dataset, containing network attacks such as password bruteforcing, man-in-the-middle attacks and the network traces of ransomware. In a supplementary paper, ToN_IoT is described as as having high heterogeneity according to a set of statistical tests [29]. However, these tests are poorly defined, of unclear relevance and often discredit the dataset's heterogeneity. For instance, calculating the Kolmogorov–Smirnov statistic [118] between each feature and a nonsensically-defined distribution demonstrates that the training and test sets are nearly identical. Similarly, the original paper shows that both a simple neural network and a random forest are capable of achieving near-perfect classification results.

Bot IoT [119] Similar to ToN_IoT, Bot IoT contains data from IoT devices, including a weather station and a smart thermostat, and six attack classes: service scanning, OS fingerprinting, DDoS, DoS, keylogging and data theft. These are described using a set of 43 custom features, which were selected to maximise model performance. The original paper detailing the design of the dataset shows that it is possible to get a perfect F1 score using a standard LSTM. The dataset is profoundly imbalanced, almost exclusively consisting of volumetric attack data; out of a total of 72 million flows, approximately 1000 are benign. The paper describing *Bot-IoT* stresses realism as a design goal, stating that it is a '*realistic . . . dataset*' with a '*massive amount*' of '*realistic benign traffic*'. The authors provide two variants of *Bot-IoT*: a full version with a truncated feature set and a condensed version with an additional 16 aggregate features, such as the number of packets per IP.

UGR '16 [141] UGR '16 dataset is a large dataset created by a collaboration between the University of Granada and an ISP. It includes both real user-generated traffic and artificially generated traffic that simulates normal and attack scenarios. UGR '16 contains several weeks of background traffic and the accompanying documentation claims that the dataset is uniquely suited to evaluating NIDS that consider the long-term evolution of traffic or traffic periodicity, such as day/night cycles.

MAWI Working Group Traffic Archive [46] The MAWI traffic archive is a series of network traffic datasets collected from the WIDE backbone, a large-scale research network that connects Japanese research institutes. The traffic archive consists of daily captures, each 15 minutes long, and has been operating since 2006. Captures are supplied as truncated pcaps (preventing any deep packet inspection).

Although an extremely useful resource, the MAWI traffic archive provides little insight into the structure of an enterprise network, a more standard setting for NIDS. Furthermore, although a subset of these captures have labelled anomalies, these are detected via simple heuristics and low-volume attacks are likely not present.

2.4 Network Intrusion Detection Systems

The development of NIDS, is an active research area and has been for many decades. Early work in the 1980s by Denning and Neumann [61, 62] proposes a framework for detecting intrusions and outlines desirable properties of an NIDS, such as the ability to self-teach and high discriminative power. Early research into NIDS can be divided into two categories: *signature-based*, which flag traffic as being malicious according to some pre-defined rules, and *machine learning*-based, where malicious criteria are automatically inferred from a training dataset.

Signature-based methods are commonly used in industry, with Suricata [158] and Snort [172] being popular commercial options. However, signature-based approaches are known to have issues: signatures are fragile and can be evaded with simple substitutions [217]; generalising to unseen attacks is unlikely, requiring frequent updates to the rule set; rules must be manually written and false positives are common [8], necessitating a human-in-the-loop to distinguish potential threats from misidentified benign traffic.

Machine learning potentially offers a solution to these problems. By automatically extracting statistics that govern benign and malicious behaviour from data, an ML-based NIDS could autonomously infer decision criteria which, if sufficiently robust, could detect previously unseen intrusions, analogous to the generalisation prowess

of ML models in fields such as computer vision. Hoping to unlock this potential, most contemporary research into NIDS focuses on ML-based approaches, which can be divided into supervised and unsupervised methods. This thesis contributes to this strain of '*Research NIDS*'.

For supervised methods, an algorithm is trained on a labelled dataset that provides both the input features and the corresponding output labels. In NIDS, the input features are extracted from network traffic, and the labels indicate the threat nature of the traffic. The primary objective in supervised learning is to construct a predictive model that can infer the correct output for new, unseen instances based on the knowledge gleaned from the training data. The training process seeks to optimise a 'loss function' which measures its performance. It does this by adjusting model parameters, via, say, gradient descent, to minimise prediction errors. This prediction target can be either a class label for multiclass classification or a continuous value for regression.

In contrast, unsupervised methods, such as anomaly detection, do not require labelled data. Instead, unsupervised methods model the benign network traffic, marking traffic as malicious if it deviates too far from this model of normal behaviour. For network anomaly detection to be effective, it assumes that benign traffic adheres to some statistical distribution B. Given adequate benign data for training, an ML model can then replicate B as some approximation, $\hat{B}$. When new network traffic is ingested by the anomaly detector, these connections can then be checked against $\hat{B}$ to establish whether they are anomalous. Malicious traffic is assumed to be quantitatively distinct from benign traffic with respect to their features, adhering to some separate distribution M where the overlap between B and M is minimal. More rigorously, we could say for input sample x, we assume that B and M satisfy $\int_{-\infty}^{\infty} \texttt{min}(B(x), M(x))dx < \varepsilon$, where ε is some small threshold value (which is non-trivial to determine). Thus, x can be identified as anomalous if $\hat{B}(x) < \varepsilon$. As labelling network data is a labour-intensive and challenging process, unsupervised methods have been widely researched in network intrusion detection.

2.4.1 Historical Research NIDS

Early ML-based NIDS were limited heavily by contemporaneous computing power and difficulties sourcing datasets. Proposed systems included RIPPER [124], a greedy rule inference algorithm, shallow neural networks [83] and hidden Markov models, each using unique feature sets. However, following the release of the DARPA 1998 and KDD Cup 99 datasets, the first public labelled intrusion datasets, acquiring the necessary data became far simpler. These datasets quickly became common benchmarks, reducing the barrier for NIDS research. For instance, Mahoney and Chan [143] evaluate ALAD and PHAD, two Bayesian anomaly detectors trained on **a**pplication **l**ayer and **p**acket **h**eader features, on DARPA 1999 [3].

Notably, alongside raw network packets, KDD Cup 99 included a tabular dataset of 41 flow-level features. This decision significantly influenced later NIDS research, which often eschews feature extraction, instead relying on these pre-computed features. Furthermore, this allowed for *generic* statistical methods to be applied to intrusion detection, without domain-specific modifications. Examples of this include a kernel density-based anomaly detection method by Yeung and Chow [237], the independent component analysis of Yang and Qi [235], the active learning approach of Abe et al. [5], the modified decision trees of Reif et al. [168] and the kernel parameterisation of Gao et al. [84].

In the late 1990s, *stide* [226] was a state-of-the-art algorithm for detecting intrusions, including network intrusions, by monitoring sequences of system calls. Based on a sliding window, there was consensus that a window of size six or greater was needed to detect intrusions, based on empirical measure on various datasets [103]. However, little reasoning was provided as to why this value was effective and the question 'Why 6?' emerged. Later work by Tan et al. [208] established that the performance of stide was biased by the length of the *minimal foreign sequences* — sequences of system calls whose proper subsequences are present in the training data — in

[3] Before criticising the dataset in a later work [144]

Warrender et al.'s evaluation datasets. This was, arbitrarily, six, a synthetic dataset artefact completely disconnected from real-world intrusions. In Chapter 3, this work argues that, in NIDS research, data artefacts are insidious and can lead to misleading results. Reviewing this early literature, there is historical precedence for this viewpoint.

2.5 Synthetic Data

Synthetic data refers to artificially generated data that, unlike real data, is produced by computational models and algorithms, aiming to replicate the statistical properties of real-world data, thereby creating data that can be used to solve various machine learning tasks while maintaining important characteristics of the original datasets. The advantage of using synthetic data lies in its flexibility to introduce controlled modifications, which can be impossible with data drawn from real-world data streams.

Synthetic data is used in many areas, particularly where data privacy, bias, and volume are concerns. One primary advantage is enabling private data release — datasets that fall under strict privacy regulations can still be shared in a synthetic form without comprising privacy. Moreover, synthetic data can be used to de-bias data; by adjusting underlying biases in datasets, researchers can achieve better outcomes in machine learning models.

Throughout this thesis, we use synthetic network data frameworks to improve existing datasets (Section 4), to understand the complexity of benchmark datasets (Section 5), to evaluate arbitrary model generalisation (Section 6) and to produce evasive traffic (Section 7). Here, we discuss synthetic data and its evaluation.

2.5.1 Utility & Fidelity

Fundamentally, synthetic data is used in place of real-world data when it has some shortcomings or weaknesses. In order for synthetic data to account for these flaws, the data must be useful, it must be qualitatively similar to the real-world data and it must not leak private information from the original data. Thus, synthetic data is typically evaluated along three axes: utility, fidelity and privacy. As this thesis is less concerned with privacy as a goal, here, we focus our attention on utility and fidelity.

Utility indirectly measures the similarity of synthetic data versus real data by comparing their usefulness with respect to a given task. This is often done via measures such as precision, recall or fairness metrics [218] when a model is trained on synthetic data and tested on real-world data, or vice versa. High utility in synthetic data ensures it can serve as a reliable proxy for real datasets, yielding comparable results and insights for the task at hand.

In contrast, fidelity refers to how accurately synthetic datasets replicate the statistical properties of real datasets. Unlike utility, fidelity focuses on direct statistical comparisons, ensuring the synthetic data closely resembles the real data by comparing distributions. This property is crucial because synthetic datasets with high fidelity can serve as viable substitutes for real data, allowing analyses and machine learning models developed on the synthetic data to be transferable and relevant to real-world datasets. However, achieving high fidelity is not trivial; it involves balancing the statistical resemblance with practical constraints like privacy and the potential for replicating biases inherent in the original data. The need for data fidelity is often coupled with the need for data *dissimilarity*. Otherwise, a synthetic data generator can produce high fidelity data simply by recreating a small subset of the target data, contrary to sensible aims. Instead, the generated traffic typically must also be *diverse*.

2.5.2 Synthetic Data Fidelity Metrics

Early metrics attempted to encapsulate generative data quality with a single number. For instance, the *Inception Score* proposed by Salimans et al. [175] assumes a generator has high fidelity when the conditional label distribution $p(y|x)$ has low entropy and high diversity when the marginal $\int p(y|x = G(z))dz$ has high entropy. These concepts are then combined to produce a single metric:

$$\exp(\mathbb{E}_x KL(p(y|x)||p(y)))$$

However, note that the Inception Score does not consider a target data distribution, limiting its ability to compare the generated data to a real-world dataset. Thus, Heusel et al. propose using the Fréchet Inception Distance (FID) [101]. To calculate FID, Heusel et al. first assume that some feature extraction process (e.g., an autoencoder) can be used to transform both the generated and target distributions, $\mathscr{G}$ and $\mathscr{T}$ respectively, into multivariate Gaussians, $\mathscr{N}(\mu_g, \Sigma_g)$ and $\mathscr{N}(\mu_t, \Sigma_t)$ respectively. The FID is then calculated as the Wasserstein-2 Distance between these distributions:

$$||\mu_t - \mu_g||_2^2 + \text{tr}(\Sigma_g + \Sigma_t - 2(\Sigma_g \Sigma_t)^{\frac{1}{2}})$$

However, it is difficult to quantify many distinct concepts in a single metric, leading to situations where FID does not correspond with human intuition about generated data quality [135]. More recent metrics attempt to quantify fidelity and diversity use twin metrics such as Naeem et al's Density and Coverage metrics [155]. Given target samples T_i and generated samples G_j, Density and Coverage are based on the binary decision of how many generated samples reside in the k-Nearest Neighbour spheres over the target set — $B(T_i, \text{NND}_k(T_i)))$. Naeem et al. then define Density as the number of target neighbourhoods contain G_j:

$$\frac{1}{kM} \sum_{j=1}^{M} \sum_{i=1}^{N} 1_{G_j \in B(T_i, \text{NND}_k(T_i)))}$$

Similarly, Coverage is defined as the fraction of target samples that have at least one generated sample in their neighbourhoods:

$$\frac{1}{N}\sum_{i=1}^{N} 1_{\exists j\ s.t.\ G_j \in B(T_i, \mathsf{NND}_k(T_i))}$$

Sajjadi et al. suggest that generative model quality should be measured by generalising the concept of *precision* and *recall* [174]. This extension involves comparing the support of the distributions $\mathscr{T}$ and $\mathscr{G}$ to measure relative overlap, producing a precision-recall curve that varies based on penalising the non-overlapping regions. Alaa et al. propose modifying these to consider only a subsection of each distribution's support, weakening the impact of outliers on the metrics [6].

2.6 Network Data Generation Frameworks

Unlike static network datasets, data generation frameworks allow for the production of targeted data based on specific research needs. This has obvious advantages over the 'one size fits all' approach of static datasets. As discussed in Section 4, the low diversity of traffic in benchmark datasets limits their utility for evaluating concept drift detection. Synthetic traffic generators provide an alternative evaluation approach, allowing authors to explicitly define the purviews of concept drift that they consider and generate appropriate data. Furthermore, static datasets often fail to fully evaluate a model's performance due to their limited size. In contrast, by generating arbitrary traffic, NIDS failure modes can be better explored via model probing [48].

2.6.1 Model-based Data Generation

Model-based generators directly output synthetic traffic from a generative ML model. Several GAN-based generators have been proposed for augmenting or creating flows [130, 170] as well as packets in a limited capacity [44]. More recently, diffusion-based generators have emerged, such as NetDiffusion [108], a variant of the popular Stable Diffusion model. As Stable Diffusion is primarily an image generator, NetDiffusion is fine-tuned on traffic traces encoded as bitmap images [104]. However, these

models have not been applied to intrusion detection and their ability to generate security-relevant data has not been evaluated. Furthermore, GAN-based traffic generators have been criticised for their poor efficacy, with Bayesian-based approaches performing better on appropriate generative metrics [181].

Model-based generators may offer certain privacy advantages over static datasets, but this remains unclear. By restricting access to the original source dataset and instead providing a data generator that mirrors the distribution of the original data, the intention is that private data within the source is obscured in the generated output whilst maintaining utility. Some traffic generators offer differential privacy guarantees; however, due to the complexity of applying differential privacy to packet-based data, they exclusively generate aggregate flow statistics. Their effectiveness, however, has been called into question. Stadler et al. [202] have demonstrated that generative models are susceptible to *linkage attacks*, enabling adversaries to ascertain the presence of sensitive records within a source dataset. Furthermore, as highlighted by Sun et al. [204], many of these flow generators achieve differential privacy through DP-SGD, a modified version of stochastic gradient descent that clips training gradients and adds noise via the Laplace mechanism to create differentially private flows. Nevertheless, in certain settings, DP-SGD can introduce excessive noise because its threat model assumes differential privacy is required during each iteration of gradient descent. This, in turn, can severely impact the utility of the generated flows.

2.6.2 Emulation-based Data Generation

Emulation-based data generation is an alternative to model-based. Rather than generating data according to a target distribution, emulation-based approaches mimic user behaviour across a network testbed and record the resultant traffic. This approach has several advantages over model-based methods. First, the direct manner in which traffic is generated simplifies verifying traffic properties. For instance, modifying the bandwidth of a network connection will produce traffic that certainly conforms to that change, whereas a statistical approach might only approximate it, leading to potential inaccuracies. Additionally, emulation captures the complexity and variability

of real-world interactions, potentially providing richer datasets for evaluation. However, producing realistic traffic remains a challenge. The scope and variability of user-behaviour across different networks is massive and emulating all of these patterns is a monumental engineering challenge. Furthermore, emulation-based methods are inefficient, often requiring that data generation takes place in real time, i.e., two hours of background traffic takes two hours to emulate.

DetGen [49] is a traffic generation framework — largely developed during my MSc [78] — extensively used throughout this thesis to produce bespoke, contextually relevant data for experimental evaluation. DetGen generates traffic across a virtual network according to scripted interactions, called *scenarios*. To ensure minimal noise or overhead from background services, all DetGen hosts are isolated from one another using minimal Docker containers. This isolation facilitates the production of data that is deemed “deterministic," within the limits of networking and computational differences.

DetGen provides high generative control, enabling users to adjust traffic properties in a targeted manner, such as altering network bandwidths or packet retransmission rates. This capability is crucial in later sections, as it allows for the generation of valid network traces that better evaluate the generalisation performance of models. Furthermore, DetGen’s modular and scalable design allows users to independently configure, expand, and implement scenarios without disrupting the overall framework, accommodating a broad range of research requirements and evolving traffic patterns.

Alternatively, emulation-based data generation frameworks such as netUnicorn [24] offer significant flexibility and adaptability in data collection. Similar to DetGen, netUnicorn facilitates data generation within varied network environments but emphasises the iterative enhancement of the dataset’s quality through explainable ML tools. This iterative process continuously refines data collection intents based on model feedback, thus progressively eliminating biases and improving model generalisation, allowing it to separate data collection intents from their execution mechanisms. This enhances the ease of deploying and adapting scenarios across diverse infrastructures, allowing netUnicorn to be ran on both real-world and emulated networks.

The Intrusion Detection Dataset Toolkit [76] (ID2T) is an alternative traffic generation framework. ID2T supports two primary methods of attack generation: template-based and script-based. The template-based approach modifies existing network packets from a PCAP file based on user-defined parameters, preserving existing traffic patterns. Conversely, the script-based method constructs attack packets from scratch using detailed scripts that specify packet attributes. Whilst this allows for complex attack simulations, this approach requires manually defining packet parameters which prevent easily introducing new scenarios.

Finally, user and adversary emulation tools such as GHOSTS and CALDERA, the latter based off of MITRE ATT&CK, can be converted into traffic generation frameworks. Although these are not necessarily designed for this purpose, the resultant traffic can be collected and used to construct NIDS datasets [89, 90].

2.7 Neural Network Verification

This thesis relies on neural network verification tools, primarily Marabou [113] and Vehicle [55]. In this section, we discuss neural network verification generally, with particular emphasis on these tools.

Neural network verification consists of a set of methodologies aimed at formally ensuring predefined behavioural properties of neural network models written as constraints on model inputs and outputs, such as adversarial robustness or compliance with regulatory standards. This is computationally intensive due to the non-deterministic and often opaque nature of these models, and verification tools currently only scale to moderately sized networks. Leading verification methods are exemplified by Marabou and α, β-CROWN [222].

Based on a modified version of the simplex algorithm called ReLUPlex [112], Marabou is an exact verifier, i.e., it is guaranteed to terminate. ReLUPlex translates neural network verification tasks into equivalent sets of linear constraints, efficiently checking for satisfaction via an SMT solver. However, as the simplex algorithm is limited to linear operations, ReLUPlex requires that all activation functions in the network are ReLUs (where the activation of the j-th neuron in the i-th layer is given

as ReLU$(x_{i,j}) = \max(x_{i,j}, 0))$, which are handled separately as they introduce non-linearities to the verifier. As such, ReLUPlex initially fixes only the linear constraints of the network, violating the ReLU constraints if necessary. If this is satisfiable, then the ReLU constraints can be added one at a time while checking whether these additional constraints are violated and attempting to fix them. Broken non-linearities are fixed via a branching process, where the ReLU is split into two sub problems corresponding to the cases where $x_{i,j} \leq 0, \text{ReLU}(x_{i,j}) = 0$ and $x_{i,j} > 0, \text{ReLU}(x_{i,j}) = x_{i,j}$. This approach is considerably more efficient than a naive implementation, which scales exponentially with the number of nodes, i.e, a 300 node network would require 2^{300} checks. Marabou's implementation of ReLUPlex introduces further optimisations, improving the algorithm's scalability and the efficiency even further [229].

In contrast, α, β-CROWN [222, 234] uses a bounding-based verification algorithm [32], calculating relaxed linear bounds to approximate the behaviour of activation functions within the network. These bounds are then propagated through the network using forward propagation of the initial input bounds. This involves using linear relaxations for non-linear activations, like ReLUs, employing piecewise-linear caps to maintain computational feasibility. Although not exact, α, β-CROWN calculates sound and complete bounds on neural network outputs given input constraints, whilst scaling better than Marabou according to verification competitions. However, α, β-CROWN is more limited that Marabou, and the former can only verify a subset of the properties that the latter can.

Vehicle is a front-end domain specific language for Marabou, allowing these constraints to be expressed concisely and in a human-readable format. The Vehicle language helps users by converting potential complex verification criteria into a comprehensible format without requiring extensive expertise in formal methods. Vehicle helps map problem-space properties into feature-space or embedding-space equivalents, which is particularly useful for the NIDS due to the large gap between the problem-space and the feature space. The Vehicle language consists of a dependently-typed,

functional λ-calculus with operations for manipulating logical, numerical and vector expressions, which the user can use to express the desired behaviour of a network. We defer a more detailed explanation of the Vehicle language as well as some illustrative examples to Chapters 6 & 7.

To our knowledge, outside of this thesis, very little prior work exists applying neural network verification techniques to the security domain despite the seemingly natural fit. This is presumably due to the difficulty of representing security requirements as neural network constraints. Chen et al. [43] verify several constraints for security classifiers. However, these specifications simply act as common-sense checks on model output, for instance, verifying that models maintain high classification accuracy when less important features are augmented. In contrast, our approach in Chapter 6 bridges the problem-space/feature-space divide, allowing us to represent domain-specific security properties more concretely. Alternatively, robustness certification, a technique that provides probabilistic robustness assurances by sampling a model repeatedly at inference time, has been applied to NIDS [221]. However, we show the limitations of this approach in Chapter 6.4.5.

Chapter 3

Bad Data Design Smells & their Impact on Downstream Research

***Thesis Context**: This chapter introduces many of the issues with benchmark NIDS datasets that will be highlighted throughout this thesis. These bad data design smells degrade trust in the experimental results that depend on these datasets. Faulty assumptions about data quality can complicate research practises and introduce bias.*

3.1 Introduction

Benchmark datasets are vitally important in machine learning research. Datasets such as MNIST [123] and CIFAR-10 [120] allow researchers to compare methodologies on a fixed playing field, helping drive forward innovation. Unfortunately, datasets are rarely perfect real-world representations. It is well known that statistical properties of datasets may be considerably simplified when compared to that of real-world data. The ML pipeline is delicate; improper data may introduce experimental bias, weakening research findings. Even established benchmark datasets can contain defects, such as mislabeling in CIFAR-10 [153], arbitrary class distinctions in ImageNet [25] or run-to-failure bias in the Yahoo S5 dataset [231]. In the absence of high-quality datasets and critical examination, experimental bias may be endemic to an entire research field. To maximise the utility of datasets, identifying mistaken assumptions and eliminating their downstream effects are vital.

Our work shows that widely-used NIDS datasets could suffer similar issues when used for benchmarking ML methods. Unlike fields such as image or voice recognition, popular NIDS datasets can consist of synthetically generated data. Often, this data is generated via a series of scripted interactions across a testbed of virtual machines

which are recorded and converted into summary network flow statistics. The research community should be grateful for these datasets; publicly available network data is precious and these issues do not make these datasets unsuitable for all purposes. However, properly understanding their suitability for a given task is critical.

We are not the first to highlight this: research discussing issues with NIDS datasets is a well-established topic [16, 38, 39, 107, 114, 201] alongside critiques of specific datasets [73, 133]. In contrast to prior work, we aim to assess the potential impact of *data design* of NIDS datasets on downstream research. When creating an image dataset, many choices are ultimately arbitrary such as the contents or size of each class — there is little reason for CIFAR-10 to contain frogs over, say, turtles. In image recognition, the feature space and problem space are also closely aligned. In contrast, NIDS datasets rely heavily on domain-specific choices: dataset providers must decide how the network is configured, what attack classes to include, how attacks are launched, what features to extract, how to imitate benign traffic, and so on, which are typically obfuscated in the feature space. These choices provide a contextual backdrop that significantly alters threat models, attacker behaviour and implicitly defines generalisation standards. We describe these choices, conscious or unconscious, as *data design*. Questionable data design choices are difficult to correct, potentially introducing serious bias.

In this work, we analyse seven well-cited NIDS datasets, each with varying levels of documentation. To abstract away from the available documentation in our analysis, we distilled six indicators of potential design violations. Analogous to the term *design smell* [23, 205] in software engineering — signals of questionable design practises — we observe *data design smells*. We find dubious practises in all datasets: attacks launched against closed ports, labels leaked via features and millions of near-duplicate flows, to name a few. Altogether, our work shows that smells are ubiquitous in modern NIDS datasets.

For all seven datasets we analyse, we identify six data design smells — **poor data diversity**, **highly dependent features**, **unclear ground truth**, **traffic collapse**, **artificial diversity** and **wrong labels**.

In Section 3.2, we introduce these seven datasets. Then, in Section 3.3, we examine 38 papers that use these datasets, seeded from top conferences and their citations. We investigate four papers in detail, recreating their work when necessary, and demonstrate how NIDS dataset design could undermine their results. We find that the performance of *CADE* [236], a state-of-the-art concept drift detector, could rely on dataset artefacts, presenting worse performance on fixed data. We also can outperform AJSMA [190], an adversarial perturbation method, with a trivial attack, suggesting that "smelly" data must be used with caution when benchmarking adversarial attacks on NIDS. These examples and two more are covered in Sections 3.3.1–3.3.4. Looking at the remaining papers more generally, we assess authors' assumptions about NIDS data design, either explicitly stated or implied via their methodologies. We find that questionable assumptions are common and that examination of raw network data is rare, possibly leading to experimental bias. We discuss this in Section 3.3.5. To summarise, the contributions of this chapter include:

- **Design**: Analogous to *code smells* in computer programming, we identify six indicators of potentially bad data design choices that we call *data design smells*.
- **General Impact**: We study 38 papers, seeded from top conferences, that rely on these datasets, summarising their questionable assumptions. We find that using benchmark NIDS datasets 'off-the-shelf' appears to be general in NIDS research despite the prevalence of bad smells, potentially impacting conclusions.
- **Specific Impact**: Alongside the above, we also investigate four papers in detail, recreating their methodologies when necessary, and demonstrate how bad data design smells impact their experimental results. Again, we find that bad data design smells can insidiously complicate research if not properly accounted for.

This chapter, as well the following chapter, consists primarily of work published in "Bad Design Smells in Benchmark NIDS Datasets" at EuroS&P 2024 [81]. This work unanimously won the Distinguished Paper award at the conference. All work presented is mine, with the exception of the CADE case study and the paper overview, which was performed in concert with Gints Engelen.

3.2 Background

Releasing real-world data has severe privacy drawbacks and establishing the ground truth of real-world traffic is notoriously difficult [34]. Thus, synthetic datasets are commonly used in IDS research, generated using data collection testbeds.

We examine NIDS datasets that consist of two parts: the original traffic, in PCAP format, and a set of preprocessed statistics summarising each flow. This is a limitation of our approach as we require traffic captures that we can manually audit and assume there is an accompanying feature set for our automated analysis. Despite this commonality between our chosen datasets, there are fundamental differences that complicate comparisons. For instance, despite attempts to standardise feature sets [14, 177], researchers often use the bespoke flow statistics that accompany a dataset.

We list the datasets that we examine in Table 3.1 and refer readers to Section 2.3 for more detailed information about their composition. We omit the popular datasets KDD Cup [3] and NSL-KDD [210], as these are both derivatives of DARPA 1999 [131], which has long known to be faulty [144]. In our experiments, we evaluate the condensed version of Bot-IoT as it is more commonly used in the papers discussed in Section 3.3. For CTU-13, we use the normal traffic for our comparative measures in Section 4.3, as this was used as the benign traffic in the original accompanying paper. Due to the highly granular nature of the labels, we combine similar labels to form our classes, providing more detail in the Appendix.

There are some modified versions of these datasets [73, 133, 177], which either fix some labelling issues or alter feature sets, however, we examine the underlying dataset design, which cannot be changed by modifying feature sets. We use the original versions — unless where otherwise stated — as we aim to evaluate existing research, which predominately uses these original datasets.

[1] Number of citations according to Google Scholar, 21/03/2024

[2] We split ISCX's attack class into 5 based on destination port information, corresponding with each unique stage. Splitting the attack traffic is common in the wider research [63, 75].

Table 3.1: Dataset Summary

Dataset	Year	Class	Feat.	Hosts	Cit.[1]
CIC IDS 2017	2017	14	80	14	3264
CIC IDS 2018	2018	16	80	500	3264
ICSX 2012	2012	2/5[2]	20	25	1365
UNSW-NB15	2015	10	49	45	2817
Ton_IoT	2019	10	44	12	254
Bot-IoT	2021	5	45	10	1217
CTU-13	2014	13	15	-	866

3.3 Bad Smells and their Downstream Impact

As ML-based NIDS in research often use flow statistics rather than raw network data, the underlying traffic is obfuscated, such as the services within or low-level choices about the attacks. The papers accompanying these datasets sometimes provide limited descriptions of the generation process [29, 151, 186] and there is no comprehensive account of what specific traffic is in these datasets. As a result, a naive security researcher could be unaware of what they are detecting beyond high-level labels, such as 'Exploits'. Although researchers could produce such an account themselves, in Section 3.3.5, we argue that assuming that datasets can be used 'off-the-shelf' with limited analysis has become the default in the research community. Thus, there has been little auditing to uncover potential complications in these datasets. We aim to bridge this gap in knowledge.

To evaluate the usage of these datasets in research, we systematically review a subset of well-regarded papers that rely on these datasets. For our selection criteria, we began with works published between 2015 and 2023, inclusive, at the seven top-ranked non-cryptography computer security conferences — according to [241] — which cite at least one of these datasets. As we could not find many papers citing ISCX 2012, Ton-IoT, Bot-IoT or CTU-13 via this list, we expanded our criteria to include a greater number of security conferences, including CNS, RAID and DIMVA, as well as networking and data mining conferences, including KDD, WWW, CIKM and InfoCOM. Thus, we source papers from USENIX Security, S&P, EuroS&P, CCS, AsiaCCS, CNS,

RAID, DIMVA, KDD, InfoCOM, WWW, SAC, ACSAC and CIKM. We collate a list of 38 papers via this process, excluding systematisation of knowledge papers and papers whose main aim is to point out issues in other areas of NIDS dataset usage (for more details on our paper selection criteria, see Appendix A).

We look more closely at four example papers — two directly from the above overview and two cited by papers in the overview — and demonstrate how questionable data design may have impacted their results. In doing do, we observe that these complications stem from data patterns which we explicitly highlight. These observations lead directly to our bad data design smells, which we ***emphasise*** in the text. Altogether, our aim is to demonstrate that these datasets are being used at top-level conferences with little auditing or examination of the underlying data, whilst referencing similarly unsuspecting work.

We stress that we choose the phrase 'data design smell' because, just like bad smells in software design, they are merely *indicators* of *potentially* bad practises, and using "smelly" data does not immediately invalidate research results. In the following examples, we do not claim to negate the methodologies of the examined work. Instead, we wish to demonstrate how assumptions about NIDS data design can produce misleading conclusions.

3.3.1 Example 1 - LUCID

Original Paper LUCID [69] is a highly-cited, state of the art DDoS classifier, evaluated using the DDoS traffic in *ISCX 2012*, *CIC 17* and *CIC 18*. The authors' code has been made open-source [68].

At *LUCID*'s core is a traffic preprocessing algorithm. Ten features are extracted from the first n packets of a flow, zero padding when necessary, combining packet-level and flow-level information. This produces a 2-dimensional data structure of size $10 \times n$ which feeds into a Convolutional Neural Network to discriminate DoS flows from benign.

Data Design Upon examining *CIC 17*, we find that a single webpage is attacked across all DoS classes, namely, the default Apache page. As a result, the packet size features are extremely narrowly distributed, with 97% of total backwards packet size features approximately equal to $11595 \pm 1\%$ bytes (discounting flow calculation artefacts [73]). Moreover, flows with this value do not appear in the benign data.

Experiment Fixing $n = 10$, we repeat LUCID's feature extraction process on *CIC 17*. The authors consider a number of values for n and also truncate flows according to timing parameters. We found these modifications negligible and use the default values from [68]. We compare this to a simpler feature extraction process as a baseline experiment. Whilst we still consider just the first n packets of a flow, we extract only 3 features: total TCP size, total packet size and flow duration. Note that we've discarded the granular packet-level information, resulting in a massive reduction of LUCID's 100 features. We use a random forest as our classifier.

Table 3.2: Results of LUCID, Baseline and Baseline (Corrected) on *CIC 17*

Classifier	ACC	F1	TPR	TNR
LUCID	0.997	0.997	0.9988	0.9953
BL	0.997	0.997	0.9985	0.996
BL (C)	1.000	1.000	1.000	1.000

Results & Analysis Table 3.2 shows that we achieve comparable results to LUCID, despite using a much smaller feature set. For both LUCID and our baseline model, most misclassified flows were failed TCP handshakes. Filtering these flows, we produce a *corrected* version of the DoS dataset used by *LUCID*. On this dataset, a random forest achieves perfect accuracy and recall (BL (C)).

The design of LUCID implicitly assumes that predictive power can be gained by combining packet-level and flow-level information. However, given the severe lack of variation in *CIC 17*, this is not true. The paucity of variation in *CIC 17*'s DoS traffic stems from two data design choices made by the dataset authors: only launching DoS attacks against a single webpage and using fixed network conditions. Our analysis of *ICSX 2012* and *CIC 18* suggests that similar results would hold for those datasets.

Note that we cannot pass judgement on the effectiveness of LUCID in other, more realistic settings, where a more complex architecture might be justified. However, due to the design of the chosen test datasets, *LUCID's complexity is not justified by the experiments performed by Doriguzzi et al.*

Bad Smell 1 Many NIDS datasets contain data generated via automated tooling with fixed configurations or limited exploration of an attack's capabilities. This homogeneity causes ***poor data diversity***, inadequately testing a model's generalisation capabilities and rewarding overfitting.

3.3.2 Example 2 - AJSMA

Original Paper Considering adversarial attacks in constrained domains, Sheatsley et al. [192] present the Augmented Jacobian Saliency Map Attack (AJSMA), a white-box attack evaluated on *NSL-KDD* and *UNSW NB15*. The motivating insight of AJSMA is that, in intrusion detection, the problem space and feature space are distinct and arbitrary transformations may result in invalid data. Thus, when perturbing features, attacks must adhere to constraints. The ability of AJSMA to generalise across models is tested using five neural networks (trained using a stratified shuffle-split and labelled $M_A - M_E$) as well as other ML models, including Decision Trees (DT).

Data Design As a result of *UNSW NB15*'s testbed, a subset of features are highly performant across multiple attack categories, despite being apparently unrelated to the attacks' underlying mechanisms. In particular, the *Protocol* and *TTL* features overlap minimally between the benign and malicious classes and it is possible to separate these classes with 98% accuracy using these features alone. We provide more insight into why this is the case in Section 4.3.1

Experiment As a baseline comparison to AJSMA on *UNSW NB15*, we consider a simple feature perturbation attack: by modifying features, we 'convert' all attack flows to UDP (by altering the 'Protocol' and 'RTT' features), and TTL values to match those of benign traffic. These modifications are considered valid under the constraints that AJSMA adheres to; the original paper assumes that sound attack traffic can be created by converting TCP traffic to UDP and vice versa, provided constraint satisfaction. Unlike AJSMA, we do not assume access to the model's gradients or parameters. Because of this, we cannot replicate Sheatsley et al.'s evaluation process exactly. However, we do consider the performance of our attack across multiple models.

Table 3.3: Results of AJSMA and our Heuristic Attack on *UNSW NB15*. We consider the average accuracy across the tests presented by Sheatsley et al., using the notation $M_i \rightarrow M_j$ to denote an attack on model M_j using the gradients of M_i where $i,\ j \in A, B, C, D, E$ and $i \neq j$.

Attack	$(M_i \rightarrow) M_i$	$(M_i \rightarrow) M_j$	$(M_i \rightarrow)$ DT
AJSMA	1.000	0.790	0.166
HA	1.000	1.000	1.000

Results & Analysis Table 3.3 shows our attack achieves identical performance to AJSMA on *UNSW NB15*. However, we note that AJSMA's performance degrades when generalising across models. As our attack does not rely on the gradients of a specific model, we maintain perfect adversarial accuracy across all models tested.

We note that the performance of our attack is not due to any inherent qualities of malicious traffic. Instead, the large disparity between the benign and attack traffic is the result of data design choices, in the form of protocol and TTL choice in *UNSW NB15*. As a result, whilst AJSMA may be a superior attack in the general setting of constrained adversarial examples, the presented accuracy on *UNSW NB15* is not a meaningful measure of the attack's effectiveness. Moreover, as producing perfect adversarial perturbations is trivial, *worthwhile comparisons between AJSMA and alternative attack methodologies using UNSW NB15 are impossible*.

Bad Smell 2 Poor design of simulation testbeds can result in features of outsized importance that are unrelated to the underlying mechanism of an attack. Such ***highly dependent features*** reduce the complexity of attack detection and lead to overly optimistic interpretations of classifier performance.

3.3.3 Example 3 - Domain Adaptation (ADA)

Original Paper Due to the high rate of concept drift in security tasks, such as intrusion detection, ensuring that a deployed classifier can generalise to unseen attack classes is important. Singla et al. [199] propose a methodology for training NIDS to a rarely seen attack class via adversarial domain adaptation, evaluated on *UNSW NB15*.

Singla et al. preprocess *UNSW NB15* into two datasets: a *source* dataset, containing benign traffic and eight attack classes, and a *target* dataset, containing benign traffic and a ninth attack class, not included in the source dataset. At training, only a small number of samples from the target dataset are used. We focus on the case where 100 samples are used as this situation is highlighted by Singla et al, who consider the *Exploits*, *Reconnaissance* and *Shellcode* classes as holdouts.

Singla et al.'s ADA architecture has two parts, a generator and discriminator model. The generator has two goals, taking samples from both the source and target datasets, converting them into a domain-invariant embedding. This embedding is fed into a softmax layer, which classifies a sample as malicious or benign. Simultaneously,

the discriminator identifies whether the embedding comes from the source or target dataset. The generator is trained such that this embedding fools the discriminator. Once finished, the generator functions as a NIDS, capable of high performance on the target dataset despite having access to only a small number of samples.

Data Design Analysing *UNSW NB15*, we found heavy overlap between many of the attack classes, as well as features that correlate highly with all classes. As a result, it is dubious whether the target dataset can meaningfully be considered distinct from the source dataset, an implicit assumption in Singla et al.'s training methodology. In particular, the three most common combinations of forward and backward packets counts for the *Exploits* class also make up 40% of the *Shellcode* and *Reconnaissance* classes, and many of these flows appear to be notionally identical. This overlap between attack classes also leads to an overlap in highly discriminative features. Having highly similar attacks across disconnected attack categories is an implicit data design choice that, if unaccounted for, leads to test set leakage in experiments similar to Singla et al.'s.

Experiment We recreate Singla et al.'s set-up, reproducing their results. We then repeat the experiment whilst removing entries from the source dataset that are found in multiple classes, identified via the source and destination packets features.

We also remove features that are unjustifiably performant on the malicious data, such as *sttl*, *dttl* and *synack* (specifically, we remove the 'TTL' and 'RTT' features with HDF_C values higher than 0.7, detailed in Section 4.2.2). This process results in a large number of samples being removed from both the *Reconnaissance* and *Shellcode* classes, preventing us from repeating Singla et al.'s experiments using those classes as holdout classes. Thus, we only consider the case where the target dataset contains samples from the *Exploits* class.

Table 3.4: Best Reported Accuracy of ADA model vs Base case model for 100 target training samples.

Classifier	Original	Modified
Base (60 epochs)	0.82	0.82
Base (1000 epochs)	0.8484	0.8442
ADA (10000 iterations)	0.8804	0.8350

Results & Analysis On the unmodified version of *UNSW NB15*, we reproduce similar results to those presented by Singla et al., achieving a 6% gap between the base case and ADA models[3]. However, when we remove the problematic attack samples and classes, this advantage drops to a 1.5% performance gap. When removing malicious traffic, we downsample the benign traffic to maintain the same benign/malicious ratio as before. We also note that the base model neural network can exceed this score by extending its training regime.

From our experiment, Singla et al.'s results are biased by several data design issues, namely, unclear attack classes with incomplete attack capture, which lead to poorly defined boundaries between attack classes. We note that the plurality of this overlap stems from attacks in UNSW with no apparent effect, and it is unclear how legitimate this attack traffic is. We emphasise that the assumptions made by Singla et al. about *UNSW NB15* are completely reasonable; *Exploits*, *Reconnaissance* and *Shellcode* are distinct categories of attacks and there is little reason to assume that this conceptual blurring between classes would be present in the data. However, this demonstrates that, *without modification, UNSW NB15 is unsuitable for evaluating the ability of classifiers to generalise between attack categories.*

[3]Although Singla et al. report that they use a source training dataset with 83,961 samples, it's unclear what ratio of benign to malicious traffic they use. We achieve similar results using a source training dataset with 53,112 benign and 38,679 malicious samples.

Bad Smell 3 Datasets can lack clear labelling logic, often labelling background services as attacks for unspecified reasons. This ***unclear ground truth*** creates a disconnect between what researchers understand a class to contain and what it actually contains, limiting their ability to reason about their methodology and results.

3.3.4 Example 4 - CADE

Original Paper Yang et al. [236] also combat concept drift via contrastive learning with CADE. CADE leverages *contrastive learning* to detect drifting samples, including an evaluation on *CIC 18*.

In their experimental setup, the authors picked one day's worth of benign traffic in addition to malicious traffic from the *Infiltration*, *DoS Attacks - Hulk* and *SSH - Brute Force* classes. They then iteratively train their classifier on the benign traffic and two malicious classes; the third malicious class represents the 'unseen' class and is only used in the test set.

Data Design Analysing *CIC 18*, we note that attacks take place within short time frames. Whilst this is a legitimate data design choice, downstream researchers must be aware of this fact when evaluating their methodologies. Unfortunately, CADE does not remove the *Timestamp* feature during training and evaluation, leading to a contextually *highly dependent feature*.

Experiment We use a corrected version of *CIC 18* [133] after verifying the author's fixes to the labelling and feature extraction process, and reran the CADE experiments (with the original and fixed versions of the dataset), whilst removing the *Timestamp* feature.

Results & Analysis Table 3.5 shows our results. The performance is severely degraded for all but the original setup. These results reinforce **Bad Smell 2** and we note that mislabelled data cause serious experimental bias.

Bad Smell 4 Inaccurate ground truth of generation testbeds can lead to mislabelled data. The ***wrong label*** smell degrades the ML pipeline by altering classification complexity. Furthermore, researchers discovering disparate subsets of labelling issues can lead to inconsistent benchmarks, complicating direct comparison between techniques or architectures.

This example highlights how data design smells are **not** harmful in all contexts: we emphasise that including the *Timestamp* is not unreasonable if the data shows periodic behaviour, as in UGR'16 [141]. However, in *CIC 18* attacks reside within narrow time-windows [186], making *Timestamp* a *highly dependent feature* for this dataset, as it is both highly performant with no connection to the attack's underlying mechanism. Similarly, certain model architectures may rely on timestamp features to infer dependencies between flows, but CADE does not do this. Overall, we do not claim that training on timing features is bad in general. Instead, researchers must understand whether some features are appropriate for a given detection method, aided by documentation from dataset creators.

Table 3.5: F1 results for CADE on the original and fixed *CIC 18* dataset, with and without the *Timestamp* feature.

	With Timestamp		Without Timestamp	
	Original	Fixed	Original	Fixed
SSH - BF	0.8687	0.4968	0.1214	0.0
DoS Hulk	0.9997	0.9988	0.7614	0.9987
Infiltration	0.9999	0.9929	0.0537	0.9964

3.3.5 Potential Data Bias in Research

We now look at the papers more generally. We assess implicit assumptions made across four criteria, reflecting our smells thus far. First, we consider assumptions about *data diversity*, which we subdivide into attack variation (AV) — the number of distinct interactions in a class — and feature variation (FV) — the variability of features in a class. We then assess whether papers include critical, post-hoc analysis of feature importance, connecting features to mechanism of an attack. In the absence of such analysis, we say the paper assumes that the data was free from *highly dependent features* (HDF). Finally, we consider whether papers assume the data was free from *wrong labels* or *unclear ground truth*, which we combine into a single criteria (W/U). This process was undertaken by a single author and then repeated by a second author on a random subset of 25% of papers, who then cross-referenced their findings to ensure agreement. The full paper analysis methodology as well as paper selection criteria are detailed in Appendix A.

Results & Analysis We list our results in Table 3.7, marking assumptions as 'unclear' when unable to fairly judge. As papers made few comments about the data, we judged assumptions implicitly via their methodologies.

Many papers applied techniques that seem unjustifiably complex given the low attack variation in classes, such as training individual models for each attack [219] or using a complex setup such as a LSTM variational autoencoder [225], whilst a minority aim to generalise between attack classes [96], intentionally injecting variety. Without a critical examination of feature importance, we believe that papers in our overview overwhelmingly assumed the datasets were free of *highly dependent features*. Such analysis was rare, with [91, 107, 122, 183] being notable. Some papers used techniques that assumed greater feature variety than is present, such as oversampling via SMOTE [17]. Due to low variability, we demonstrate how naive application of

techniques such as SMOTE [40] fails to introduce feature variety in Table 3.6 and how packet-level features succumb to low variation in Section 3.3.1. Papers also made statements about the properties of NIDS data *generally*, without investigating whether specific dataset design characteristics were responsible for their results [75, 193].

With a few exceptions [107, 122, 238], we found no evidence that papers audited any raw PCAP data. As a result, almost all papers used mislabelled or unclear data, highlighted in Section 4.3.1. Based on their research aims, mislabelled data was irrelevant for some papers [42, 104] (as such, these papers were left out of our scope), and a minority used a corrected version of *CIC 17* [183] and *CIC 18* [122]. Although these corrected dataset were released after many of these papers, cursory manual analysis would have also uncovered mislabelling issues.

Often, misconfigured testbeds result in failed attacks e.g., attacks launched against closed ports. We understand that detecting these connections is a reasonable goal for an IDS. However, due to the homogeneity of the traffic, these can be trivial to detect in a machine learning setting, as standard, randomised train/test splits result in data leakage. No papers in our overview commented on this or amended their evaluation process, and all were seemingly unaware of these discrepancies. We believe it is exceedingly likely that papers would re-evaluate their proposed model architectures/pipelines if aware of the simplicity of the classification task. This suggests a new bad smell.

Bad Smell 5 Simple configuration mistakes can extinguish data diversity from a class. We call this problem ***traffic collapse***, as statistics 'collapse' into a trivial distribution, preventing models from learning any meaningful information from features.

In *CIC 17* and *Bot IoT*, because of spurious network conditions, some attack traffic is questionable, such as malformed connections or extraordinarily high retransmission rates. We believe these out-of-distribution flows are pernicious, preventing classifiers from achieving **perfect** accuracy, instead presenting **near-perfect** accuracy. The former implies that classification may be trivial whilst the latter does not, justifying the use of complex ML architectures. Only very few papers appeared to examine misclassifications to some extent [117, 225, 238]. This introduces our final bad smell.

Bad Smell 6 When generating network traffic, a number of difficult-to-control variables — poor network conditions, network capture failures, retransmission rates etc. — impact the structure of flows. If not properly managed, these variables create ***artificial diversity***, causing researchers to overestimate classification complexity.

Table 3.6: *Total Length of Fwd Packet* is highly discriminitive in *CIC 17*. For 72.7% of *Heartbleed* flows, this feature equals 7920. Furthermore, all variability in this class is caused by *artifical diversity*. Correcting the original data (C) exaggerates this effect.

	Org.	Org. (C)	SMOTE	SMOTE (C)
TLFP	72.7%	100%	71.4%	100%

Table 3.7: Paper Assumptions. ✓: assumption present, ✓*: assumption partially present, ✘: assumption not present/relevant, -: unclear. I: *ISCX 2012*, C: *CIC 2017*, C2: *CIC 2018*, U: *UNSW NB15*, CT: *CTU-13*, B: *Bot IoT*, T: *Ton_IoT*.

		Assumptions			
Paper	Dataset	FV	AV	HDF	W/U
[75]	I,U	✓	✓	✓	✓
[239]	I,CT	✓	✓	✓	✓
[159]	U	✓	✓	✓	✓
[199]	U	✓	✓	✓	✓
[7]	I,CT	✓	✓*	✘	✓
[125]	C	✓	✓	✓	✓
[233]	CT	✓	✓	✓	✓
[19]	C	✓	✓	✓*	✓
[126]	C	✓	✓	✓	✓
[209]	C	✓	✓	✓	✓
[238]	U	✓	✓*	✓	✓
[96]	C	✓	✓	✓	✓
[236]	C2	✓	✓	✓*	✓

		Assumptions			
Paper	Dataset	FV	AV	HDF	W/U
[26]	C2,U	✓	✓	✓	✓
[4]	I,C,C2	✓	✓	✘	✓
[99]	I,C	✓	✓	✓	✓
[225]	U	✓	✓*	✓	✓
[224]	C	✓	✓	✓	✓*
[107]	C	✘	✘	✘	✓
[128]	C,C2	✓	✓	✓	✓
[27]	U	✓	✓	✓	✓
[18]	B,T	✓	✓	✓	✓
[185]	I,C2,CT	✓	✓	✓	✓
[232]	U	✓	✓	✓	✓
[17]	T	✓	✓	✓	✓
[221]	C2	✓	✓	✓	✓

		Assumptions			
Paper	Dataset	FV	AV	HDF	W/U
[98]	C2	✓	✓	✓	✓
[122]	C	✘	✘	✘	✘
[117]	U	✓	✓	✓	✓*
[64]	C	✓	✓	✓	✓
[91]	U	✓*	✓*	✘	✓
[219]	CT,T	✓	✓	✓	✓
[183]	C	-	-	✘	✓*
[193]	U	✓	✓	✓	✓
[67]	B	-	-	-	-
[53]	C,C2	✓*	✓	✓	✓
[160]	U	✓	✓	✓	✓
[28]	I,C	✘	✘	✘	✓

3.4 Conclusion

In this chapter, we present the first part of our analysis into benchmark NIDS datasets, focusing on bad data design smells and their downstream impact on research. In Chapter 4, we expand this criticism by describing our manual analysis process for uncovering bad smells, detailing specific dataset issues we discovered, developing a series of automated heuristics for detecting bad smells, comparing bad smell prevalence in other domains and providing a set of recommendations for using smelly NIDS data.

Our results worryingly suggest that almost all papers we investigate assume that these datasets can be used without amendments or corrections. Despite systematising our literature overview, it is difficult to ascertain whether this assumption actually degrades research conclusions. However, at a minimum, we believe that our case studies present some circumstantial evidence that these issues are widespread. When selecting case studies to investigate further, we chose papers that we felt were representative of the wider literature. Instead, these were simply papers that were high-quality — either highly-cited, published in top venues or a key reference for later high-quality research — that we found early in our wider investigation. In fact, the four case studies presented in this chapter were the only papers we investigated in more detail. Although too small a sample size to be rigorous, the fact that our hunches were correct with such high accuracy suggests that these assumptions can uncover experimental pitfalls.

Chapter 4

Finding and Correcting Bad Smells

***Thesis Context**: This chapter details many specific issues with benchmark NIDS datasets. By quantifying their prevalence, we show the extent of the issues. Finally, we provide some recommendations to mitigate these bad data design smells and strengthen trust in these datasets.*

4.1 Introduction

In the previous chapter, we introduced a set of suspect design choices in benchmark NIDS datasets — bad data design smells. By relating dataset design choices to experimental choices, we observed a stark gap between dataset contents and the downstream research relying on these datasets. In particular, researchers typically made assumptions regarding the soundness of these datasets and appeared to proceed with their experiments without checking for artefact, mislabelling or complexity issues, all of which could bias their results. Although we highlighted the impact of these bad design smells on four case studies, we have only assumed that these smells are broadly applicable to the full set of examined datasets. In this chapter, to rectify this and demonstrate that these smells are general across our examined datasets, we undertake a thorough manual and automated analysis.

This chapter is structured as follows: we discuss the design of our manual analysis in Section 4.2.1 and we comprehensively analyse 65+ attacks. In Section 4.2.2, we develop six heuristic measures to assess the severity of these issues automatically, which can be used on future datasets. Both stages of our analysis uncover suspect design choices in all datasets, covered in Section 4.3. We catalogue these so researchers using these datasets can avoid experimental pitfalls. For instance, we show that it is often trivial to get perfect classification accuracy using a single feature

that is unrelated to the underlying mechanism of an attack. Moreover, our automated heuristics score far better on general tabular anomaly datasets [166], suggesting that the impact of these design choices are particular to NIDS datasets. In Section 4.4, we discuss some potential recommendations to account for NIDS dataset design choices and to improve the standard of intrusion detection research. We conclude with related work in Section 4.5, limitations in Section 4.6 and discussion in Section 4.7.

Our results enable a deeper understanding of the implications that questionable data design may have on downstream NIDS research. To summarise, the contributions of this chapter include:

- **Dataset Analysis**: We devise a novel, comprehensive methodology for analysing network data design, identifying potential research pitfalls stemming from data design. We apply this to seven popular synthetic NIDS datasets and 65+ individual attack classes. We find that dubious data design practises are ubiquitous across popular NIDS datasets, necessitating their careful treatment as benchmarks.

- **Prevalence**: For each smell, we distil a heuristic measure to evaluate its severity. These heuristics are designed to be lightweight, allowing us to measure 65+ attack classes. Our results show that negligible data diversity, severe mislabelling and trivial classification complexity are common in NIDS datasets.

- **Recommendations**: We propose guidelines for using/developing NIDS datasets to minimise the impact of these design smells, providing insights into how to improve data usage such that we as a community ensure higher quality intrusion detection research.

4.2 Finding Bad Smells

In this section, we introduce a methodology for examining NIDS datasets. This analysis has two stages: a manual stage — a qualitative evaluation of the problems with these datasets — and an automated stage — a quantification of bad smell prevalence and severity via heuristic measures. To assess the rate of false positives, we design our methodology without reference to *CTU-13* and use it as a test case.

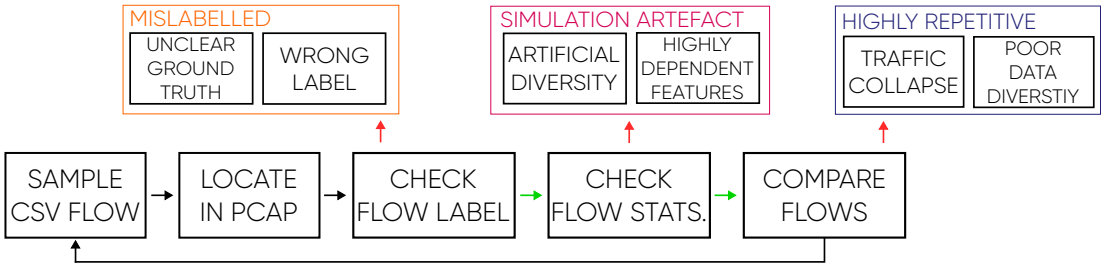


Figure 4.1: Overview of Manual Analysis Process. To facilitate easier discussion, note that we group our bad smells into three categories: ***Mislabelled***, ***Simulation Artefact*** and ***Highly Repetitive***.

Although we wish to compare the data produced by these generation testbeds to some real-world dataset, the research community's reliance on synthetic data is precisely due to the difficulties in obtaining data drawn from real-world operational environments; as such, we do not have an exemplar dataset to utilise during this process. Instead, it is necessary for our methodology to be largely self-contained, evaluating the flaws of a dataset with minimal reference to other sources of data.

For a synthetic dataset to function as a baseline dataset, the challenge of classifying intrusions or anomalies must be comparable to detecting these abnormalities in a real-world setting. To evaluate this claim, we rely on several ML-models trained to classify malicious data from benign data. Despite bearing superficial resemblance, these models are **not** NIDSs. Instead, they merely form part of our data design evaluation process.

4.2.1 Manual Analysis

We aim to document all flows within each attack class. Complete coverage is vital; it is likely that researchers using these datasets will find a subset of problems and remove the offending traffic. Thus, researchers are not comparing their results on a fixed dataset but rather on several disparate datasets, each corrected in a unique manner. Standardising this process requires a full examination of the underlying PCAP data.

Examining each flow is onerous and time-consuming. Instead, we assume that we can identify unique attacker behaviour via unique values of the *Total Source Bytes* feature, a standard inclusion in the accompanying feature sets. Similarly, we assume that unique values for the *Destination Port* and *Total Destination Bytes* features correspond with unique victim behaviours. We cluster flows that attain the same values on these features, up to small variations, reducing the number of flows to be analysed from tens of thousands to a small number of clusters based on CSV data alone.

For each of these clusters, we randomly select an exemplar flow and locate it in the PCAP data via Source and Destination IPs/Ports and Timestamp information. Using these exemplar flows, we examine each cluster in parallel. For each flow cluster, we aim to understand the generation process that give each cluster its characteristic properties. These include understanding the attack, how the attack is realised, the target service, the labelling logic and the intra- and inter-cluster relationships between flows. We survey each flow via a series of yes/no questions, each related to a bad smell from Section 3.3.

Q1: Wrong Label – Does the flow's label accurately describe its behaviour? Relying on the provided documentation and contextual clues, we assess whether its label is correct. Labels have varying degrees of granularity, including specific attacks, attack classes and high-level descriptions — e.g., *Heartbleed*, *Reconnaissance* and *Attack*. We rely MITRE's *CVE*s [145] and *CWE*s [146] for attack definitions. For vaguer labels, we rely on personal assessment. If we can't associate a flow cluster to its label, the *wrong label* smell is present.

Q2: Unclear Ground Truth – Does the flow originate from the attacker network and/or is directed towards the victim network? With the exception of 'insider' attacks, we expect attacks to occur between the attacker and victim networks, or within the attacker network for, say, C&C traffic. Failing this indicates that *unclear* traffic has been labelled alongside the attack. We check whether these flows are associated with any background processes to confirm this.

Q3: Highly Dependent Features – Do distinct clusters share similar properties? Although we aim to capture similar flows in our clusters, it is problematic when attacker behaviour (determined in **Q1**) is identical across clusters (whilst differing from the benign data). These properties may be reflected in the chosen feature set, biasing models via unrelated features. If so, we consider the *highly dependent feature* smell to be present.

Q4: Artificial Diversity – Is the primary difference between clusters due to network artefacts? Unrealistic network conditions may lead to similar flows (which we determine via **Q1**) being distributed across several clusters. We consider failed handshakes, aborted flows, frequent network anomalies — e.g, dropped packets or retransmissions — and differences in flow termination as network artefacts. If clusters differ due to these phenomena, the *artificial diversity* smell is present.

Q5: Poor Data Diversity – Are clusters large, relative to the size of the class? Based on our assumptions, large cluster sizes indicate that a class mostly consists of both the attacker and victim engaging in the same behaviour repeatedly. If a cluster (and any related clusters identified in **Q4**) contains over 25% of all flows, we assume there is *poor data diversity*. When possible, we confirm the source of this lack of variety — such as the reliance on automated tooling — via the details gathered during **Q1** and **Q3**. We note that it is reasonable to expect certain volumetric attacks, such as ACK floods, to have low data diversity. However, detecting such attacks via ML still requires careful evaluation, due to the risk of overfitting to arbitrary features or leaking test set data.

Q6: Traffic Collapse – Has the attack been fully executed? When an attack is not fully realised, due to, say, a secure service, the response from the victim is typically limited across the entire interaction. We examine flows for evidence of host responses. Unexpected behaviours include backwards flows containing only RST packets and flows with no response. Here, we say the *traffic collapse* smell is present.

A high-level summary of this process is in Figure 4.1. If a question's answer is unclear based on a single flow, we sample new flows until we can satisfactorily answer, indicated by the backwards arrow in Figure 4.1.

4.2.2 Automated Prevalence Analysis

Although a qualitative analysis is necessary, it is arduous and time-consuming. An automated process that can highlight problems quickly when generating a dataset would be highly beneficial. However, PCAP data is a complicated format. As the properties of each dataset are highly variable, it is difficult to verify all attacks across all network conditions. Instead, we design some heuristics for CSV data, with minimal reference to the original PCAPs.

We preprocess all features according to standard practises: we use a min-max scaler and convert categorical features to ordinal or one-hot encoded features. Our aim was to mimic how these datasets may be used by a researcher who hasn't checked the underlying data.

Mislabelled We perform two tests for *mislabelled* traffic, corresponding to the *unclear ground truth* and *wrong label* smells respectively. Naive labelling algorithms based on IPs have a tendency to mislabel background traffic that are orthogonal to the mechanism of an attack as malicious, such as authentication traffic, discovery services, and advertising features. Thus, labeling decisions can become ambiguous or unclear. To address this, we maintain a list of ports related to well-known background services and flag a flow as *unclear* if its destination port feature, denoted as $F_{\text{Dst Port}}$, belongs to this set of ports (indicated by BG Ports[4]). Note that we base our ports on the datasets examined; services operating on these ports can still be abused by attackers, and the specific ports chosen may not be suitable for other datasets, leading to false positives. We quantify potentially *unclear* flows by calculating the ratio of flows sent to these ports to the total number of flows in the dataset. Thus, for given class C:

$$UGT_C = \frac{|F_{\text{Dst Port}} \in \text{BG Ports}|}{|C|} \quad UGT_C \in [0,1] \tag{4.1}$$

[4] BG Ports $= \{0, 53, 67, 68, 111, 123, 137, 161, 179, 389, 427, 520, 1723, 1900\}$

To estimate the number of flows with the *wrong label* smell, we use the Edited Nearest Neighbour Rule (ENN), proposed by Wilson et al. [163, 228]. ENN identifies a sample as close to a decision boundary if its label differs from the majority of its k-Nearest Neighbours. We modify the original ENN process by breaking ties in favour of the mislabelled class.

Setting $k = 4$, we define the majority class identified by ENN as $ENN(x)$ where $x \in C$. We then measure the degree of mislabelling via the percentage of elements of C misclassified by ENN, or:

$$WL_C = \frac{|\hat{C}|}{|C|} \text{ where } x \in \hat{C} \textit{ iff } \text{ENN}(x) \neq C,\ WL_C \in [0,1] \tag{4.2}$$

Simulation Artefacts To detect *highly dependent features*, we use a maximal feature efficacy process. Looking at each attack class C separately, we measure the F1 score of a random forest classifier distinguishing C from the background when trained on a single feature, F_i. Intuitively, if F_i is highly dependent, we expect an unreasonably high F1 score. Although this process can detect multiple artefacts, for brevity, we report only the most severe instance. Thus, we define HDF_C as:

$$HDF_C = MAX(F1(F_i))_i \quad HDF_C \in [0,1] \tag{4.3}$$

During manual analysis, *artifical diversity* was mostly seen when unstable network conditions caused high numbers of dropped or re-transmitted packets. We check that problematic packets remained within the bounds set out by prior work [71]. As this tended to be general across the network capture, we only report this in Section 4.3 when notable for a dataset. This is the only heuristic which requires access to the original PCAPs.

Highly Repetitive Traffic Rather than classification complexity [102], we aim to measure data diversity independent of classification. Thus, our measures for the *traffic collapse* and *poor data diversity* smells use a two-stage clustered similarity process: first, we estimate the number of clusters, N, within class C via the Elbow method [212]. We then apply KMeans to assign each data point a cluster, C_i. Via cosine similarity, we measure similarity between randomly sampled pairs as:

$$CS_{C_i} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \quad \text{where } \mathbf{A} \sim C_i, \mathbf{B} \sim C_i, \\ CS_{C_i} \in [0,1] \tag{4.4}$$

We record the average CS_{C_i} between M[5] pairs from each cluster weighted by cluster size, expecting a score of approximately 1 for near-identical pairs. This provides quick insight into a class's cluster sizes as an approximate measure of data homogeneity, corresponding to our *poor data diversity* smell:

$$PDD_C = \sum_{i<N} \sum_{j<M} \frac{|C_i|}{|C|} \frac{CS_{C_i}}{MN} \qquad PDD_C \in [0,1] \tag{4.5}$$

For our *traffic collapse* bad smell, we wish to measure an egregious lack of data diversity, potentially caused by configuration issues. We repeat the above process, but measure the percentage of pairs from each cluster where CS_{C_i} exceeds a threshold value of 0.95[6], indicating that the flows are functionally identical. We report the maximum value across our clusters. Using Iverson brackets, we write this as:

$$TC_C = \max_i \left(\sum_{j<M} \frac{[CS_{C_i} > 0.95]}{M} \right) \qquad TC_C \in [0,1] \tag{4.6}$$

[5]We found scores to be converge consistently with $M \approx 100$

[6]We select this value as corresponds to an angle of approximately $\frac{\pi}{10}$ between sampled flows, or 10% of maximal dissimilarity

4.3 Results

4.3.1 Manual Analysis

We apply our manual analysis process to over 65 attack classes. In this section, we demonstrate the scope of the problems we uncovered with examples.

Mislabelled Mislabelling stems from design choices throughout the generation process. Researchers must be clear about what they are generating. However, this can be murky, such as in *UNSW NB15*, which used the predetermined 'strikes' of the IXIA PerfectStorm tool [115]. This causes *unclear ground truth*, such as the *Fuzzing* class containing dubious routing attacks. These have no associated CVE and simply alter minor aspects of the protocol, creating flows that are statistically identical to their benign equivalents. We also note that the definition for *UNSW NB15*'s *Generic* class, confusingly, involves block cipher vulnerabilities with no relationship to the dataset's contents [151].

Labelling flows via IPs and timestamps is challenging. Researchers must account for background traffic of the attacker network; naive logic may mislabel these flows. *TON_IoT* consistently treats background DNS traffic as malicious, despite being unrelated to the attack chain; in the *DoS* and *XSS* classes, 55% and 28% of functioning flows are DNS requests, respectively. Labelling multi-stage attacks is complex, leading to errors. In *CIC 2018*, the *Infiltration* class misses entire attack stages, incorrectly labelling them as benign. Mislabelling can also occur during final processing steps. Again, in *CIC 2018*'s *Infiltration* class, several benign flows were duplicated and included, exacerbating the previous issue. In contrast with other datasets, *CTU-13* provides highly granular labelling, making it easier to discard mislabelled flows. Despite this, some problems persist, such as OS updates labelled as malicious adware and flows that appear to have been accidentally filtered from the network capture.

Aside from black-and-white errors, NIDS datasets are plagued by murky labelling, exacerbated by poor documentation. Although better than other datasets, we found discrepancies between the documentation [196] for *ISCX 2012* and the PCAPs. For example, *HTTP DoS* is reportedly executed using *Slowloris*, which overwhelms a server with incomplete HTTP requests. However, we found no evidence of these partial requests; instead, the attack consisted of generic GET requests.

Simulation Artefacts Simulation artefacts can affect an entire dataset. Consider *UNSW NB15* whose features include the hosts' *time-to-live* values. This appears to inadvertently fingerprint operating systems and in, say, the *Exploits* class, the distribution of operating systems among attacked machines differs significantly from that of the machines receiving benign traffic. Consequently, this *highly dependent feature* simplifies classification. The ratio of TCP to UDP traffic between the benign/attack classes causes similar issues. Subtle choices, such as attackers targeting small webpages whilst benign users visit large webpages can bias features. In *CIC 17*, this results in highly discriminative *total packet length* features, even when unrelated to an attack's underlying behavior. Mistaken flow calculation can also cause artefacts: in *CTU-13 Scenario 4*, several hundred non-existent UDP flows with impossible characteristics are recorded due to mistaken processing. Alongside mislabelled NetBIOS traffic, these account for 99.9% of malicious *UDP Attempt* traffic.

Table 4.1: Results of Heuristic Measures (- indicates indeterminate due to small class size, * indicates clash with *FTP-BruteForce*, † indicates clash with *LOIC*, ‡ indicates clash with *Background*). We provide results for the primary class of each CTU-13 scenario, measured by packet volume, with more detail in the Appendix.

Dataset/Class	PDD_C	TC_C	WL_C	HDF_C	UGT_C
UNSW NB15					
Generic	**0.98**	0.92	0.0	**1.0**	**0.98**
Exploits	0.89	0.41	**0.10**	0.88	**0.37**
Fuzzers	0.93	0.6	**0.52**	0.66	**0.51**
DoS	0.91	0.48	**0.12**	0.90	**0.78**
Recon.	**0.95**	0.76	**0.38**	**0.95**	**0.84**
Analysis	0.93	0.54	**0.21**	0.89	**0.77**
Shellcode	**0.97**	**0.95**	**0.57**	0.69	0.0
Backdoor	0.91	0.48	0.0	0.91	**0.82**
Worms	0.94	0.46	**0.60**	0.78	0.0
ToN_IoT					
scanning	**0.97**	**0.95**	0.0	**0.99**	0.01
dos	**0.99**	**0.97**	0.0	**0.98**	0.03
ddos	**0.99**	**0.98**	0.0	**0.97**	**0.12**
mitm	0.83	0.85	**0.27**	0.73	**0.56**
xss	0.84	0.86	0.0	**0.97**	**0.27**
backdoor	**1.0**	**1.0**	**0.31**	**1.0**	0.0
injection	**0.95**	0.92	0.0	**0.98**	0.03
passwords	0.89	**1.0**	0.0	**0.99**	0.0
ransomware	0.83	0.91	0.05	0.84	0.0
Bot IoT					
DDoS	0.86	0.5	0.0	**0.98**	0.0
DoS	0.87	0.37	0.0	**0.99**	0.0
Recon.	**0.93**	0.53	0.01	**0.98**	0.0
Theft	0.89	0.67	0.06	**1.0**	0.0
CTU-13					
Neris 1	0.84	0.36	0.0	0.83	0.0
Neris 2	0.85	0.36	0.0	**0.96**	0.0
Rbot 1	**0.98**	**0.99**	**0.18**‡	**1.0**	0.0
Rbot 2	0.87	0.7	0.0	0.0	**0.98**
Virut 1	0.94	0.67	0.0	0.8	0.0
Donbot	**0.97**	**1.0**	0.0	**1.0**	0.0
Sogou	-	-	-	-	-
Murlo	**0.96**	0.84	**1.0**	0.87	0.0
Neris 3	0.9	**0.97**	0.0	0.87	0.0
Rbot 3	**0.99**	**0.99**	0.0	**1.0**	0.0
Rbot 4	**0.98**	**0.96**	0.01	**1.0**	0.0
NSIS	0.78	0.68	0.02	0.88	0.0
Virut 2	0.88	**0.95**	0.0	0.9	0.0

CIC-IDS 2017					
Portscan	**0.99**	**0.99**	0.0	**0.98**	0.0
DoS Hulk	**0.98**	**0.98**	0.0	**1.0**	0.0
FTP-Patator	**0.98**	**0.98**	0.0	**0.99**	0.0
SSH-Patator	**1.0**	**1.0**	0.0	**0.98**	0.0
DDoS	**0.98**	0.94	0.0	**0.99**	0.0
Bot	**0.98**	**1.0**	0.01	**1.0**	0.0
Slowloris	**0.97**	**1.0**	0.0	**0.98**	0.0
Slowhttptest	0.88	0.53	0.0	**0.96**	0.0
GoldenEye	**0.95**	0.68	0.0	**1.0**	0.0
Infil.	0.92	0.65	**0.81**	0.75	0.0
Brute Force	**0.99**	0.93	0.06	0.91	0.0
XSS	0.78	0.48	**0.35**	0.93	0.0
SQL	-	-	**0.63**	-	0.0
Heartbleed	-	-	**0.18**	-	0.0
CIC-IDS 2018					
Infil.	0.67	0.32	**0.65**	0.63	**0.32**
Bot	**0.99**	**0.99**	0.0	**0.99**	0.0
Hulk	**0.98**	**0.99**	0.0	**1.0**	0.0
Slowloris	0.83	0.89	0.0	**0.99**	0.0
SSH-Bruteforce	**0.99**	**0.99**	0.0	**1.0**	0.0
FTP-BruteForce	**0.99**	**1.0**	0.0	**0.99**	0.0
LOIC	**0.96**	**0.99**	0.0	**1.0**	0.0
LOIC-UDP	**0.96**	0.82	**0.16**†	**0.99**	0.0
HOIC	**0.98**	0.88	0.0	**1.0**	0.0
GoldenEye	0.93	**0.99**	0.0	**1.0**	0.0
SlowHTTPTest	**0.99**	**1.0**	**0.56***	**1.0**	0.0
XSS	0.90	**1.0**	0.05	0.83	0.0
Web	0.77	**1.0**	**0.21**	0.79	0.04
SQL	0.86	0.85	**0.21**	0.77	0.0
ICSX 2012					
BruteForce	**0.99**	**0.99**	0.02	**0.96**	0.0
SSH	**0.98**	**1.0**	0.0	0.93	0.0
nmap	**0.93**	**1.0**	0.04	0.78	0.02
IRC	**0.96**	**1.0**	0.0	0.70	0
Other	0.75	0.37	**0.35**	0.56	**0.13**

Highly Repetitive *Highly Repetitive* data undermines the ubiquitous train/validation/test pipeline. This prevents building meaningful holdout sets and, consequently, the generalisation abilities of ML classifiers, a primary research goal, are not examined. Our analysis reveals the extent of this issue, exemplified by *UNSW NB15*'s *Reconnaissance* class. The primary protocol in this class is Portmap at roughly 80.5% with very little variation between flows. Similarly, 97.8% of malicious flows in *CTU-13*'s *Scenario 5* are related to a basic nmap scan.

Classifying volumetric attacks is straightforward. Without exception, DoS attacks were launched against static targets. This design choice produce millions of near identical flows. This is particularly noteworthy in *Bot IoT*, as only 0.00013% of traffic is benign, and the overwhelming majority of attack traffic is volumetric. In Section 4.4.1, we demonstrate how lack of diversity rewards overfitting models without evaluating their generalisability.

When this smell coexists with other smells, it can be masked, potentially misleading researchers into overestimating the classification challenge presented by a dataset. For instance, in *CIC 17*, the minority classes *SQL Injection* and *Heartbleed* suffer from the *mislabelled* and *artificial diversity* smells respectively. Remediating these issues, it is possible to achieve perfect accuracy classifying these attacks even with simple models.

Often, data was not adequately audited. Launching attacks against closed ports is common, including the *Ton_IoT backdoor*, *Bot IoT Theft* and *CIC 18 FTP-Bruteforce* classes. Attacks were also launched against secure services, such as *CIC 17*'s *DoS Goldeneye* and *CIC 18*'s *XSS* classes. Although detecting failed attacks is potentially good, these should be explicitly labelled as failures. During our analysis in Section 3.3 we did not encounter a single paper that appeared to be aware that they were working with failed attack data, leading to overly optimistic interpretations of their results.

4.3.2 Automated Analysis

In Table 4.1, we report the results of our heuristic measures for 68 classes, bolding any results that we feel indicate serious data design issues. We also run our measures on the multi-dimensional ODDS collection [166], a set of tabular benchmark datasets for anomaly detection (including data taken from *KDD Cup*). The ODDS datasets are mostly non-synthetic, providing a comparison between synthetic network data and real-world data. We present these results in Table 4.2. Our treatment of CTU-13 differs slightly from the other datasets, with more detail in Appendix B.

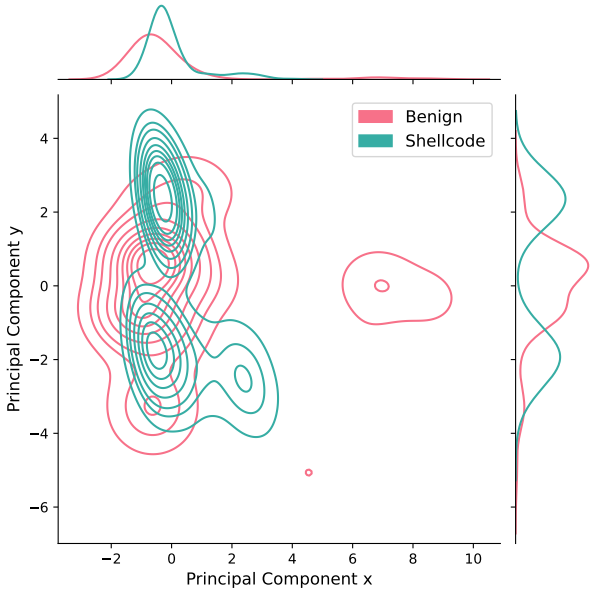


Figure 4.2: Overlap of Shellcode class with Benign traffic in UNSW NB15. We perform principal component analysis to represent each class in two dimensions.

Mislabelled Table 4.1 shows that a minority of classes exhibit significant labelling issues, according to our WL_C and UGT_C measures. Although complex data could cause points to lie on the decision boundary and lead to high scores, our manual analysis refutes this. Instead, mislabelled and contextually benign data cause this overlap, demonstrated in Figure 4.2, where *UNSW NB15*'s *Shellcode* and *Benign* classes coincide heavily due to unrelated DNS traffic. This also occurs in *CIC 17*'s *SQL injection* class, where many flows consist of simple, benign GET requests. Even simple tests, such as our UGT_C, highlight severe issues with unclear attack classes. For

instance, in *UNSW NB15*, common sense checks would expose the *Generic* class's issues, which also predominantly consists of DNS records. However, we also note that our set of *BG Ports* generalised poorly to *CTU-13*. As a result, naive application of our UGT_C measure produced a false positive rate of approximately 20%, as *CTU-13* contains malicious DNS and ICMP traffic.

Highly Repetitive The results of PDD_C show that the attack traffic of these datasets have low diversity. Worryingly, many classes achieve extremely high TC_C scores, implying that classification is equivalent to identifying a small number of flows or, potentially, a single flow. Figure 4.3 shows an example of this problem.

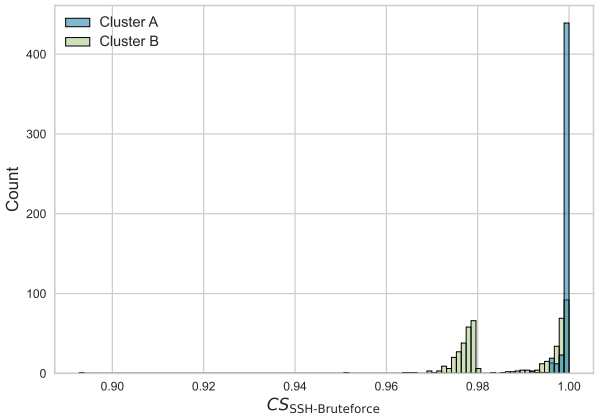


Figure 4.3: CS_C of two clusters of *CIC 18 SSH-BruteForce* class, partially launched against a closed port. Almost all sampled flows are identical in Cluster A.

Such patterns severely degrade the ubiquitous train/validation/test pipeline, preventing the formation of meaningful holdout sets and favoring models that overfit. Consequently, the generalization abilities of ML classifiers — a crucial aspect of NIDS research — are not effectively examined.

Simulation Artefacts According to our HDF_C measure, *highly dependent features* are ubiquitous across the datasets tested. Our one-feature baseline frequently separated attack classes from the background traffic perfectly. Near-perfect scores were also common. Upon examining these features, we observe few connections to the attack's underlying behavior. In *CIC 18*, our baseline achieves a near-perfect F1 score classifying *DoS Hulk* traffic using the *FWD Init Win Bytes* feature. This attack

and feature also appears in *CIC 17*. However, here, our *FWD Init Win Bytes* random forest fails to correctly classify a single *DoS Hulk* flow. This discrepancy highlights the arbitrary nature of the connection between the *DoS Hulk* class and the *FWD Init Win Bytes* feature.

Measuring the impact of artefacts that increase diversity, such as our *artificial diversity* smell, is difficult. A notable example exists in *Bot IoT* and *CIC 17*, where the rate of TCP retransmissions exceeds 34% and 15%, respectively. In standard traffic, this rate typically does not exceed 3% [71]. This disparity introduces unrealistic variability in the flow's features, artificially complicating the data.

ODDS Dataset Collection For comparison, we run our non-network specific metrics — WL_C, PDD_C, TC_C and HDF_C — on the ODDS collection. We limit our analysis to datasets with more than 2500 background samples[7]. The complexity of these datasets varies widely. State-of-the-art methods achieve F1 scores between ~ 0.2 and 1.0 [93, 129, 195]. Our results are collected in Table 4.2.

With the exception of the *shuttle* dataset and some WL_C measures, none of the ODDS datasets attain the extreme scores of our NIDS datasets. We emphasise that these measures are heuristics and, without the original data to analyse, we can't draw conclusions based on features alone. For instance, our results for WL_C correlate inversely with anomaly detection score [195] and the *mammography* or *speech* datasets may simply be challenging benchmarks. However, contrasting the scores attained on these non-synthetic datasets and our synthetic NIDS datasets, we see a marked difference: on a class-by-class basis, our metrics flags issues at a rate three times higher on the NIDS datasets than the ODDS collection. Strikingly, the best scores attained by our metrics occur on ODDS datasets.

[7]We also exclude the *Mulcross* dataset (as the official link was dead at the time of this experiment) as well as the *KDD Cup* based datasets.

Table 4.2: Results of Heuristic Measures on ODDS

Dataset/Measure	PDD_C	TC_C	WL_C	HDF_C
annthyroid	0.89	0.51	**0.6**	0.88
cardio	0.81	0.37	0.12	0.81
cover	0.91	0.31	0.0	0.92
mammography	0.92	0.67	**0.3**	0.78
mnist	0.53	0.0	0.08	0.77
optdigits	0.86	0.06	0.0	0.49
pendigits	0.93	0.68	0.02	0.56
satellite	**0.95**	0.81	0.06	0.8
satimage	0.88	0.75	0.07	0.94
shuttle	**0.98**	**0.95**	0.0	**0.98**
speech	0.80	0.0	**0.3**	0.53
thyroid	0.85	0.3	**0.19**	0.92

4.4 Recommendations

We conclude with some recommendations for using NIDS data, in Sections 4.4.1–4.4.2, and developing NIDS datasets, in Section 4.4.4. Whilst building a 'perfect' dataset is ambitious, these suggestions could improve data quality and minimise design issues.

4.4.1 Testing for Generalisation Explicitly

The oft repeated advantage of machine learning in security is that models can generalise to unseen attacks. Whilst somewhat straightforward in other domains [162, 215], this goal is poorly defined in intrusion detection: along what axes should models generalise? What does it mean for an attack to be 'different' to another? If generalisation is the goal, papers should explicitly define success conditions and datasets should be used in manner that supports this aim. Instead, most work relying on these datasets measure generalisation via a typical train/test pipeline. Given the

design choices of the analysed NIDS datasets, this is often equivalent to training and evaluating models on flows from the **same** attack, a form of data leakage. For the reasons outlined in this paper and others [38], this is not enough to demonstrate model generalisation.

Given existing NIDS data design, we recommend avoiding using training and test attack data from the same class and dataset, as recommended previously [11, 38, 171] and undertaken by some work [14]. Evaluating models based on their cross-class/dataset performance aligns more closely with the intended use case of machine learning-based NIDS: generalising to unseen attack data. Even better, by using prior work in synthetic NIDS data generation [49, 51, 121], it is possible to test whether model architectures can generalise to *arbitrary* attack traffic. Using DetGen [49], we generate malicious data to show how synthetic data can be used like this.

Consider detecting DoS traffic to demonstrate generalisation between different network bandwidths and web page sizes. Based on our analysis of *CIC 17*, the *DoS Hulk* class had *poor data diversity* with *highly dependent features* due to the fixed network conditions and target web page. We generate arbitrary DoS data by randomising webpage size — between 1 and 10MB — and attacker bandwidth — between 1Mbps and 50Mbps. In contrast to *CIC 17*'s single example, we collect volumetric DoS attack data for 60 combinations of bandwidth limit and web page size, which we inject into *CIC 17*. Although this process is artificial, it is similar to domain randomisation in computer vision, where arbitrary synthetic data has been leveraged to improve model generalisation [214]. The increase in diversity is reflected in our heuristics: compared with *CIC 17*'s *DoS Hulk* class, this process produces data that fares considerably better, scoring 0.89 and 0.62 on our PDD_C and TC_C measures, respectively.

By producing such traffic, researchers can query a far larger breadth of an attack's possible data distributions during their testing and evaluation. Critically, this greater test data diversity allows for stronger generalisation claims, better probing of model failures — as shown by Clausen et al. [48] — and weakens the efficacy of naive application of flow statistics, as we show in Figure 4.4. Figure 4.4 also shows the inability of a

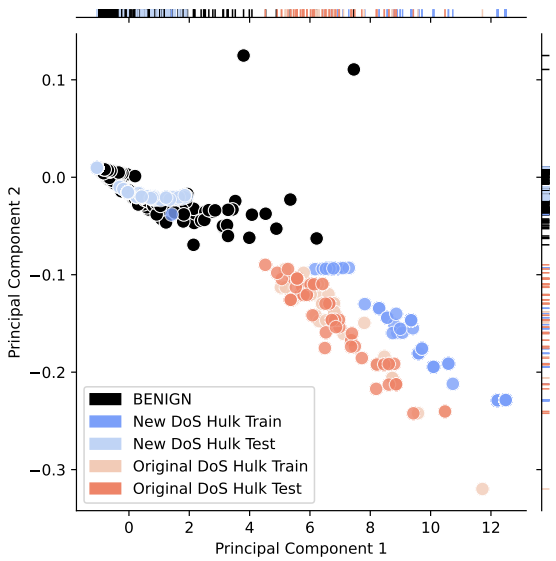


Figure 4.4: Overlap between train/test sets on *CIC 17* via standard evaluation pipeline (Original) vs. our bespoke data (New). Note that the train/test sets of *CIC 17* overlap almost entirely.

standard train/test split on an unmodified *CIC 17* to evaluate generalisation. In contrast, by considering training and test data from different runs of our generation process, naive application of flow statistics results in considerable less overlap between the sets.

4.4.2 Improving Feature Selection

In general, we caution against directly using the feature sets provided with these datasets. It is widely understood that feature engineering is a vital step in the machine learning pipeline. However, NIDS research often completely avoids this process, instead using the provided features by default. These features rarely capture dependencies between flows, necessitating sequential machine learning methods, such as those investigated by Corsini et al. [54]. As far as we are aware, there is no

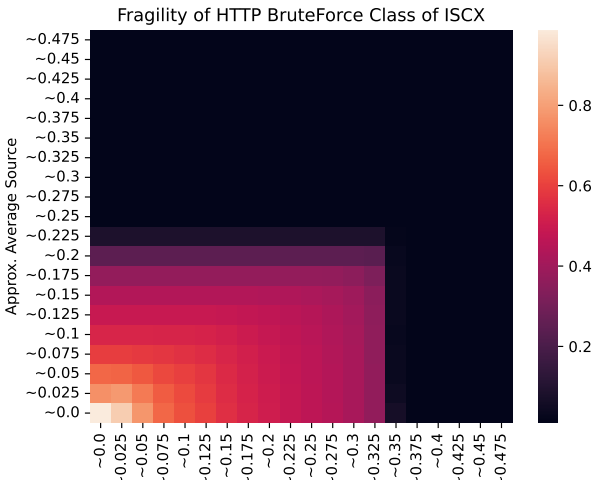


Figure 4.5: Heatmap of baseline model's F1 scores as simple perturbations are applied to the *Total Source Bytes* and *Total Destination Bytes* features. Perfect score is achieved in bottom left (unperturbed). Feature values normalised between $[0, 1]$.

established methodology for associating features with attacks. Even recent attempts to standardise the feature sets amongst NIDS datasets typically focus on the feasibility of collecting such features, rather than the relationship between the chosen features and attacks [14, 177].

As *highly dependent features* can easily occur via poor feature selection, high F1 scores are not enough to validate a NIDS model's performance, as highlighted by Jacobs et al. [107]. We recommend ***justifying*** model performance — using techniques described by Nadeem et al. [154] — connecting important features to an attack's properties across multiple diverse interactions to validate model efficacy. Papers should explicitly state this connection. If a feature is inexplicably highly discriminative then perhaps it should be discarded.

Following Geirhos et al.'s suggestion that adversarial samples are useful for understanding such shortcut features [87], we recommend researchers visualise egregiously dependent features by analysing a baseline model's 'fragility' — i.e., the ease with which unguided perturbations cause samples to cross the decision boundary. We demonstrate this for the *HTTPWeb* class of *ISCX 2012* in Figure 4.5. A simple random forest achieves a near-perfect F1 score separating *HTTPWeb* from the background. However, the most important features — *Total Source Bytes* and *Total*

Destination Bytes — are attacker controlled, unrelated to the attack specification and, thus, potentially shortcut artefacts. Unguided perturbations on these features quickly reduce the model's F1 score without changing the semantic properties, indicating shortcut learning has taken place.

4.4.3 Avoiding Bad Data Smells

Mislabelling Network intrusions data requires unique labelling questions: How should rare, malformed flows be labelled? Should exploratory traffic[1] be labelled as malicious? Should disparate attacks be grouped under the same label? How should causal attack patterns be linked together? The goals of a given dataset will likely inform any answers. Regardless, the author's choices should be explicit and easily available.

When it comes to labelling, the dataset authors are responsible to verify their labelling logic thoroughly, double-checking that the underlying traffic fits their description of the label. However, despite knowledge about the attack strategy, attacker and victim hosts, dataset authors do not have perfect knowledge of what the underlying generated attack traffic looks like. Because of this, improving the ground-truth labelling of a dataset can be an iterative process with improvements provided by multiple domain experts.

Researchers adopting a NIDS dataset should thus not blindly trust the provided labelling logic. When evaluating a model on a dataset, a good first heuristic to detect mislabelling issues is to look more closely at the samples that the model got wrong. Note however, that some of these datasets are so simplistic that an ML model will fit perfectly to them despite faulty ground truth labelling. As such, we still strongly recommend researchers to take an inquisitive stance towards the underlying attack traffic. Does the attacking host with other hosts outside of the victim? Does that traffic look suspicious? Is the victim being attacked by more than just the advertised victim host?

[1] For example, connecting to a server to obtain a cookie used in a later attack

Simulation Artefacts Attacks should be considered anomalous with respect to the normal usage of a specific service. For instance, if an attack targets a particular webpage that does not appear in the dataset in a benign context, then there is no guarantee that a model has learnt any information related to the attack specifically. Instead, it may simply be learning incidental aspects of the attack generation process. These normal behaviours are vital; without them, a dataset's utility is severely limited.

High Repetition Synthetic dataset creators seem to have taken a 'more is more' approach to traffic, leading to high reliance on automated tooling. This, in turn, leads to bloated classes comprised largely of redundant traffic. Using automated tools is not unreasonable; they form the basis of many intrusions. However, these tools typically have multiple settings. For instance, in the datasets analysed, port scanning was a simple, deeply repetitive attack whereas, in reality, tools such as Nmap [140] offer a wealth of configuration options, including scan technique, service detection, OS detection, timing/performance preference, IDS evasion and HTTP proxying. As discussed in Section 4.4.1, the definitional boundaries of an attack are blurry. However, by including a wider breadth of a tool's settings, a more comprehensive view of its behaviour emerges, providing a more in-depth test of a model's generalisation capabilities.

For almost all network interactions, researchers can introduce spacial and temporal variation by modifying the free variables of the interaction. For example, consider video streaming traffic. Whilst many properties of the interaction are fixed via protocol choice — say, RTMP — other aspects can vary widely, such as bandwidth of the client, bandwidth of the server, overall network conditions, watch duration and video bitrate. Randomising these free variables produces diverse traffic, again, providing researchers with a better benchmark to evaluate a model's generalisation capabilities.

4.4.4 Towards Better Dataset Design

It is an ongoing process to consider how our smells should be best addressed when creating a dataset. However, based on our analysis, we can recommend some improvements.

Whilst the advice in Sections 4.4.1 & 4.4.2 is primarily geared towards dataset users, we believe they are also relevant for dataset creators. Namely, creators should include multiple diverse test scenarios for each class, whilst documenting their differences. For instance, tests sets may differ according to spacial or temporal factors, as in TESSERACT [162]. Creators should also highlight potential shortcut artefacts via analysis of baseline models trained naively on flow statistics.

A well-formed dataset should focus on a specific class of attacks or tactics, drawing on vulnerability and adversary analysis [95, 134, 203]. Many published classifiers have lofty ambitions, with model architectures trained and evaluated on all attacks or anomalies within a dataset, instead of a more focused goal. We believe this is heavily influenced by the design of current benchmark datasets, which often contain several disconnected attacks. We also recommend that datasets should include the normal usage of the service being attacked. Otherwise, models may simply be learning incidental aspects of the attack generation process, rather than distinguishing normal and abnormal behaviours.

In NIDS, there is a large gap between the feature and problem spaces. We believe this contributes heavily to labelling issues. For, say, images, converting features into their original source data is trivial and researchers can inspect data easily. In comparison, locating flows in PCAPs is painful. Berkeley Packet Filters [147] can filter PCAPs in a flexible manner and useful filters that isolate malicious from background traffic or separate network services would allow researchers to bridge this gap and should be a critical part of a NIDS dataset, alongside documentation detailing the source of the traffic. We note that CTU-13 appeared to be labelled similarly, simplifying our analysis process massively.

4.5 Related Work

Investigating issues with NIDS data is a well-established area and we build on many prior works. There are several NIDS surveys and overviews that highlight issues: Kenyon et al. [114] critique poor data provenance and simplistic synthetic models; Catillo et al. [39] catalogue several questionable practises in NIDS data, including shortcut artefacts and labelling issues; Apruzzese et al. [15] et al. provide a pragmatic assessment of machine learning for NIDS and recommend using multiple datasets for evaluation; Cordero et al. [51] provide a dataset overview when presenting their synthetic generation framework. These works typically discuss issues at a high level, rarely highlighting specific problems with specific datasets or quantitatively measuring issues as our work does. In contrast, Jacobs et al. [107] identify specific shortcut artefacts in *CIC 17* and *UNSW* via a process similar to HDF_C.

Other works provide more specific but limited NIDS dataset analysis. Analyses of single datasets exist for old, outdated datasets, such as Tavalee et al.'s criticism of *KDD Cup 1999* [3, 210]. In recent criticism, Liu and Engelen et al. [73, 133] provide analyses of *CIC 17/18* and discuss their shortcomings, however, the primary contribution of their work is the discovery of miscalculated flow statistics and labelling issues. Peterson et al. [164] provide a limited overview of pitfalls in *Bot IoT* and Catillo et al. [38] critique public datasets, centering their discussion on the poor transferability of classifiers between datasets.

Despite this prior research, to our knowledge, this is the first paper to present a methodology for uncovering design problems with NIDS datasets, to provide indicators of such issues and to provide a systematic overview of the pitfalls and problems that may be encountered by those who use these datasets. Previously, dataset evaluation frameworks such as that proposed by Gharib et al. [88] consist of simple checks, with no guarantee of the quality of the data that satisfy these criteria. NIDS datasets surveys often propose ways of mitigating problems, such as Ring et al. [171], who suggest using multiple datasets to evaluate performance. Again, these surveys rarely highlight specific problems as our work does.

Similar analyses do exist in other domains, such as time-series anomaly data [231] and image classification [215], as well as more general analyses of data quality in machine learning pipelines [176] or machine learning applied to security [16]. We also note that the term 'data smell' [197] is already used to refer to minor problems with datasets, such as formatting issues.

4.6 Caveats & Limitations

Despite these critiques, we emphasise that imperfect NIDS datasets are still useful tools for researchers and can be completely sufficient for certain tasks. For example, based on their usage of *CIC 17*, we see little reason why the covert communication channels of Chen et al. [42] or the OS fingerprinting of Holland et al. [104] would be impacted by bad data design smells. For NIDS specifically, these datasets do contain valid attack traces which can be properly utilised by research. However, we maintain that thorough understanding is needed to assess a dataset's suitability for a given task.

Whilst we try to be as thorough as possible, our analysis may be limited. Some of our criteria are inherently subjective, reflecting the heterogeneity of these datasets. It is difficult to formulate strict criteria that generalise across all current NIDS datasets, and our suggested heuristics must be applied sensibly to prevent false positives. Whilst we assigned multiple authors to cross-reference our findings, we did not contact the dataset authors for verification, performing this analysis ourselves.

4.7 Conclusion

Via an in-depth analysis of seven popular network intrusion datasets, we identify six data design smells. We find that insufficient data auditing is general across the field of network intrusion research and that these smells could severely bias downstream research.

We hope that the research community takes these issues seriously. Across many facets, synthetic data can provide advantages over real-world data such as complete ground truths, high generative control and repeatability. None of these are utilised by the current crop of datasets and static datasets are still the de facto standard. Further research into producing quality benchmark datasets via synthetic data generation techniques is one potential approach to ameliorate these issues.

Our goal was to inform researchers about dataset contents and limitations. Moreover, our work provides insight into the choices necessary to improve NIDS dataset design. By shedding light on these issues, we hope to stimulate discussions and encourage the community to reassess the suitability and reliability of these datasets for network intrusion research. The paucity of quality, public data is a major problem for the research community as a whole and continued critical evaluation of network data can only be good.

Chapter 5

Measuring Network Traffic Complexity via Spectral Clustering Analysis

Thesis Context *This chapter builds on the work in Chapters 3 and 4, placing benchmark NIDS dataset issues in a wider context by comparing their input complexity with benchmarks from other domains. We also highlight the connection between low data complexity and model overfitting, showing how this limits benchmark dataset utility.*

5.1 Introduction

Despite the ubiquity of synthetic datasets in machine learning-based network intrusion research, their suitability as benchmarks relative to real-world data is unknown. To our knowledge, no work establishes a general baseline for the classification complexity of network anomalies that could be used to evaluate lab-based datasets. Due to their synthetic nature, the *input complexity* of these benchmarks is vitally important to understand their relationship with real-world traffic. Using a novel metric, we aim to bridge this gap in knowledge, demonstrating that these datasets often have minimal data diversity that fails to represent the breadth of even repetitive, volume-based attacks.

Motivation: Lab-based datasets are entrenched in network intrusion research. Machine learning research investigating the detection of network anomalies relies heavily on datasets such as CIC IDS 2017 and CSE-CIC IDS 2018 [188], which have thousands of citations. These datasets are made by scripting interactions across a laboratory environment, simulating the internal network of an organisation. Often, the exact traffic, protocols and interactions within these datasets are poorly documented, raising significant issues regarding the complexity of the data: primarily,

do these scripted processes accurately reflect the variability of real-world traffic? Although carefully generated synthetic data may produce sufficiently complex traffic, recent criticism of these particular datasets suggests that they might not reach this standard [73, 133]. As a result, these may be considerably simplified compared to a real-world network, potentially leading to experimental bias or limiting researchers' abilities to reason about their methodologies. Given this issue, a researcher developing, say, an anomaly detection system, may want to know which dataset most complex, relying on reasonable heuristics such as number of hosts, the size of the dataset or some statistical analysis, such as feature correlations. However, there is no guarantee that these heuristics exhaustively capture data complexity. Thus, machine learning NIDS techniques may depend on simulation artefacts specific to these datasets, failing to generalise to real-world data.

The realism of a network intrusion dataset is difficult to quantify. The paucity of data from equivalent, real-world networks prevents easy comparison between synthetic and real-world data. Furthermore, there is likely not a single fixed notion of what a 'realistic' network intrusion dataset looks like, with numerous dissimilar datasets being admissible, depending on the network topology, services, reliability etc. That said, even given the difficulty of defining 'realistic' data, it is unlikely that some lab-based datasets are valid representatives of real-world networks. For instance, in CIC IDS 2017, FTP traffic consists of downloading a `.txt` file of the *Encryption* Wikipedia page several thousand times [49]. Whilst these connections might be individually realistic, the lack of data *diversity* does not capture the breadth of the FTP protocol, resulting in data with high inter-flow and inter-feature dependencies. Noting this, we aim to evaluate synthetic NIDS datasets by measuring *input complexity*, an expected aspect of real-world data.

Whilst this shift moves us closer to evaluating NIDS datasets empirically, measuring input complexity is still difficult. Whilst numerous measures exist for *classification complexity* between classes, selecting an objective complexity metric is challenging as the Kolmogorov complexity [118] of a dataset, a reasonable starting point, is uncomputable. Compression-based metrics [184] have been used to approximate input complexity, but these fail to capture inter-feature dependencies. Moreover, NIDS

datasets are increasingly being used outside of security contexts as general anomaly or classification datasets. Thus, it would be highly beneficial to compare input complexity of NIDS datasets with those of different fields, for instance, benchmark image datasets, which, again, can not be captured by compression-based metrics.

By measuring inter-feature relationships directly, via, say, feature correlation, we can gain some intuition about the input complexity of a NIDS dataset. However, this approach is limited. For an 80 feature dataset, such as CIC IDS 2017, this requires 6320 comparisons, which is onerous and difficult to contrast across datasets. However, by comparing features on a pairwise basis like this, we have implicitly defined a weighted graph, where the weight of each edge depends on the relationship between features, reflecting the dataset's input complexity. Spectral clustering theory is a natural choice for analysing these complexity graphs, and has previously been used to define classification complexity metrics [30].

Idea: We define an input complexity metric, the *Spectral Input Complexity* or SIC, based on this insight. By embedding datasets into a latent space of fixed size, d, and defining a sensible measure of inter-feature dependency, we can build a weighted graph where weakly dependent features have a higher weighting and strongly dependent features have a lower weighting. We can then compute a $d \times d$ Laplacian matrix and calculate its spectrum, $\{\lambda_0, ..., \lambda_{d-1}\}$. These λ_i will be lower for 'simpler' datasets and higher for 'complex' datasets and, as each λ_i is bounded according to d, we can encapsulate input complexity as a single, normalised number. This measure directly captures feature dependencies and can be used to compare datasets across different fields. Our contributions include:

- **Novelty:** We present a novel metric, SIC, based on spectral analysis to measure input complexity of network intrusion datasets for machine learning. This metric directly captures inter-feature dependencies and can order samples by their contribution to overall data complexity.

- **Comparison:** We compare the input complexity of popular, benchmark NIDS datasets to real world traffic as well as benchmark datasets for other tasks. Based on SIC, we find these NIDS datasets to be considerably less complex than datasets that are understood to be simple, such as MNIST.

- **Dataset Selection:** SIC demonstrates that, despite the recency and expanded size of CSE-CIC IDS 2018, the input complexity of its background traffic is lower than that of its predecessor, CIC IDS 2017, suggesting, for some purposes it is a simpler benchmark dataset.

- **Improvement:** Due to the poor input complexity of volumetric attacks in lab-based datasets, we generate attack traffic to maximise input diversity, demonstrating that this is not a flaw of volumetric attack data generally. We use SIC to quantify this increase in complexity and demonstrate the impact this has on classification difficulty when injected into real-world data.

This chapter consists of work published in "*Measuring the Complexity of Benchmark NIDS Datasets Via Spectral Analysis*" at WTMC. [79]

5.2 Background

5.2.1 Complexity Measures

Existing work on complexity in machine learning primarily focuses on the classification complexity of data, defined as the Kolmogorov complexity of the classification boundary, i.e., the minimal length of a computer program that describes the boundary, with more complex boundaries requiring longer programs. However, Kolmogorov complexity is uncomputable. Thus, practical measures of classification complexity are instead based on geometrical estimates of the decision boundary. Twelve of these geometrical descriptors are outlined by Ho and Basu [102], who characterise measures belonging to three categories: feature overlap, class separability and geometry, topology & density of manifolds. Measures of class separability have been successful: Branchaud

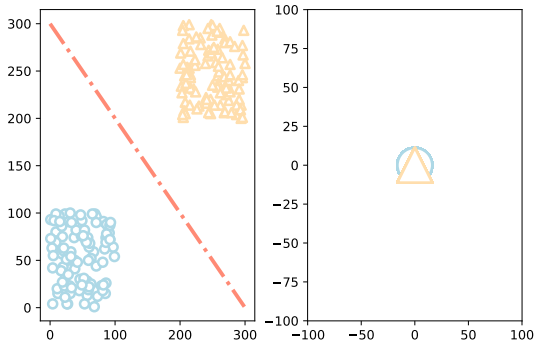


Figure 5.1: A toy example demonstrating the difference between input complexity and classification complexity. On the left, the two classes have high input complexity, but trivial classification complexity whilst the opposite is true on the right.

et al. [30] measure classification complexity via spectral analysis of a graph with weights representing the overlap between classes of a dataset. In this work, we follow this general approach of forming a sensible complexity measure across multiple nodes, condensing this to a single number via spectral clustering.

However, classification complexity is distinct from the complexity of the data within a dataset, such as the visual complexity of a images or the inter-feature correlations of tabular data, or the *input complexity* of the dataset, which we focus on. Figure 5.1 demonstrates the difference for a toy example. Lloyd [137] present a large list of categories of complexity measures, including entropy-based, dimension-based, correlation-based and grammatical-based, reflecting the difficulty of defining complexity for a specific problem. Furthermore, certain methods, including entropy-based approaches, are difficult to calculate for high-dimensional data. Instead, investigating the relationship between input complexity and out-of-distribution detection, Serrà et al. [184] show that compression-based complexity measures are good indicators of input complexity for benchmark machine learning datasets.

5.2.2 NIDS Datasets

Several well-cited, benchmark NIDS datasets do not contain data collected from the real-world. Instead, datasets are sometimes *lab-based*, consisting of data generated via scripted procedures on virtual machines. We do not claim that synthetic network datasets *must* have lower input complexity than real-world datasets. However, these scripted interactions could produce 'simple' data if they are not sufficiently heterogeneous, which we investigate.

In this work, we focus on CIC IDS 2017 and CSE-CIC IDS 2018 — collectively the *CIC* datasets [188] — discussed in Section 2.3. We use the datasets' accompanying features, including measures such as the *Total Backwards Flow Length*. Attack classes are largely *volumetric* in nature, such as denial of service or password bruteforce attacks. As both of these datasets have flaws, we use a partially amended version released by Liu and Engelen et al. [73, 133]

5.2.3 Spectral Analysis

Consider an undirected, weighted similarity graph $G = (V, E)$. We denote the weight of edge E_{ij} as $w_{ij} \geq 0$. The weight of an edge relates to the 'closeness' of two adjacent nodes with closer nodes having a higher weights. A weight of 0 suggests no relationship between nodes. We summarise G via an adjacency matrix A where $A_{ij} = w_{ij}$. Note that A is a symmetric, square matrix of size n, where n is the number of nodes. Spectral clustering provides us with the machinery to divide G into a series of subgraphs of minimal weight by 'cutting' the graph along its edges and incurring a greater 'cost' when cutting highly weighted edges. To do this, we first calculate the Laplacian as $L = D - A$ where D is the degree matrix with $D_i = \sum_j w_{ij}$. As L is symmetric and positive semi-definite, we can calculate n eigenvalues $\{\lambda_0, ..., \lambda_{n-1}\}$. All λ_i are real and greater than 0, except for λ_0, which equals 0 for a well-formed graph. These eigenvalues are known as the *spectrum* of L, and they provide inside into the cost associated the cuts necessary to form each subgraph. These eigenvalues are ordered by increasing size, and the size of a given λ_i is proportional to the cost of subdividing G into i subgraphs.

Dataset/Class	0/A	1/B	2/C	3/D	4/E	5/F	6/G	7/H	8/I	9/J
MNIST	0.559	0.229	0.546	0.483	0.476	0.464	0.532	0.393	0.586	0.432
notMNIST	0.658	0.721	0.65	0.646	0.604	0.646	0.679	0.613	0.508	0.542
	Port 80	Port 443	FTP-BF	SSH-BF	DoS Hulk	DoS-GE	DoS-SL	nmap	DDoS	DoS-SHT
CIC IDS 2017	0.143	0.284	**0.031**	**0.02**	**0.032**	**0.000**	**0.022**	**0.000**	**0.000**	**0.043**
CSECIC-IDS 2018	**0.026**	0.265	**0.000**	**0.000**	**0.039**	**0.066**	**0.001**	**0.000**	**0.000**	-
Port No.	80	443 (TCP)	443 (UDP)	22	25/537	53	30303	33**	-	-
MAWI	0.195	0.21	0.257	0.111	0.142	**0.072**	0.133	**0.0249**	-	-

Table 5.1: SIC calculated across several benchmark datasets. **NB**: A SIC score of zero does not imply that all entries are identical. However, it does indicate that all entries can be separated into a small number of tight, non-overlapping clusters. Port 30303 is Ethereum network traffic. Ports 3300-3399 appear to be P2P torrenting traffic.

5.3 Methodology

5.3.1 Our Complexity Measure, *SIC*

Let x be an input sample and $\phi(x) \in \mathbb{R}^d$ be some embedding of that sample. To measure input complexity via spectral analysis, we first need to produce a graph where more complex relationships between nodes result in edges with higher weights. For our metric, we produce a weighted graph using the inter-feature conditional expectation of $\phi(x)$, i.e., $E(F_j|F_i)$, where $\phi(x) = \bigoplus_{n=0}^{d} F_n$.

Intuitively, given data with high input complexity, it is difficult to extract meaningful features from the data, and conditioning the expected value of $E(F_j)$ on F_i provides little information. For data with low input complexity, the opposite is true. $E(F_j|F_i)$ will yield significant improvement over $E(F_j)$. Our metric relies on the quality of $E(F_j|F_i)$ to measure complexity.

Once we have a low-dimensional embedding $\phi(x) \in \mathbb{R}^d$, $E(F_j|F_i)$ can be estimated quickly, either in a symmetric[1] manner using, say, linear regression or a non-symmetric manner using, say, a simple random forest trained and tested on a small subset of the embedding. We measure the goodness of each test fit by calculating R^2, the coefficient of determination [2]. Doing so leads to a $d \times d$ matrix $\hat{S}$. Next, $\hat{S}$ can be converted to a similarity matrix S by subtracting from the matrix of all ones: $S = 1 - \hat{S}$. Because S is symmetric, up to small deviations, it is thus an adjacency matrix. If we use a non-symmetric method to calculate $E(F_j|F_i)$ we have two weights for each edge in $\hat{S}$. We prioritise the smaller weight, calculating S as $S = 1 - min(\hat{S}, \hat{S}^\intercal)$.

Intuitively, a class with little variation and high correlation between features will produce a highly connected graph. For this graph, any cut will be expensive. We expect higher eigenvalues for graphs representing complex data structures and vice versa for simpler data structures. Thus, we measure input complexity via the normalised sum of eigenvalues:

[1] $E(F_j|F_i) = E(F_i|F_j)$

[2] As extremely inaccurate fits cause R^2 to be negative, we bound it from below by 0.

$$SIC = \sum_i \frac{\lambda_i}{d(d-1)} \in [0,1] \tag{5.1}$$

Effectively, spectral complexity analysis allows us to reduce unwieldy feature relationships to a single, normalised measure. Although dataset agnostic, SIC is particularly applicable to lab-based NIDS datasets due to two common weaknesses: low data diversity and high inter-feature dependency — with flow statistics often containing highly correlated measures (such as 'Total Forwards Packet Length', 'Total Backwards Packet Length' and 'Total Packet Length'). As each entry contributes individually to SIC, we can order entries by their predictability and infer such correlations or patterns. We apply SIC to tabular, numerical datasets as the pre-calculated flow statistics of NIDS datasets are commonly used off-the-shelf in this format.

Whilst our approach is limited by only considering the relationship between pairs of features, rather than, say, arbitrary numbers of features, we show that this is a good approximation of input complexity in Section 5.4.1. Furthermore, measuring input complexity in this way can provide additional insight into the structure of the data via closer examination of the eigenvalues and eigenvectors, which we hope to explore in future work.

5.3.2 Finding a Good Embedding ϕ

For our embedding $\phi(x)$, we consider the latent vector of a fully-connected autoencoder trained using a modified loss function detailed in Section 5.3.2.

Loss Function

$\phi(x)$ produces two outputs: our latent vector z and y such that $y \approx x$. We could train our autoencoder just using a Mean Squared Error loss:

$$\frac{1}{N}\sum_{i=0}^{N}(y_i - x_i)^2 \tag{5.2}$$

However, relying exclusively on Eq. 5.2 gives us few guarantees about the structure of the embedding and, as a result, SIC can be quite volatile. To remedy this, we train ϕ with an additional *cross-covariance loss* term that aims to maximise the covariance between latent features, encouraging smaller values of SIC.

Given z of dimension $batch\_size \times d$, we calculate the cross covariance matrix of z as $\bar{z}^\intercal \bar{z}$ where $\bar{z} = \frac{z-\mu_z}{\sigma_z}$. This cross-covariance matrix has diagonals of value $(batch\_size - 1)$, and each off-diagonal element is bounded by $[-(batch\_size-1)$, $(batch\_size-1)]$. We normalise $\bar{z}^\intercal \bar{z}$ by $(batch\_size-1)$, and calculate the MSE between it and a matrix that maximises (or minimises) the covariance of $\bar{z}$, i.e., $sign(\bar{z}^\intercal \bar{z})$. Our additional loss term is then:

$$\frac{\varepsilon}{N}\sum_{i=0}^{N}\Big(\frac{\bar{z}^\intercal \bar{z}}{batch\_size-1} - sign(\bar{z}^\intercal \bar{z})\Big)^2 \tag{5.3}$$

where ε is a hyperparameter controlling the influence of this term. Our full loss function is:

$$\frac{\varepsilon}{N}\sum_{i=0}^{N}(y_i - x_i)^2 + \varepsilon\sum_{n}^{i}\Big(\frac{\bar{z}^\intercal \bar{z}}{batch\_size-1} - sign(\bar{z}^\intercal \bar{z})\Big)^2 \tag{5.4}$$

5.4 Experiments & Results

For all of our experiments, we use a fully connected autoencoder with three layers of size $(512, 256, 128)$ with $ReLU$ activations and a latent space of size $d = 16$ as ϕ. As overly large values of ε cause the model to learn meaningless embeddings, distorting SIC, we choose ε for each dataset such that Eq. 5.3 is bounded by approximately $0.01 *$ Eq. 5.2, as we found this ratio to produce consistent values of SIC whilst not impacting the mean squared error between x and $\phi(x)$ when trained for 400 epochs.

5.4.1 Relationship with Complexity

Although we aim to measure input complexity, the relationship between our metric and input complexity is not immediate. To demonstrate that SIC accurately reflects input complexity, we perform two simple comparisons.

As an initial common sense check, we create a toy dataset with 128 features, all perfectly linear dependent on one another. Using this as a base, we produce 9 additional datasets, D_i, $i \in [1,9]$. For a given D_i, we perturb all features with random, normal noise $\sim N(0, i*m)$, where m is a multiplicative factor. Thus, higher values of i result in weaker correlations between features and $i < j$ should imply that $SIC(D_i) < SIC(D_j)$.

Secondly, we demonstrate how SIC behaves on a standard benchmark dataset: MNIST [123]. As an approximation to measure the complexity of each MNIST class, we modify the approach of Serrà et al [184]: given a set of inputs x, we calculate the mean Normalised Compression Distance [47] (NCD) between randomly sampled pairs of input, i.e., $NCD(x_1, x_2)$, using `lzma` as our compression algorithm. By using a compression-based distance metric, we encapsulate both the average image complexity as well as similarities in complexity between images. However, unlike SIC, this NCD procedure does not capture inter-feature dependencies between images. We then order the MNIST digits by input complexity by calculating the mean score produced by this NCD procedure, which we compare with *SIC* for each digit.

Results

For both tests, we find that SIC accurately reflects the input complexity of the data, measured by our NCD procedure. On our toy dataset, SIC monotonically increases as the correlations between features become weaker, as in Figure 5.2. On MNIST, we find a Spearman correlation between SIC and NCD of 0.952, indicating an extremely high correlation. With the exception of the digit *4*, the complexity ordering of SIC is identical to that of NCD, as demonstrated in Figure 5.3.

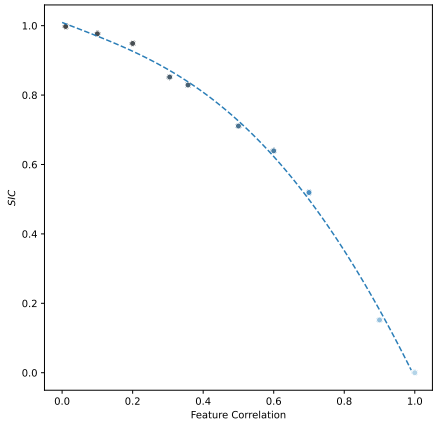


Figure 5.2: Inter-feature correlation vs. SIC on our toy data.

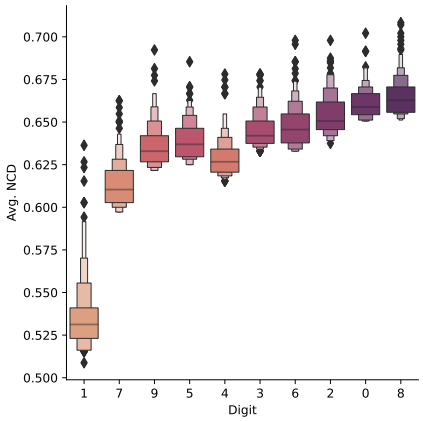


Figure 5.3: Box plot of the NCD metric. Digits are ordered according to their SIC score. With the exception of digit *4*, NCD increases monotonically as SIC increases.

5.4.2 Benchmark Comparisons

Although recent criticism of the CIC datasets [114, 133] may raise questions about their input complexity, it is unclear how simple they are relative to other benchmark datasets. SIC allows us to measure the input complexity of many datasets in a directly comparable manner.

Placing the input complexity of synthetic NIDS datasets in context requires real-world traffic from an internal network. However, this data is extremely difficult to obtain. Instead, we provide two juxtapositions: common benchmark image datasets and backbone network traces. First, we use two benchmark image classification datasets: alongside MNIST, we use notMNIST, a dataset containing 28$\times$28px images of the

letters A through J in a variety of typefaces. Second, we calculate SIC on network traces from the MAWI Traffic Data Repository [94] between 2017 and 2023. This dataset repository contains daily network traces of backbone internet traffic between the USA and Japan. Whilst this is not directly comparable to the traffic of an internal network, it does provide some indication into the breadth of data diversity possible for a network dataset. As CICFlowMeter unfortunately crashes when run on MAWI datasets due to their size, we use our own tooling to extract 41 flow statistics. For all datasets, we calculate SIC on a class-by-class basis. We limit our investigation to 10 classes with over 1000 samples in each CIC dataset, including benign traffic directed towards ports 80 and 443. As the MAWI dataset does not have classes, we categorise flows based on destination port.

Results

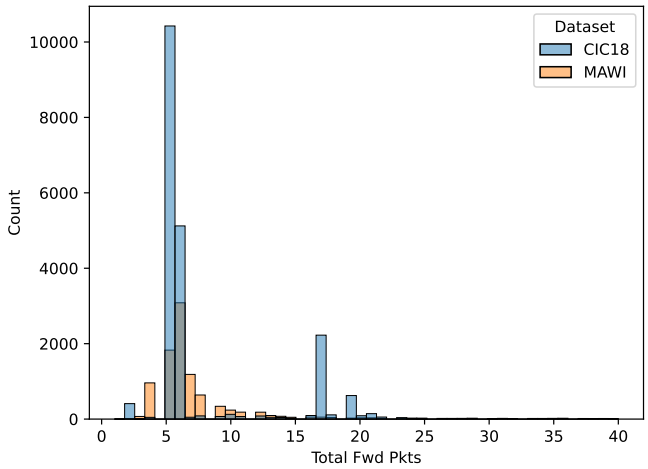


Figure 5.4: Histogram of Total Fwd Packet feature for CIC IDS 18 vs. MAWI Lab data.

We report our results in Table 5.1. Some values stand out, such as the extremely low input complexity of the benign HTTP data in CSE-CIC IDS 2018. Investigating samples that contribute minimally to SIC, we see that, despite benign data having generally higher complexity, CSE-CIC IDS 2018 disproportionately contains flows with 5 forward packets and 5/6 backward packets, as seen in Figure 5.4. The statistics of such flows are highly homogeneous, resulting in low input complexity.

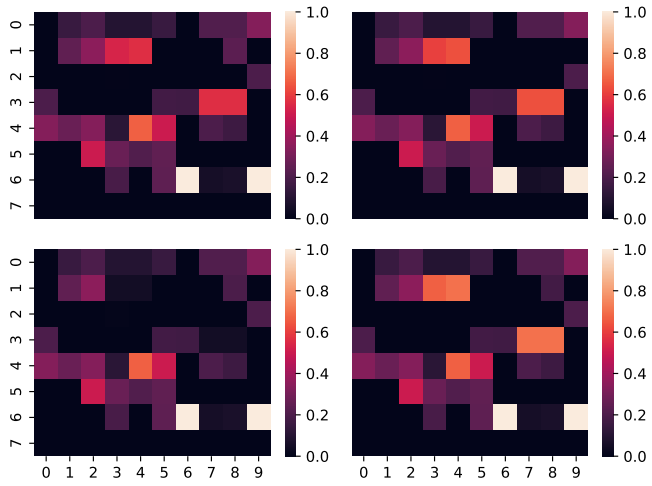


Figure 5.5: Heatmap of normalised features of four random, distinct CSE-CIC IDS 2018 FTP-Bruteforce samples.

Worryingly, according to SIC, all analysed attack classes in CIC IDS 2017 and CSE-CIC IDS 2018 have minimal complexity stemming from trivial data diversity, an example of which can be seen in Figure 5.5, providing numerical support for Engelen et al.'s claim that launching attacks using fixed tooling leads to poor data quality [73]. In contrast, the minority classes of our MAWI data score higher, with the exception of DNS and torrenting traffic. Furthermore, the complexity of network features are considerably lower than that of image datasets, with very few exceptions. Whilst the homogeneity of lab-based data contributes to this, redundancy between features is likely also a factor.

5.4.3 Improving NIDS Dataset Complexity

Whilst Table 5.1 suggests that the input diversity of attack traffic in the CIC datasets is low, this may be a natural consequence of volumetric attacks. After all, these attacks are intentionally repetitive, and it may be incorrect to say they could have more diverse feature distributions.

To test this, we attempt to rectify this poor diversity by generating our own volumetric attack traffic, using SIC to demonstrate the improvement in input complexity. We utilise the synthetic data generation tool DetGen [49], allowing us to simulate traffic with specific network conditions in a controlled and fine-grained manner. Like the CIC

Dataset	F1 Score
CIC IDS 2017 (Hulk)	1.0
MAWI/DetGen	0.681

Table 5.2: F1 score of a simple DoS classifier trained on Packet Size Features, CIC IDS 2017 vs. MAWI Labs data combined with our data.

datasets, we collect denial of service traffic by repeatedly visiting webpages using malicious tooling. However, we aim to maximise both spacial and temporal variation in our generated traffic. We increase spacial diversity by attacking randomly generated webpages between 0MB and 12MB in size. Similarly, we increase temporal diversity by artificially limiting host bandwidth to five rates: 50Mb, 25Mb, 10Mb, 5Mb and 1Mb. Overall, we capture 125 combinations of bandwidth limit and webpage size. As small webpages combined with high bandwidth limits produce highly imbalanced data, we downsample flows generated under these conditions. For a like-with-like comparison with the CIC datasets, we calculate flow statistics using CICFlowMeter.

Results

Following the above procedure, we generate valid DoS data with considerably greater SIC complexity than its CIC equivalents: **0.141**, achieving near-parity with traffic from the Mawi Labs Repository and demonstrating that volumetric attack data is not necessarily devoid of input complexity.

Importantly, generating traffic whilst deliberately increasing input complexity results in challenging test sets that contain flows dissimilar to the training data. In other words, by increasing input complexity, we increase classification complexity in parallel. This likely better reflects the real-world challenges of intrusion detection, such as identifying out-of-distribution attacks, and provides non-trivial benchmarks. We quickly show the impact of this in Table 5.2 by injecting our DoS traffic into the MAWI Labs data and training a simple random forest-based IDS, contrasted with CIC IDS 2017.

5.5 Related Work

As well as general techniques such as KL-divergence or information entropy, several works investigate the input or classification complexity of machine learning datasets. As mentioned previously, Branchaud et al. [30] use a similar spectral clustering-based metric to measure the classification complexity of various image datasets. In their analysis of the relationship between input and classification complexity and model capacity, Mei et al. [148] propose an input complexity measure based on the relative entropy of randomly sampled subsets of a dataset. However, like other measures, this fails to directly capture inter-feature relationships. Recently, Schmidt et al. [179] and Carmon et al. [36] have investigated the relationship between sample complexity and adversarial robustness.

The evaluated datasets have been criticised before in literature reviews and dataset overviews [114, 211]. However, these critiques tend to be high-level with little accompanying evidence. More specific critiques have been raised by Liu and Engelen et al. [73, 133]. Whilst not the focus, their results point towards the low classification complexity of the CIC datasets, with simple random forests achieving extremely high F1 scores.

5.6 Conclusion

We present an analysis of the input complexity of two highly-cited benchmark intrusion detection datasets via a novel metric, SIC, experimentally justifying its efficacy. Using this measure, we contrast the input complexity of benchmark NIDS datasets with benchmarks from image classification as well as internet backbone traffic. We consistently found these NIDS datasets to be generatively trivial compared to other benchmarks, particularly the volumetric attack traffic which makes up the majority of the malicious data. Following this, we briefly investigate whether this simplicity is an inherent property of volumetric attacks. Using DetGen, we generate attack traffic with

higher complexity, pointing towards better methods for generating synthetic attack data. Altogether, our results have potentially significant consequences for the utility of the examined datasets as benchmarks, the realism of their traffic and their suitability for developing machine leaning-based intrusion detection methods.

The measure has some limitations. It is sensitive to hyperparameter selection and comparisons between datasets are easiest when ϕ is fixed. Large outliers in the data must be dropped or normalised carefully as they can impact SIC disproportionately when scaled. For future work, we would like to utilise additional information related to SIC, such as the eigengaps or the eigenvectors, as a similarity measure to compare disparate network statistics. We would also like to investigate other measures of feature relations to determine graph weightings, including entropy-based approaches, and apply the measure to other domains.

Chapter 6

Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models

Thesis Context: *This work demonstrates how simple models with curated data can perform better than the equivalent state-of-the-art. By focusing on augmenting data via specification-driven adversarial training, we improve model reliability and interpretability. Furthermore, the experiments in this chapter follow the recommendations outlined in Chapter 4.*

6.1 Introduction

Given the adversarial nature of intrusion detection, NIDS must be robust to evasive behaviours. Security is often likened to a cat-and-mouse game and this is seen in neural network robustness: adversarial attacks are proposed, defences are designed, before new attacks are created, restarting the cycle. In contrast to other defences, *neural network verification* promises an alternative, formal approach which guarantees robustness within specified parameters. Recent work on methods like Reluplex [112, 230] uses logical specifications to impose constraints on neural networks, restricting their outputs for given inputs, thus providing mathematical guarantees about robustness.

But adversarial robustness is not the only form of robustness needed: NIDS should also be robust to inherent problem-space diversity, such as changes to network bandwidth or to minor variations in malicious traffic. In research NIDS pipelines, these differences are often not evaluated. One reason is the scarcity of sufficiently diverse public NIDS data; models are often trained and tested on the same dataset, without testing for robustness to concept drift. Furthermore, without sufficient data, it is difficult

to validate that models do not overfit to arbitrary features (which may not be truly linked to attack classes being recognised). We see a marked reduction in the performance of typical models when trained and tested using disparate datasets, even if the datasets contain similar types of malicious traffic. In other words, *cross-dataset generalisation* is often poor. This is likely also true in real-world scenarios when IDS developers have limited access to sufficiently varied data.

This work addresses these twin issues of verifiable robustness and generalisation by ensuring that the models we train adhere to certain *global constraints* across benchmark datasets and bespoke data, which we formally verify. In contrast to other fields, such as computer vision where specifications are defined ***locally*** for each input image, often via ℓ_p-balls, discussed in Section 6.2.3, network data is structured enough that we can specify expected properties ***globally***. For example, we can specify the structure of well-formed TCP handshakes, whilst allowing our models to learn other relationships in the data independently. An analogy with image recognition may help: we can verify that a model classifying animals biases black and white pixels when identifying pandas, while allowing the network to infer the general shape of the animal.

Our global constraints define known regions of benign and malicious traffic, allowing us to enforce model behaviour for corresponding network flows. We do this by generating additional malicious and benign data via *specification-driven adversarial training*, finding counter-examples via PGD [142] that lie within our benign/malicious constraints and adding them to the training set. This process not only expands the, initially limited, training data, but also helps the model satisfy our specifications. By strengthening expected model behaviour in this manner, our models prioritise features that directly reflect attack behaviour over less relevant features that, in the absence of sufficient data, may cause overfitting. In other words, our constraints not only guarantee adversarial robustness in certain regions of the feature space, they also reinforce model generalisation conditions. We show a high-level overview of this process in Figure 6.1, with more detail in Section 6.3.

We define our properties using *Vehicle* [55, 56], a specification language for writing and testing logical constraints for neural networks. Vehicle offers a high-level, readable DSL which acts as a front-end for the Reluplex-based verifier Marabou [230]. Vehicle and Marabou are emerging tools, and our work demonstrates their non-trivial application to a new domain area.

Altogether, this novel approach yields several benefits: first, our models generalise better than state-of-the-art architectures between benchmark datasets as well as our own bespoke data; second, our models are verifiably robust, eliminating the risk of adversarial perturbations in certain regions; finally, by examining and contrasting satisfiable and unsatisfiable specifications, we can better reason about model behaviour. To summarise, our contributions include:

- **Specifications**: We write, apply and test several specifications for model verification. In doing so, we eschew ℓ_p norm measures of flow closeness. Instead, we embed expert knowledge about intrusions in a *global* manner, producing specifications that are non-trivial, novel and performant.

- **Generalisation**: In contrast with most NIDS research, we explicitly outline the generalisation conditions of our networks, considering both *cross-dataset* and *cross-attack* generalisation. Alongside standard benchmark datasets, we also generate our own bespoke attack data.

- **Verification**: Using these specifications and adversarial training, we produce NIDS models that are mathematically, verifiably robust against adversarial and natural perturbations in certain regions of feature space. We also constructively utilise specification counter-examples to examine model weaknesses and rank strategies for producing evasive traffic. Using this process, we prove that, for our models, delaying packets is the most effective evasive strategy.

- **Performance**: Our simple, verifiable model outperforms a comparable state-of-the-art approach under our generalisation conditions, achieving up to a 40% improvement in generalisation. Additionally, the SoA model is also too complex to be verified according to our specifications using current frameworks.

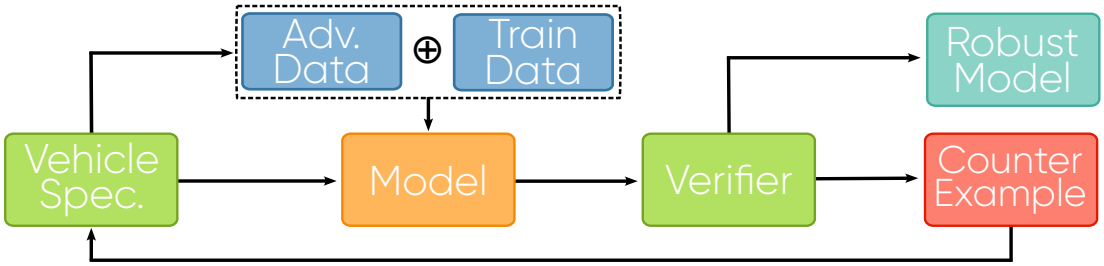


Figure 6.1: The specification-driven adversarial training pipeline.

The work presented in this chapter is awaiting publication as "*Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models*" [82]. This work was performed in concert with Marco Casadio, who contributed the adversarial training regime and coverage metrics. All other work presented is mine.

Structure of the chapter. Section 6.2 covers some basic background on NIDS, adversarial robustness and recent work on neural network verification. Section 6.3 describes the methodology for our new verified NIDS pipeline, in particular, explaining the format and meaning of specifications. Then Section 6.4 describes eight example applications of verification in NIDS, including cross-dataset generalisation (robustness), cross-attack generalisation and using the verification process to generate realisable evasive traffic. This last application can help understand model failures using the specification language, suggesting fixes or pre-processing. Section 6.5 discusses the limitations of our approach. Finally, Sections 6.6 and 6.7 discuss related work and summarise the paper.

6.2 Background

6.2.1 Network Intrusion Detection

ML-based NIDS ability to generalise is highlighted as a key advantage over signature-based methods, a driving motivation for the field. Whilst some works explicitly test for generalisation [11, 14], the rarity of cross-dataset evaluation, or explicit generation of an alternative test data, fails to interrogate this assumption. Furthermore, when

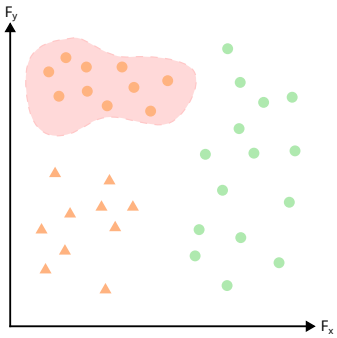


Figure 6.2: Initial decision boundary: Fails to generalise, given lack of representative data

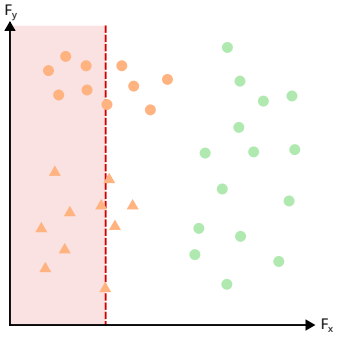


Figure 6.3: Global Specification: We underspecify a decision boundary to force the model to learn relevant features

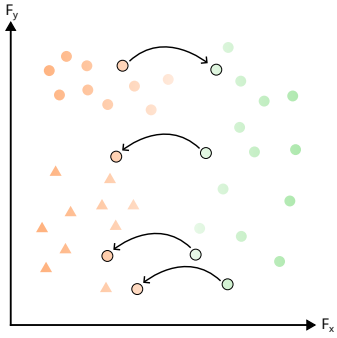


Figure 6.4: Local Specification: We specify behavioural constraints for individual entries, dependant on model output

●: Malicious Data - Training, ▲: Malicious Data - Test, ●: Benign Data, F_x: A *relevant* feature, F_y: An arbitrary feature

such results are reported, they can be disappointing. This is despite the fact that some benchmark datasets are highly similar, such as CIC IDS 2017 and 2018 [188] generating their benign traffic via *B-profiles* [187] and containing conceptually similar attacks.

6.2.2 Neural Network Verification

Recently, several neural network verification frameworks have emerged [20, 22, 132, 198]. These consist of complete verifiers which return *true/false* and incomplete verifiers which return *true/unknown*. Complete verifiers can be based on Satisfiability Modulo Theories (SMT), Mixed-Integer Linear Programming (MILP) or Branch-and-Bound (Bab). SMT-based verifiers [112, 165, 230] and MILP-based approaches [45, 139, 213] encode the provided specification as a conjunction of linear inequalities or as a mixed-integer linear programming problem, respectively. Their main limitation is poor scalability, while their strength is that they precisely encode constraints. In contrast, branch-and-bound based verifiers (such as [32, 33, 77, 86]) over-approximate the constraint, sacrificing precision in favor of scalability. In this work, as our network are relatively small, we opt for complete verification and we use the SMT-based verifier Marabou [230].

Marabou resolves queries about neural networks and their properties in the form of constraint satisfaction problems. Relying on a version of the simplex method [58], modified to work for networks with piece-wise linear activation functions, called ReLUplex [112], Marabou infers bounds for each node in a fully-connected network. It then attempts to satisfy these linear constraints, producing counter-examples for unsatisfiable conditions. To write specifications and invoke Marabou, we use the front-end Vehicle [55, 56]. Vehicle is a domain-specific specification language, a high-level, functional language for writing mathematically-precise neural network specifications.

Given scalability and tightness challenges, neural networks trained via a standard pipeline are unlikely to satisfy any meaningful constraints. Instead, to satisfy non-trivial constraints, networks are trained for *robustness*. After robust training, models often achieve higher verification success and are more likely to satisfy the desired properties. Robust training techniques can be grouped into data augmentation (new data is created by manipulating existing data points), adversarial training, where the worst performing counter-examples within a bounded region are added to the training set, and certified training, which provides mathematical guarantees about model behaviour within certain bounds.

We adversarially train our models to satisfy the desired property in a *specification-driven manner*. In other words, we generate additional traffic that adheres to our specification. We do this via projected gradient descent, a state-of-the-art method, restricted to regions encapsulated by the desired property.

6.2.3 Adversarial Robustness in NIDS

Given their security-centric role, adversarially robust NIDS are vital. Despite this, there is a disconnect between existing NIDS evasion methodologies and those often used to determine model robustness. In the latter case, ‘closeness’ can be defined as a region of perturbed inputs alongside some norm [127]. *Classical robustness* [37], intuitively, states that small variations to model input should result in small changes in its output. Mathematically, given model N with input x, is said to be robust iff for all ε there exists δ such that:

$$\forall x, x_0.\ |x - x_0| < \delta \Rightarrow |N(x) - N(x_0)| < \varepsilon \tag{6.1}$$

However, in the network domain, attacks that are identical in purpose can have wildly varying flow statistics and existing IDS evasion techniques do not map neatly onto the above notion of a perturbation. Instead, several papers craft adversarial examples with domain constraints in mind, aiming to maintain the original flow semantics[105, 189, 191, 220].

6.3 Methodology

To our knowledge, we are among the first to apply neural network verification tools to the network security domain. Rather than simply apply tools to a new setting in the most straightforward way, we want to explore how verification can spawn new ideas in NIDS research. The Vehicle [55] tool is ideal for this, allowing to pinpoint and interpret details of model behaviour.

To develop the methodology, we have to confront limitations of current verification frameworks whilst ensuring that our specifications are non-trivial. First, the Vehicle language is limited to simple quantifiers, boolean operands, conditionals and arithmetic. Thus, our feature set needs to express concrete properties of the traffic, avoiding features with high inter-feature dependency and low explainability, such as '*Forward Packet Size Mean*' and '*Total Packet Size Deviation*'. Second, for our properties to correctly bound *benign/malicious* traffic, we rely on global constraints that accurately reflect domain knowledge. Third, we have to write sensible constraints that are both non-trivial and do not cause exponential blow-up of Marabou. However, we note that rules for signature-based NIDS have similar complexity and expertise requirements. We detail our approach to these issues in Sections 6.3.1, 6.3.2 and 6.3.3 respectively.

Our verification examples in Section 6.4 have unique specifications with many unique constants. For readability, we replace these constants, writing lower bounds for a feature F as α_F, upper bounds as β_F and constants as γ_F. Thus, a simple, arbitrary specification for model N and input x that constrains N to output class C_0 when the first, say, five features lie within certain bounds takes the form:

$$\texttt{Bound5Feats} : \forall i \in [1,5].\ \alpha_i \leq x_i \leq \beta_i \implies N(x) = C_0 \tag{6.2}$$

We can also use specifications to check for best practises. For instance, all models we develop adhere to `ValidInput`, which ensures that all features lie between 0 and 1:

$$\texttt{ValidInput} : \forall i, x.\ 0 \leq x_i \leq 1 \tag{6.3}$$

Throughout this section, we introduce our methodology with our first sanity-checking verification experiment to examine whether, given a simple attack and weak generalisation conditions, it is possible to write a satisfiable robustness specification. To do this, we train a model to detect portscan traffic, generated using the nmap tool [140], and verify its correctness properties.

6.3.1 Feature Set & Data

As large feature sets can cause overfitting to arbitrary features [107], we use a restricted feature set. Via bespoke tooling, given a flow of size N, we extract packet-level features from the first m packets: packet sizes, inter-arrival times (IATs), TCP flags and packet directions. We supplement these with two features that are constant across a flow: transport protocol — TCP or UDP — and time since last flow with identical Source/Destination IPs — a feature we call *TimeElapsed*. Otherwise, when $N \leq m$, we zero pad features for 'missing' packets. For a given flow x, we write the packet features of the i-th packet as: $x_{\textit{FeatureName-}i}$.

We test our model's cross-dataset generalisation performance with an even split between the benign and malicious classes. In total, we consider *DoS Hulk*, *DoS Slowloris*, *FTP/SSH Bruteforce* and *SQL injection* traffic as our malicious classes. We sample training data from CIC IDS 2017 and consider two test datasets. For the first, we extracting benign and attack data from CIC IDS 2018. For the second, we use our own bespoke attack data, aimed at evaluating model performance when generalising to a variety of network conditions and flow lengths. We generate this traffic via DetGen [49], a deterministic network generation framework. When applicable, we alter the *temporal* characteristics of our traffic by limiting attacker bandwidth to 10mb, 25mb and 50mb, altering both inter-arrival times and intra-arrival times i.e., *TimeElapsed*. To alter the *spatial* characteristics of our traffic, we assume that an attacker is able to attack different webpages/databases or use evasive packet-padding features of intrusion tooling. We generate attack traffic for five scenarios, corresponding to different webpage/database sizes or padding lengths and combine this with benign traffic from CIC IDS 2018. We call this our '*DetGen*' dataset. We preprocess data and select models via reference to the training data **only** to prevent data leakage and to simulate detecting unknown attacks.

6.3.2 Global vs. Local Specifications

Our aim is to write specifications that improve model robustness and cross-dataset generalisation, by encoding expert knowledge. We do this *globally*. In other words, each property is defined in a fixed manner for the entire input space. We contrast this with *local* specifications, where each property is dependant on specific inputs, such as *classical robustness* defined in Section 6.2.3. Both formulations aim to enforce model robustness. We demonstrate the difference visually in Figures 6.3–6.4.

We divide our features into three categories depending on how an expert may interpret them:

1. ***Related*** *features* which are directly relevant for classification.

2. ***Bounded*** *features* whose behaviour is fixed in certain intervals.

3. ***Unknown*** *features* whose relationship to classification is unknown.

Consider the problem of detecting denial of service attacks flooding an HTTP server. The defining feature of this attack is that a webpage is repeatedly accessed in quick succession. Thus, our $TimeElapsed$ feature is likely a *related* feature. In contrast, small changes to the size of the HTTP GET request packet — which can naturally vary — is likely irrelevant to whether a flow is malicious or benign. We would consider this feature to be *bounded*. Importantly, we do **not** want to exclude this feature entirely, as large changes could still be indicative of abnormal behaviour, such as a large payload encoded as a URL parameter. Instead, we aim to verify model behaviour in a bounded, specified interval. Finally, the relationship between, for example, packet flag features and classification may be unclear. In this case, we would consider these to be *unknown* features. For other attacks, this categorisation would almost certainly change.

We apply a similar logic to our nmap example. As standard nmap flows are short and highly predictable, consisting of quick, simple three-way handshakes with fixed packet sizes, we write specifications that describe these properties as being *related* or *bounded*, detailed below.

Related Features For our global specification, we treat *related* and *bounded* features in a similar manner, specifying model behaviour explicitly within certain regions. For *related* feature $\hat{F}$, we specify an interval where the model must make fixed classification decisions. Formally, given flow x which takes on value $x_{\hat{F}}$ for feature $\hat{F}$ and model N that performs binary classification $N(x) \rightarrow C_i, i \in [0,1]$, we fix i and specify an interval $[\alpha, \beta]$ such that:

$$\forall x. x_{\hat{F}} \in [\alpha, \beta] \implies N(x) = C_i \tag{6.4}$$

In words, all **arbitrary** x with $x_{\hat{F}} \in [\alpha, \beta]$ must be classified as C_i.

For our nmap verification, we highlight related features by specifying that malicious flows must have sufficiently 'quick' *TimeElapsed* values and that packet sizes are 'close' to those of a standard nmap connection:

$$\texttt{MalElapsed} : x_{\textit{timeElapsed}} = 0.0 \vee x_{\textit{timeElapsed}} \leq \beta_{timeElapsed} \tag{6.5}$$

$$\texttt{MalPktSize} : \forall i \in [1,3].\ \alpha_{pktSz-i} \leq x_{\textit{pktSz-i}} \leq \beta_{pktSz-i} \tag{6.6}$$

$$\texttt{nmap} : \forall x.\ \texttt{MalElapsed}(x) \wedge \texttt{MalPktSize}(x) \implies mal \tag{6.7}$$

Bounded Features In contrast, for *bounded* feature $\hat{F}$, we wish to specify an interval where altering $x_{\hat{F}} \in [\alpha, \beta]$ does not change model behaviour for **fixed** x. Formally, given two flows x and $\bar{x}$ which differ only on feature $\hat{F} \in [\alpha, \beta]$:

$$N(x) = C_i \implies N(\bar{x}) = C_i \tag{6.8}$$

In other words, altering $x_{\hat{F}}$ within $[\alpha, \beta]$ should not affect the model's output.

Unknown Features Although we write 'specifications', we stress that we do not attempt to completely encapsulate attack definitions — or that this is even possible to do by hand. Thus, we include features where we do not specify their relevancy for classification, leveraging our model's ability to learn the relationship for us.

Importantly, a feature's category can be defined **implicitly** across multiple rules. Given some property P_1 which bounds feature $F_1 \in [\alpha_1, \beta_1]$ and property P_2 which bounds $F_2 \in [\alpha_2, \beta_2]$, the following rules imply F_1 is *bounded* and F_2 is *related*:

$$\begin{aligned} S_1 : P_1 \wedge P_2 &\implies N(x) = C_0 \\ S_2 : P_1 \wedge \neg P_2 &\implies N(x) = C_1 \end{aligned} \tag{6.9}$$

6.3.3 Principles for Global Specifications

When writing specifications, we follow several principles to ensure their utility whilst adhering to the technical limitations of our verification framework, which we detail here. As they differ for each of our examples, we present more specifications in Section 6.4.

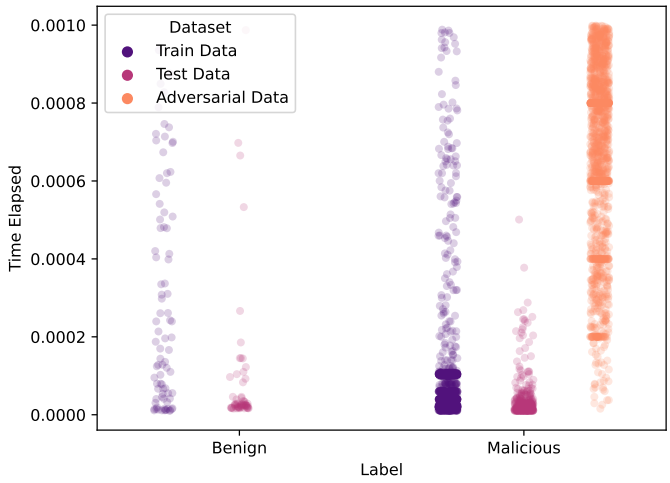


Figure 6.5: Expansion of related feature, *TimeElapsed*, via the adversarial training process in Section 6.4.1. Adversarially generated data is less dense in areas where the training data is concentrated and vice versa.

Input Expansion As we generate adversarial data, our specifications expand the space of model inputs. This is good: NIDS datasets often have limited diversity and expanding the space of malicious input could improve model generalisation. In particular, our adversarial training procedure 'fills' the loose bounds around features in our specification, particularly in areas not covered by our training data, as seen in Figure 6.5. This informs our choice to *underspecify* the decision boundary of our classifier as wider feature bounds may lead to contradictions between properties or cause our model to incorrectly learn benign/malicious behaviours.

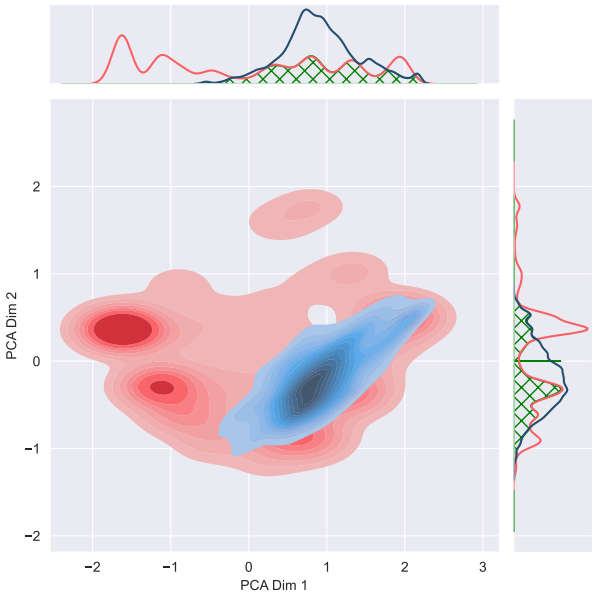


Figure 6.6: Projection showing the limited overlap between training and adversarial data in Section 6.4.1, representing the restricted verifiable subspace.

Input Reduction Although we want to write an exhaustive specification that captures all malicious flows in the training dataset (whilst also capturing all realisable malicious flows that do not appear in the data), in reality, this is impossible. One highly limiting factor is that `or` statements cause exponential blow-up on the Marabou backend. For instance, we can write a property that must hold regardless of the permutations of packet directions:

$$\begin{aligned} \texttt{allDirections} : \forall i \in [1, 10].\, x_{pktDir-i} = \text{in} \\ \vee\, x_{pktDir-i} = \text{out} \implies y \end{aligned} \tag{6.10}$$

However, this requires 2^{10} checks to verify and poorly written specifications can easily require 10000+ checks, which is overly costly. Thus, as well as generating additional inputs as above, we must also ***restrict*** our verifiable input space to prevent exponential blow-up of the SMT solver.

Our specifications restrict the space of input, limiting verifiable flows to certain regions. For instance, we can only consider flows that conform to expected protocol behaviour (which we determine via packet sizes and flags). Similarly, we limit properties that may cause Marabou blow-up by only considering the most common combinations in the training data. Together, these allow us to write specifications representing realistic traffic, based on large swathes of training data, without onerous resource usage. For instance, in Section 6.4.1, we consider only four combinations of packet directions, representing approximately 60% of malicious training data whilst reducing the number of checks from 1024 to 16.

This limited subspace can be visualised as the space occupied by our adversarially generated data, which conforms to these constraints. In Figure 6.6, we demonstrate this reduced overlap via principal component analysis along our bounded features.

Counterexample-Guided Specifications Whilst it is possible to write a specification that is immediately satisfiable, in our experience, this is unlikely and often stems from trivial conditions. Due to the gap between feature and problem space in NIDS data, it is difficult for specifications to cover all edge cases, which then may fail in some unexpected region of feature space. For instance, when specifying the behaviour of benign traffic, one might not bound a flow's duration. However, such a specification will likely be unsatisfiable as, say, the model may classify negative flow durations as malicious. However, in this case, this flow is not actually realisable, suggesting a disconnect between our benign specification as-written and what the model understands to be benign.

Thus, we write our specifications in a counterexample-guided manner. Given a counterexample x, we inspect whether x conforms to our understanding of the malicious data. Keeping a human-in-the-loop, if we identify x as unrealisable, due to, say, impossible packet sizes, inter-arrival times or flag combinations, we rewrite our specification to eliminate this mistaken counterexample. Our specifications in Section 6.4 vary in complexity and sensitivity, with some necessitating 40+ counter-examples in order to be verifiable.

Nmap Results As can be seen in Table 6.1, we achieve a perfect F1 score classifying our basic nmap traffic following this process. Due to the attack's homogeneity, this is not surprising. However, unlike state-of-the-art equivalents, our model is mathematically guaranteed to be robust against minor perturbations.

Importantly, our specification is only satisfied *after* adversarial training. Before, none of our properties could be verified, even after we reduced our global bounds. This is worrying, given how we replicate the common, standard evaluation pipeline. This highlights that, given insufficient data, machine learning-based NIDS can easily overfit, justifying using model verification to guarantee model robustness, even for simple, repetitive attacks.

6.4 Applications of Verification

All models we train are of fixed architecture, consisting of a feedforward network with shape $(256, 128, 2)$ with approximately 44000 parameters. For these networks, Vehicle takes roughly 8 minutes to verify all global specifications.

6.4.1 Cross-dataset Generalisation

Experiment Following our nmap verification, we consider more complicated attacks with more complex specifications. First, we train a model to detect all volumetric denial of service attacks in the CIC datasets, aiming to generalise to arbitrary network conditions and page sizes. To ensure that our model learns the salient features of each attack, we write properties that reinforce these generalisation aims. Our DetGen data consists of HTTP flood traffic generated in the manner outlined in Section 6.3.1. As a benchmark, we compare our results to LUCID [69], a state-of-the-art DoS classifier, as well as our simple model without adversarial training.

Specification All DoS attacks in the CIC datasets function via asymmetric resource usage, repeatedly accessing a specific HTTP page. Some attacks increase the target load by delaying responses, leading to artificially high inter-arrival times and flow durations, or deviate from the expected behaviour of a TCP/HTTP connection. We aim to capture these defining qualities in our properties. In addition to `ValidInput` and `MalTimeElapsed` from Section 4.2, we write three base properties:

$$
\begin{aligned}
&\texttt{ValidTCPHandShake}: \\
&x_{\textit{pktFlag-1}} = \text{SYN} \wedge x_{\textit{pktSize-1}} = 52 \wedge x_{\textit{pktDir-1}} = \text{out} \wedge \\
&x_{\textit{pktFlag-2}} = \text{SYN + ACK} \wedge x_{\textit{pktSize-2}} = 52 \wedge x_{\textit{pktDir-2}} = \text{in} \wedge \\
&x_{\textit{pktFlag-3}} = \text{ACK} \wedge x_{\textit{pktSize-3}} = 40 \wedge x_{\textit{pktDir-3}} = \text{out} \wedge \\
&x_{\textit{protocol}} = \text{TCP}
\end{aligned} \tag{6.11}
$$

$$
\begin{aligned}
&\texttt{ValidHTTPConn}: \\
&x_{\textit{pktFlag-4}} = \text{ACK + PSH} \wedge \alpha_{pktSize-4} \leq x_{\textit{pktSize-4}} \leq \beta_{pktSize-4} \wedge \\
&x_{\textit{pktDir-4}} = \text{out} \wedge \\
&x_{\textit{pktFlag-5}} = \text{ACK} \wedge x_{\textit{pktSize-5}} = 40 \wedge x_{\textit{pktDir-5}} = \text{in} \wedge \\
&x_{\textit{protocol}} = \text{TCP}
\end{aligned} \tag{6.12}
$$

$$
\begin{aligned}
&\texttt{ValidIATs}: \\
&\forall i \in [2,10].\ 0.000001 \leq x_{pktIATs-i} \leq 0.05 \vee \\
&\sum_i x_{pktIATs-i} \leq 0.2
\end{aligned} \tag{6.13}
$$

$$
\begin{aligned}
&\texttt{validSizes}: \\
&\forall i \in [1,10].\ (x_{pktSize-i} >= 40) \wedge (\sum_{5}^{10} x_{pktSize-i} >= 400)
\end{aligned} \tag{6.14}
$$

These form the building blocks of our four main properties which specify malformed TCP connections, infrequent (presumably benign) HTTP traffic, volumetric HTTP traffic and volumetric HTTP traffic with unusual IATs: respectively, `invalidTCPHTTP`, `GoodHTTP`, `HulkAttacks` and `SlowIATsAttacks`.

$$\texttt{invalidTCPHTTP}:$$
$$\forall x.\ \texttt{validInput}(x) \wedge (\neg\texttt{validTCPHandshake}(x)$$
$$\vee \neg\texttt{validHTTPConn}(x)) \implies \mathit{mal} \tag{6.15}$$
$$\texttt{GoodHTTP}:$$
$$\forall x.\ \texttt{validInput}(x) \wedge \texttt{validTCPHandshake}(x) \wedge$$
$$\texttt{validHTTPConn}(x) \wedge \texttt{validTimeElapsed}(x) \wedge$$
$$\texttt{validIATs}(x) \wedge \texttt{validSizes}(x) \implies \mathit{ben} \tag{6.16}$$
$$\texttt{HulkAttacks}:$$
$$\forall x.\ \texttt{validInput}(x) \wedge \texttt{validTCPHandshake}(x) \wedge$$
$$\texttt{validHTTPConn}(x) \wedge \neg\texttt{validTimeElapsed}(x) \wedge$$
$$\texttt{validIATs}(x) \wedge \texttt{validSizes}(x) \implies \mathit{ben} \tag{6.17}$$
$$\texttt{SlowIATsAttacks}:$$
$$\forall x.\ \text{if } \texttt{ValidInput}(x) \wedge$$
$$\texttt{ValidTCPHandShake} \wedge \neg\texttt{ValidIATs}(x) \implies \mathit{mal} \tag{6.18}$$

Results Table 6.1 summarises performance of our models alongside comparative benchmarks. Since our verification specifications are global rather than local, we do not express verifiability as a percentage of verified subspaces. Instead, the success of our global specifications is presented as a binary outcome: either verified or not verified.

We successfully verified a series of complex specifications whilst enhancing cross-dataset generalisation. Training with our adversarial loss, we improve cross-dataset generalisation accuracy by approximately 0.35–0.4 on both our CIC IDS 2018 and DetGen datasets compared to a standard model training procedure. We also note that more complicated models do not necessarily produce similar improvements. Contrasting our results with LUCID, a state-of-the-art DoS classifier which cannot be verified due to its architecture, our models are highly performant whilst being verifiable.

Classifier	Test Data	F1
Nmap Verification		
NN	**nmap (DetGen)**	**1.0000**
Cross-dataset Generalisation		
NN	DoS (CIC IDS 2018)	0.5583
NN	DoS (DetGen)	0.5622
LUCID	DoS (CIC IDS 2018)	0.5421
LUCID	DoS (DetGen)	0.5468
NN	**DoS (CIC IDS 2018)**	**0.9111**
NN	**DoS (DetGen)**	**0.9521**
Cross-attack Generalisation		
NN	SSH (CIC IDS 2018)	0.4456
NN	FTP (DetGen)	0.4914
NN	**SSH (CIC IDS 2018)**	**0.8871**
NN	**SSH (DetGen)**	**0.8059**
NN	**FTP (DetGen)**	**0.8938**

Table 6.1: F1 score of our neural network (NN) compared with the state-of-the-art. **Bold** results indicate that the model satisfies all relevant specifications whereas non-bold results indicate benchmark comparisons that are unverifiable.

Importantly, our specifications allow us to investigate complex feature interactions in a human-readable manner and ensure that our model's behaviour conforms to reasonable expectations. For instance, our `HulkAttacks` and `SlowIATsAttacks` specifications both require our *TimeElapsed* feature to be less than some bound, β_{Hulk} and $\beta_{SlowIAT}$, respectively, whilst our `SlowIATsAttacks` specification additionally constrains the IATs of a flow to be large. Maximising these β bounds, we find that this extra IAT constraint allows us to verify `SlowIATsAttacks` specifications when $\beta_{Hulk} << \beta_{SlowIAT}$, demonstrating that unusually high IATs contribute towards classifying a flow as malicious.

6.4.2 Cross-attack Generalisation

Experiment Second, we test the impact of our verification pipeline on our model's *cross-**attack** generalisation* performance. Specifically, we evaluate our model's ability to generalise to distinct but conceptually similar attacks, not present in the original training data. Due to the volumetric nature of both the *SSH-BruteForce* and *FTP-BruteForce* attacks in CIC IDS 2017 and 2018[1], we consider these attacks to have some high-level similarity to our original DoS data.

We train our model **solely** on DoS flows from CIC IDS 2017 and our specification-based adversarial DoS flows, which we write without reference to either our malicious SSH or FTP traffic. In other words, we do **not** generate adversarial traffic based on our FTP or SSH specifications. However, we still try to verify these specifications for our model.

Specifications Similar to our `ValidHTTPConn` property in Section 6.4.1, we restrict our specifications to SSH/FTP login handshakes, identified via sequences of packet sizes, and enforce that malicious FTP/SSH flows have low inter-flow arrival time:

$$\texttt{ValidLoginFTP/SSH} : \forall i \in [1,10].\, x_{\textit{pktSize-i}} = \gamma_{pktSize-i} \tag{6.19}$$

$$\texttt{NegTimeElapsed} : x_{timeElapsed} \leq \beta_{timeElapsed} \tag{6.20}$$

$$\texttt{MalLoginFTP/SSH} : \forall x.\, \texttt{ValidLoginFTP/SSH}(x) \wedge \texttt{NegTimeElapsed}(x) \implies mal \tag{6.21}$$

[1]Whilst data has been labelled as FTP-BruteForce in CIC IDS 2018, we note that the attack was launched against a closed port so is uninteresting. We omit this part of the data and rely on our *DetGen* data instead.

Results Our approach also improves cross-attack generalisation. Our model achieves a score of 0.8871 and 0.8938 on SSH and FTP BruteForce traffic respectively, when trained exclusively on DoS traffic. As we generate traffic that adheres to our specification during our adversarial training process, we expand our training data with synthetic malicious data with low *TimeElapsed* and synthetic benign data with high *TimeElapsed*. Subsequently, our model is far less likely to overfit to the DoS traffic in CIC IDS 2017, which has little diversity. As FTP and SSH BruteForce traffic can also be identified via low inter-flow arrival times, our models successfully generalise to these distinct attack classes. Furthermore, we manage to verify the specifications outlined in Section 6.4.2 **without** additional adversarial training, providing strong guarantees about detecting malicious SSH and FTP traffic.

There are downsides to this approach. We see a decrease in training accuracy, which degrades from a near perfect score, 0.99+, to between 0.88–0.96. However, we note that this initial high score is likely the result of overfitting, given its poor generalisation. There is also a considerable training overhead.

6.4.3 Generating Realisable Evasive Traffic via Counter-Examples

Experiment Next, we aim to explore model failures constructively, using them to generate interpretable examples of verifiably evasive and realisable traffic flows for particular attacks automatically. We do this by training an initial NIDS to detect SQL injection traffic via a standard train/test pipeline — without adversarial training. We then write specifications based on random malicious samples in our training data, ensuring that they are verifiable by tightly restricting feature bounds. By repeatedly querying the verifier whilst enlarging features bounds, we can produce counter-examples akin to a white-box, feature-space adversarial attack [97]. We target a model that was trained to detect SQL injection-based dumping of a mySQL database.

If performed naively, we have few guarantees about these flows, such as whether they correspond to actual traffic. To ameliorate this, we undertake this process in a systematic manner, aiming to produce counter-examples that reflect realisable evasive traffic. Our threat model assumes that an attacker can only modify their attack traffic,

thus, we only change forward packet features. Furthermore, we assume that features can only be increased — via packet delays, random padding or turning inconsequential TCP flags on. With these restrictions, we can write hundreds of specifications and test whether any suitable evasive traffic conforms to these standards.

Specifications Given an exemplar flow, we initially write a specification that corresponds to our model classifying that particular flow as malicious, setting all features to constant values:

$$\texttt{Initial} : \forall i \in [1, m].\, x_{\textit{feature-i}} = \gamma_{feature-i} \implies mal \tag{6.22}$$

As SQL injection traffic is more complicated than our DoS traffic, we parse the first 26 packets of each flow. Approximately 12 of these are attacker controlled and, since we modify packet sizes, flags and IATs, this results in roughly 36 attacker controlled features. Due to the infeasibility of performing an exhaustive grid search over all of these features, we randomly select a subset a, re-writing our specifications such that $x_{feature-i} \in a$ is bounded from below by $\gamma_{feature-i}$ and above by $\beta_{feature-i}$.

$$\begin{aligned} \texttt{Evasive} : \forall x_{feature-i} \in a.\, \gamma_{feature-i} &<= x_{\textit{feature-i}} \\ &<= \beta_{feature-i} \implies mal \end{aligned} \tag{6.23}$$

As our `PktFlag` features can only take on a small number of values, we calculate valid perturbation by assuming that the attacker can only flip inconsequential flags, namely, the ECE and CWR flags. Thus, our `PktFlag` specifications take the form:

$$\begin{aligned} \texttt{EvasiveFlags} : \forall x_{pktFlag-i} \in a.\, x_{\textit{pktFlag-i}} = (&\gamma_{pktFlag-i} \vee \gamma_{pktFlag-i} + \text{CWR} \\ &\vee \gamma_{pktFlag-i} + \text{ECE} \vee \gamma_{pktFlag-i} + \text{ECE} + \text{CWR}) \implies mal \end{aligned} \tag{6.24}$$

We consider any counter-examples produced via this process to be realisable evasive traffic. We repeat this process 50 times, randomly selecting a for each iteration, tracking the ratio of satisifiable to unsatisfiable specifications. We consider values of $|a|$ up to 16, and our upper-bounds $\beta_{feature-i}$ in increments of 0.1. To gain better insight into the effectiveness of, say, flipping TCP flags compared with padding packet sizes as evasion strategies for this specific network, we then repeat the search, increasing the bounds of only a specific class of feature at a time.

Results Our entire verification process takes roughly 12 hours per flow on consumer hardware[2]. Our model initially separated SQL injection traffic from benign flows with an F1 score of 0.81 and our analysis only holds for this specific model. However, we note that this verification-driven approach is generally applicable and provides unique insight into failure modes of a model. We successfully begin producing realisable evasive traffic for perturbations greater than 0.2, as seen in Figure 6.7.

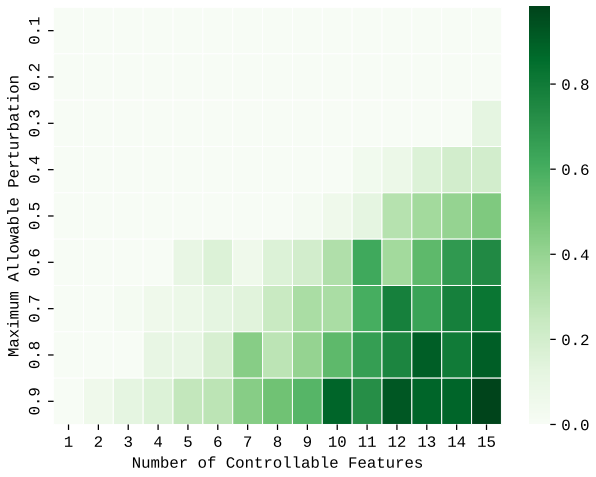


Figure 6.7: Heatmap demonstrating the percentage of specifications that produce evasive counter-examples for a given bound, when n features are attacker-controlled.

In Figure 6.8, we show the effectiveness of our three attack strategies — padding packets, delaying packets and flipping TCP flags — at different perturbation levels. Here, we see that the success of verification may vary greatly which is why the described heuristics are important. Delaying packets is the most effective strategy

[2]A laptop with an 11th Gen Intel i5-1135G7 and 16GB of RAM

by far; both delaying several packets by small amounts, or a few packets by large amounts result in realisable counter-examples. In contrast, when padding packets, our specifications require far larger bounds in order to produce counter-examples whilst exclusively flipping TCP flags fails to produce any evasive traffic whatsoever.

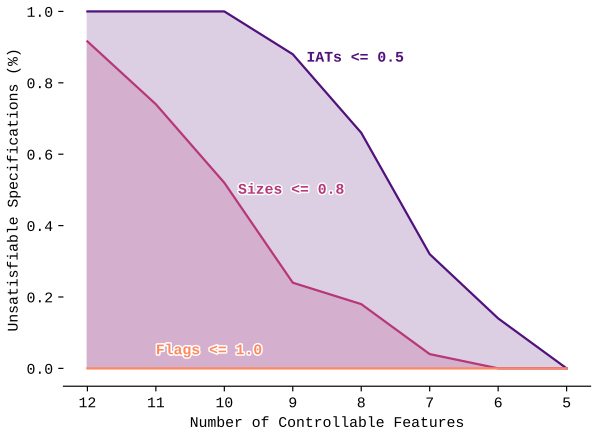


Figure 6.8: Effectiveness of our three feature-specific strategies at various levels of maximum allowed perturbation.

Although targetting specific classes of features is more effective than random features, we note that there are cases where, depending on attacker constraints, our initial general search is worthwhile. For instance, our process does not find a single evasive flow when only three IAT features are bounded from above by 0.5. However, our more general process does find such counter-examples, provided other features are also allowed to be perturbed.

Importantly, we see that the effectiveness of our evasion strategies varies widely depending on our initial seed flow, allowing us to automatically infer model failure modes with high granularity. For certain flows, delaying packets is the *only* strategy that produces counter-examples whereas, for others, alternative strategies also work.

6.4.4 Effectiveness of Local Robustness

Experiment Adversarial training is a state-of-the-art technique, typically used to strengthen local robustness and is a direct alternative to enforcing global constraints. However, there is little reason for local robustness to attenuate concept drift as our approach does. Instead, the cross-dataset generalisation performance is a consequence of enforcing **global** robustness constraints that accurately reflect the underlying structure of malicious traffic, rather than robustness in general. To demonstrate the difference, we train the NIDS to be locally robust about each input point. We use PGD with $\varepsilon = 0.1$ to train the model adversarially and verify that the model is indeed locally robust using Vehicle before evaluating its cross-dataset generalisation.

Results We successfully train a model that is verifiably locally robust about each input point. However, this provides absolutely no benefit to cross-dataset generalisation, with the locally robust model achieving an F1 score of only **0.5792** on the test dataset. As local perturbations do not adhere to the underlying structure of network traffic, the local adversarial examples contain little information about out-of-distribution inputs, unlike our global bounds.

6.4.5 Comparison with BARS

Experiment As far as we are aware, we are the first to apply deterministic neural network verification to NIDS. As a comparison, we contrast this verification approach to BARS [221], a probabilistic certified robustness methodology for NIDS based on randomised smoothing. For a given model N and input x, BARS produces targeted noise generator a G_N and robustness region r^x about x such that smoothed model $\hat{N}$ is certifiably robust when sufficient points are sampled from within r^x.

Although BARS is situated in a similar domain, its goals and approach differs from our work significantly as it focuses on local adversarial perturbations. Following from Section 6.4.4, we note that both global specifications and domain constraints are difficult to represent via $\varepsilon - \delta$ balls. Rather than supercede BARS, we intend to show how verification differs from certified robustness.

We apply BARS to a DoS NIDS, with train and test data generated according to our cross-dataset generalisation procedure in Section 6.4.1, producing certified radii for the malicious class.

Results For the globally verifiable model, we see that BARS does not correctly calculate robustness regions. As a result, the BARS robustness curve is constant for all values of ε. Investigating the noise produced by G_N, we see that features are overwhelmingly perturbed to be less than 0 and fail to change model output when clipped between 0 and 1. As a result, each r_i^x is significantly greater than 1, providing little insight into the actual robustness region of x. Whilst BARS works as intended in more general settings, in this instance, the disconnect between BARS robustness regions and the realisable values of network traffic — which are easily expressed in Vehicle specifications — impedes its performance.

6.4.6 Specification Transferability

Experiment As our counter-example driven approach requires verifying models repeatedly, we use a small model to make this feasible: each counter-example takes approximately 30 seconds. However, verifying larger models is useful. Thus, after defining the robustness regions and specifications using the small base model, we train seven additional networks of increasing size, up to approximately 2.8 million parameters. This is a similar scale to models in verification competitions [31]. As increasing model depth leads to exponentially greater verification time, we limit models to four hidden layers. We train these models according to the original robustness regions before attempting to verify the `GoodHTTP` specification, determining whether specifications and robustness regions are transferable between models of different sizes.

Table 6.2: Neural Network Parameters Versus Verification Time

Parameters	Time (s)	Verifiable?
43776	41	✓
186498	47	✓
252290	53	✓
449154	239	✓
733314	910	✓
963330	1023	✓
1750274	5661	✓
2842882	19708	✓

Results We find that the `GoodHTTP` specification transfers perfectly across all models tested. For each model architecture, we achieve perfect adversarial accuracy during training, resulting in models that adhere to `GoodHTTP` without any additional modification of robustness regions or specifications. Whilst we only have to perform this final process once, verifying larger models is more onerous; Table 6.2 demonstrates the tradeoff between model size and verification time.

6.4.7 Robustness of Global Constraints

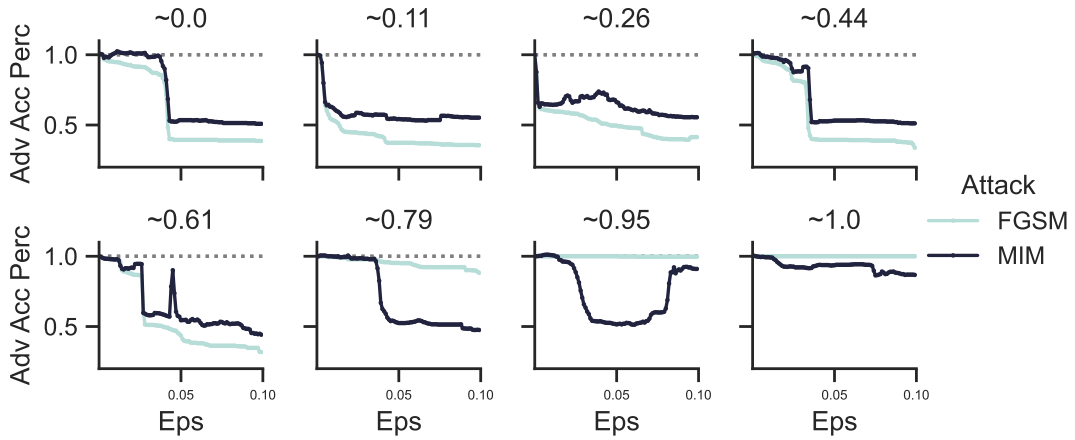


Figure 6.9: Robustness of our globally constrained models versus unconstrained adversarial attacks. Facet titles (e.g., ∼0.26) reflect the approximate accuracy that each model achieved on our constrained adversarial training data. The x-axis of each facet plots the strength of the unconstrained adversarial attack process, whereas the y-axis plots model accuracy on these unconstrained samples.

Experiment When we train our models adversarially, we generate adversarial samples based on bounds defined in our specifications, constraining features to various degrees. However, this does not imply that our models are adversarially robust in general. It is possible that our bounds are overly restrictive and that minor-but-unconstrained perturbations can still degrade model performance. If model performance degrades when naive perturbations are applied to features that we would expect to vary naturally, such as inter-arrival times or packet sizes, this suggests that our models are still overfitting to the training data, despite our attempts.

To verify that this isn't happening and to investigate the relationship between our global bounds and unconstrained perturbations, we randomly select a subset of performant models from Section 6.4.1 (each achieving an accuracy of 0.85+ on our DoS test data) and perturb our data via the Fast Gradient Sign [92] and the Momentum Iterative [66] methods. Whilst we calculate perturbations across the entire feature space, we only perturb continuous features (unlike the approach in Section 6.4.3, this does not ensure that the traffic produced is realisable).

Results We present our results in Figure 6.9. Training on adversarial samples within our specification clearly improves adversarial robustness against the fast gradient sign method, with models scoring highly on our adversarially generated data maintaining near perfect accuracy. Whilst the effect is less pronounced for the momentum iterative method, we still see benefits.

We note that Figure 6.9 violates some of aspects of the Carlini et al.'s evaluation checklist [35], such as Facet 7 increasing in accuracy for larger ε. However, similar to Section 6.4.5 this is an artefact of clipping the input data. Otherwise, the adversarial attacks behave as expected.

6.4.8 Coverage Metrics

Attack	Property	Dataset	Coverage (#)	Coverage (%)
DoS	Benign	Train	13488/189796	7.11
DoS	Benign	Test	4696/211174	2.22
DoS	Malicious	Train	190507/190663	99.92
DoS	Malicious	Test	221801/221801	100.00
DoS	Malicious*	Train	125087/190663	65.61
DoS	Malicious*	Test	183727/221801	82.83
SSH	Benign	Train	748/6876	10.88
SSH	Benign	Test	123/102624	0.12
SSH	Malicious	Train	6964/6964	100.00
SSH	Malicious	Test	108639/108639	100.00
SSH	Malicious*	Train	2851/6964	40.94
SSH	Malicious*	Test	62402/108639	57.44

Table 6.3: Generalisability of our specifications. Note that 'Malicious*' is calculated exclusively on the attacks' specifications that are verified.

Experiment Specifications for DNN verification are usually assumed to be correct. In contrast, in this work, since our specifications are manually engineered, we go a step further and validate them via coverage metrics. Specifically, we check the percentage of flows in the training/test data that satisfy our specification bounds.

Results Table 6.3 reports the results of our validation checks. Notably, our complete specifications for malicious traffic cover $\approx$100% of the malicious traffic from the datasets. Furthermore, while decreasing, our verifiable specifications still cover a high percentage of the datasets (from 40.94% to 82.83%). Lastly, while the coverage of the benign traffic is comparatively lower (from 0.12% to 10.88%), it is an encouraging result as our specifications target only subsets of the benign data, such as SSH traffic, which make up a comparatively small percentage of the dataset. These results confirm the validity of our specifications and that they reflect actual traffic in our datasets.

6.5 Limitations

Our approach has several limitations. Current verification frameworks prevent us from verifying deep networks or complex architectures, with models in state-of-the-art verification competitions limited to approximately 12 million parameters [31]. Our process also requires repeatedly querying models to produce counter-examples, limiting practical model size even further. Despite this limitation, however, our method achieves better results than deeper approaches, such as LUCID, even with our simple model. Verification frameworks are still actively researched and iteratively improved [229], potentially relaxing this limitation in the future.

Section 6.3 mentioned that increasing detail of specifications can lead to the Marabou verifier rapidly exhausting resources, so specifications that carve up the input space too finely become impractical. To account for this, for particularly onerous features, we consider only commonly occurring combinations in the training data, as discussed in Section 6.3.3. In any case, the human-in-the-loop effort also limits how complex we would want specifications to become. There is a similarity with the use of human-written signatures for IDS which have limits on scope and complexity. Ultimately, because there can be no complete specification of malicious traffic, there is a trade-off between what we want the model to learn and the global constraints we want to specify.

6.6 Related Work

To the best of our knowledge, while research exists on security applications of DNN verification [21] and on probabilistic verification for DNN-based NIDS [221], no prior work has explored the use of verification frameworks with machine learning-based NIDS.

ML-based NIDS are a well-trodden topic with many competing architectures. Often, classifiers make significant modifications to 'off-the-shelf' models, including CNN-based [69, 223], autoencoder-based [149] and graph-based [219] approaches. These architectures are too complex for current complete verification methods, making behavioural guarantees difficult.

Much work exists on robustness and adversarial examples for NIDS. Zhang et al. [240] provide a thorough overview of attacks and defenses applied to NIDS, including the HopSkipJump attack [41] and ensemble adversarial training [216].

NIDS models are often trained and tested on the same public dataset, but unfortunately many commonly used datasets have flaws [73, 81]. Even simple models with restricted feature sets can be highly performant on common datasets, as demonstrated by Jacobs et al. [107]. But these performant models fail to generalise to other datasets or attacks when withheld as test sets [14, 38]. We aim to improve generalisation via a specification-driven approach.

Existing research on DNN verification primarily focuses on local robustness [37] in computer vision [161] . In contrast, global properties are defined over regions of space not parameterised by inputs [111], making them more general and challenging to prove. Altogether, global properties are less commonly considered in research, with some exceptions. Katz et al. [112] define global robustness specifications for ACAS Xu [110]. Chen et al. [43] also verify global properties for security classifiers. However, unlike our work, their global properties are unrelated to the underlying mechanism of attacks and only bound model outputs, rather than inputs.

6.7 Conclusion

To our knowledge, we are the first to investigate the applications of neural network verification in the network intrusion domain. We do this non-trivially, aiming to improving adversarial robustness, cross-dataset generalisation as well as uncover model weaknesses and failure modes. To do this, we design our models, feature sets and verification criteria from the ground up to minimise the effects of low data diversity via targeted restriction and expansion of our input data. We developed models

with considerably improved cross-dataset and cross-attack generalisation compared to standard approaches, whilst gaining strong mathematical guarantees about our models' behaviour in regions of the input space. Our specifications allow us to reason about the complex feature interactions of our models, thanks to choosing a tractable feature set and using the high-level specification language Vehicle. Our work provides insight into how verification frameworks alongside data generation techniques can improve model behaviour.

Neural network verification is an emerging area of research and there are many possible avenues for future work, using other tools, as well as exploring local specifications, considering more complex attacks and generalisation conditions. More study is needed to understand the trade-off between adversarial training, model accuracy and verifiability.

Chapter 7

Generating Traffic-Level Adversarial Examples from Feature-Level Specifications

Thesis Context: *This chapter furthers research into constructively applying neural network verification to the NIDS domain to test model robustness. Specifically, by combining verification with synthetic data generation, we produce valid traffic with near-arbitrary perturbations, extending prior work that constructs valid traffic for only a limited set of perturbations.*

7.1 Introduction

A common problem with adversarial example generation is that adversarial methods usually operate on the feature space rather than the problem space. In, say, the image domain, converting between these is simple, however, in other settings, features are derived from the problem-space in a manner that is difficult to reverse. The gap between the two means that feature space adversarial examples may not correspond to realistic, or even valid, adversarial examples in the problem space. A recent survey has highlighted this as a failing in many adversarial methods proposed for NIDS [100].

In this paper, we introduce a novel solution to bridge the gap for NIDS. The problem space is network traffic (captured in PCAP files) and the feature space is network flow summaries, mixing information from packets such as IP addresses, port ranges, flags with statistical measurements such as sizes and interarrival times (IATs), as generated by CICFlowMeter [1]. Using a state-of-the-art machine learning approach such as a

neural network, adversarial methods can find examples which cause misclassifications. These examples represent serious security vulnerabilities such as structural failings in the IDS design or data processing pipeline (for example, inability to generalise to delayed or out-of-order traffic).

However, because adversarial techniques usually operate in the feature space there is no general way to construct corresponding network flows which can be tested in an overall NIDS pipeline to see if there are actual, realistic misclassified inputs that need to be addressed. Our solution is based around the combination of two tools with an automated pipeline, shown in Fig. 7.1:

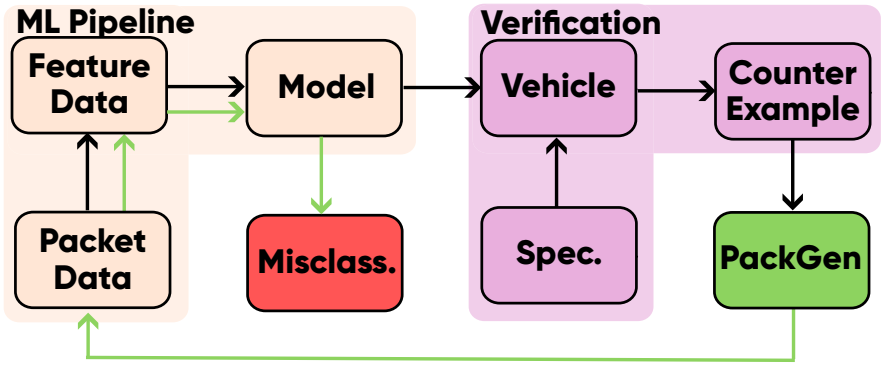


Figure 7.1: High-level view of our evasion pipeline

- **PackGen**, a specification-driven traffic generator. PackGen learns a compact Markov chain based summary for packets and sequences of packets given an input PCAP file. Changing the specification of the summary by adding or modifying constraints allows PackGen to re-generate perturbed sequences of packets which have a close connection to the original sequences.

- **Vehicle** [55, 56], a functional language for writing formal specifications for neural network verification algorithms, primarily Marabou [229]. Vehicle provides a compact way to describe logical constraints which generalise input training data, for example, allowing packet sizes or inter-arrival times to vary beyond those seen in the training data. When verification fails, Vehicle generates counterexamples, representing input data points that would be misclassified. We refer to Vehicle outputs as 'counterexamples' and describe the corresponding model inputs as 'adversarial' or 'evasive'.

- **Automated pipeline**: As usual, labelled training data is pre-processed to generate network flow summaries and then input features which are used to train a neural network. The network aims to classify network traffic into malicious or benign traffic flows. Attempting to verify the network with Vehicle against some robustness specifications as suggested produces counterexamples when the network is not robust. The counterexample is post-processed and matched against the training flow which has been summarised by PackGen. Then it is used to modify the PackGen summary specification accordingly, which in turn generates modified raw PCAP files, completing the loop back to the input format and bridging the problem-feature space divide and allows us to test for misclassification.

This work was originally published at the SECAI workshop, co-located with ESORICS as "*Generating Traffic-Level Adversarial Examples from Feature-Level Specifications*" [80]. Due to page limits, the paper described PackGen at only a high-level, excluding detailed information about the possible packet transformations. Furthermore, a description of the tooling used to convert counterexamples into PackGen configuration files was omitted entirely. For clarity, this chapter extends the original paper with these details in Section 7.3.

Overview. Following the related work (Section 7.2), the remainder of this chapter details the two tools, PackGen and Vehicle (Sections 7.3 and 7.4), their application to this problem and our initial experiments (Section 7.5). Section 7.6 discusses the limitations and conclusions.

7.2 Related Work

Network Intrusion Detection Due to their security-sensitive nature, NIDS are a natural candidate for adversarial perturbations and there is much work in this area [50, 106, 169, 180]. To tackle the problem/feature gap, many papers introduce domain constraints to ensure that any feature-level perturbations are realisable, such as the work of Sheatsley et al. [190] and Zolbayar et al. [243] . Others, such as Wang et al. [220] and Sharon et al. [189], operate directly on the packet-level, using packet modifications to alter traffic but only in a limited capacity.

Although constrained attacks produce features that are *plausible*, as far as we are aware, no traffic generation framework can easily produce flows that reflect these features. DetGen [49], our own deterministic data generation framework that requires users to write scripted scenarios, of which many aspects are difficult to control precisely. Similarly, ID2T [51] requires users to manually write packet contents and specify timing patterns, severely limiting its generalisability.

Neural Network Verification Neural networks are widely recognised for their lack of robustness [92, 207]. Among the various approaches to address this issue is the field of neural network verification, which draws from formal methods. Recently, several neural network verification frameworks have emerged [20, 22, 132, 198, 230]. The syntax for writing specifications varies across different verifiers, and attempts to create a standard [59] have not yet resulted in an intuitive and easily understandable solution for complex properties.

Furthermore, due to scalability and tightness challenges, neural networks trained via standard pipelines are unlikely to satisfy meaningful input and output constraints. Instead, to satisfy non-trivial properties, networks are trained for robustness. After robust training, models often achieve higher verification success and are more likely to satisfy the desired properties. Robust training techniques can be grouped into three

main categories: (1) data augmentation, where new data is created by manipulating existing data points; (2) adversarial training, which involves adding the worst-performing counterexamples within a bounded region to the training set; and (3) certified training, which provides mathematical guarantees about model behaviour within certain bounds.

In this work, we employ a PGD-based algorithm [142], specifically customised to work with our specifications, for both adversarial training and generating counterexamples. We then compare these counterexamples with those generated by Vehicle.

7.3 PackGen

PackGen is a commandline tool written in Go. It consists of a *Packet Reader*, which converts packets into a truncated summary, a *Chain Builder*, which converts these into a Markov chain, and a *Packet Writer*, which produces plausible traffic via this Markov chain. Although we could manipulate PCAPs directly, this representation provides us with succinct way to describe transformations when modifying traffic.

The primary aim of PackGen is to generate modified, synthetic traffic based on a set of initial flows. Many data generation testbeds or frameworks, such as DetGen [49] or Caldera [90], run automated scripts within a virtualised environment to generate traffic. When generating a large coverage of an attack's potential traces, such as, say, nmap traffic with random padding, this process can be extremely slow as the actual attacks have to be executed to completion. By using PackGen's condensed chain, we can produce several thousand packets a second, independent of flow durations, and, by manipulating the states and transitions, we can alter the properties of the traffic we produce en masse. For instance, for the above nmap problem, we can simply randomise packet lengths.

Packet Reader Given a PCAP file, the *Packet Reader* processes each flow independently. The reader then iterates over each flow and converts each packet into a *PacketRepresentation* data structure. Each *PacketRepresentation* contains a packet's *source/destination IP addresses*, *source/destination ports*, *direction*, *protocol*, *capture length*, *header length*, *inter-arrival time*, *TCP flags*, *TCP options length* and *padding length*. Depending on the use case, PackGen can either save payloads for use by the *Packet Writer* or discard them (reconstructing packets using randomly generated payloads).

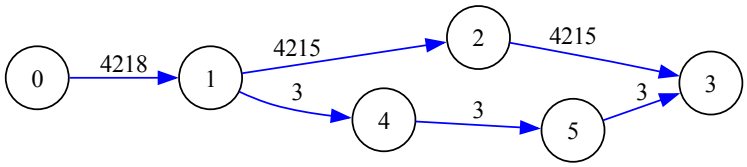


Figure 7.2: An exemplar Markov Chain representing nmap traffic. The upper path and lower paths reflect closed and open ports, respectively.

Chain Builder After reading a PCAP, we have truncated representations of each flow. The *Chain Builder* iterates over these, converting each representation into a state in the chain and tallying the transitions between each state. A start state and an absorbing state are prepended/appended to each flow. PackGen can build chains of any order. Higher order chains better model the input data but can lead to state explosion.

To help prevent blow-up, we only use a subset of each *PacketRepresentation* to define states, namely, *direction*, *bucketed capture length*, *protocol*, *bucketed IAT*, *TCP flags*, *TCP options length* and *packet padding length*, a tuple which we call the *StateRepresentation*. Although *StateRepresentation* does not contain all of the information we originally extract from the PCAP file, we store statistical representations of this additional information in a *StatsArray*, including timing and size statistics, alongside the chain to reconstruct plausible flows.

The *Chain Builder* outputs a JSON file defining the states and transition probabilities. This has several advantages over the original PCAP files. First, the file is much smaller. For instance, PackGen condenses 50000 DDoS packets to $\sim$25Kb. Secondly, sometimes, infrequent flows represent noise in the data which we can remove by discarding rare states and transitions. Finally, we can alter packet properties en masse via dictionary updates, allowing us to quickly produce many chains representing modified versions of the original PCAP.

Packet Writer To produce traffic, our *Packet Writer* randomly samples a path in the Markov chain until reaching an absorbing state. As multiple flows are compressed into a single representation, this path may contain many disparate flows before finishing. Given a list of states, we then reverse the initial packet reading process, converting our *StateRepresentations* to *PacketRepresentations*, using the size and timing distributions of our *StatsArray* to regain missing values. Payloads can then be retrieved or randomly generated. Finally, we reconstruct packets using default values which are inconsequential in our NIDS as they are not reflected in our feature values, such as IP flags or MAC addresses. If needed in other contexts, these can also be stored in and read from the *StatsArray*.

7.3.1 Perturbed PackGen Specifications with Lenses

For completeness, we briefly describe our tooling, written in Haskell, for converting Vehicle counter-examples into valid PackGen specifications below. This was omitted from the original, published version of this work due to space constraints.

Due to the intricate structure of our JSON format, modifying deeply nested objects to generate perturbed traffic was challenging and potentially error-prone, such as ensuring that flags were mapped to valid values. To simplify this task and ensure precision, we used *lenses* from functional programming. Lenses facilitate precise and predictable modifications of data structures, similar to getters and setters from object-oriented programming, governed by a set of *lens laws* that dictate their behavior. Given some structure S and values v, w, these are:

$$\text{Get-Set Law:} \quad \text{set}(\text{get}(S)) = S$$

$$\text{Set-Get Law:} \quad \text{get}(\text{set}(S, v)) = v$$

$$\text{Set-Set Law:} \quad \text{set}(\text{set}(S, v), w) = \text{set}(S, w)$$

These laws ensure that lenses are composable. Thus, by composing lens a that accesses data structure $\mathscr{A}$, with lens b that accesses sub-part $\mathscr{B}$, we form lens $c = b.a$. This mechanism allows easy and safe manipulation of the nested data without unwanted effects on $\mathscr{A}$. Lens frameworks can automatically generate a full set of lenses for a given data structure via tools such as Template Haskell, improving both the robustness and maintainability of the data manipulation process. Importantly, unlike simple getters and setters, lens operators enable additional modification of features beyond simple retrieval and replacement, allowing us to incorporate any necessary feature pre-processing. This allows us to modify dependent features such as '*Mean Packet Size*' and '*Standard Deviation of Packet Sizes*' simultaneously in a robust manner.

Via this approach, any element of PackGen's *StateRepresentation* or *StatsArray* structures with a well-defined lens can be manipulated. Furthermore, as our feature set, discussed in Section 7.5.2, closely reflects our choice of *StateRepresentation*, we can easily parse the corresponding counterexample, mapping each feature value to its appropriate lens. By parsing our input specification and sequentially applying our lenses, this process outputs a perturbed PackGen specification. While this enables easy extension of modifiable packet features, PackGen is limited to manipulating attributes with associated lenses and reconstruction methods in the Packet Writer.

7.4 Vehicle

We provide a brief introduction to Vehicle [55, 56], a high-level functional language for writing precise and expressive specifications of neural network properties. Vehicle's syntax consists of simple functions, quantifiers, arithmetic and logical operators. Listing 7.3, contains a simplified version of the specifications used in Section 7.5.2. This specification enforces that a model must be robust when the size of a flow's first packet can vary slightly. It can be read as follows:

```
@network
classifier : InputVector -> OutputVector

advises : InputVector -> Label -> Bool
advises x i = forall j . j != i =>
    classifier x ! i > classifier x ! j

minSIZE = 40
maxSIZE = 80

changePktSize1 : InputVector -> InputVector -> Bool
changePktSize1 malFlow x =
  x ! pktIAT1 == malFlow ! pktIAT1 and
  ...
  x ! pktIAT10 == malFlow ! pktIAT10 and
  minSIZE <= x ! pktSize1 <= maxSIZE and
  x ! pktSize2 == malFlow ! pktSize2
  ...

robustPktSize1 : InputVector -> Bool
robustPktSize1 malFlow = forall x . changePktSize1 malFlow x =>
    advises x mal

@property
property : Vector Bool n
property = foreach i . robustPktSize1 (trainingInputs ! i)
```

Figure 7.3: An exemplar Vehicle specification, stating that a model must be robust when the size of the first packet is allowed to vary within a specified range.

First, we define our classifier using the `@network` annotation, which indicates it will be set at compile time. Then, we define a helper function `advises` which enforces that the model must output a particular class for a specific input. Following this, we define our packet size bounds. For legibility, we also define constants corresponding to feature indices and the `mal` synonym for the malicious label. The function `changePktSize1` takes two vectors as input, `malFlow`, a malicious flow from our input dataset, and `x`, an arbitrary vector satisfying the inequality in the body of the function. This enforces that `x` must equal `malFlow` everywhere except `pktSize1`, which can vary between `minSize` and `maxSize`. `robustPktSize1` is our robustness condition, which states that all such `x` must be classified as malicious. Finally, we define the `property` that Vehicle ultimately verifies: all flows in our input data must adhere to `robustPktSize1`. If false, Vehicle outputs a counterexample.

Although attacks such as PGD [142] can be limited to certain regions of input space, complicated specifications quickly become unwieldy and difficult to debug, requiring users to define constraints as a series of feature vectors. In comparison, Vehicle is readable and flexible whilst providing stronger robustness guarantees.

7.5 Evaluation

7.5.1 Using PackGen to reconstruct network flows

We check that PackGen correctly generates reasonable approximations of its input data, matching our needs. For a test dataset, we use DetGen [49] to launch a denial of service attack on an Apache webserver, generating approximately 4000 flows. Whilst this is a volumetric attack, it is not trivial to model as there are many artefacts, such as failed handshakes, high packet delays and aborted flows. We use PackGen to train a Markov chain of order 3 on this data and generate an additional 4000 flows, seen in Figure 7.4. To evaluate PackGen's accuracy, we then contrast the timing and size statistics for these two datasets.

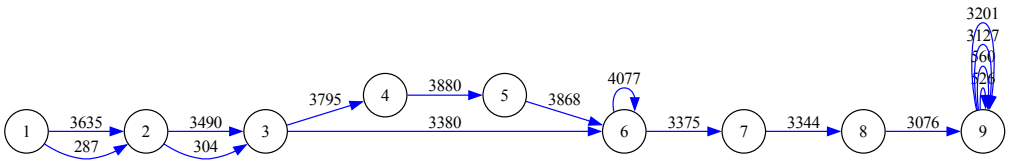


Figure 7.4: Our PackGen chain for DoS traffic. For readability, we have omitted connections with a less than 6% chance of occurring. The full chain we use to generate traffic is considerably less deterministic.

Table 7.1: Comparison of feature distributions between Original and Generated data.

Feature	p-value	μ (Orig/Gen)	σ (Orig/Gen)
Tot Fwd Pkts	0.97	5.99/6.01	2.33/2.38
Tot Fwd Vol	0.82	1357/1409	3153/3686
Tot Bwd Pkts	0.97	6.14/6.19	2.88/3.05
Tot Bwd Vol	0.7	18516/18617	7216/7926
Fwd Len Mean	0.91	221/214	383/362
Fwd Len Std	0.97	169/159	278/231
Bwd Len Mean	0.47	3034/2997	973/921
Bwd Len Std	0.47	3016/3041	1136/1024

PackGen summarises transition timings by their means and standard deviations. For transition from state i to j, we write these as $\mu_{i\to j}$ and $\sigma_{i\to j}$. PackGen then models IATs via distributions estimated using these values. We will compare the efficacy of uniform, Gaussian, exponential ($\lambda = \frac{1}{\mu_{i\to j}}$) and gamma ($\alpha = (\frac{\mu_{i\to j}}{\sigma_{i\to j}})^2$, $\beta = (\frac{\mu_{i\to j}}{\sigma^2_{i\to j}})$) distributions in matching the original data.

Results In Table 7.1, we compare the feature distributions of our input and generated data using the Kolmogorov–Smirnov test [200] as well as their respective mean and standard deviations. As *StatsArray* stores highly granular information about packet sizes, flows generated by PackGen have extremely similar size characteristics to the input dataset. For instance, the p-value of the *Tot Fwd Pkts* feature indicates that these distributions are certainly identical. Although some features score lower, such as *Bwd Len Mean*, we note that this appears to be an artefact caused by minor differences between distributions with high standard deviations.

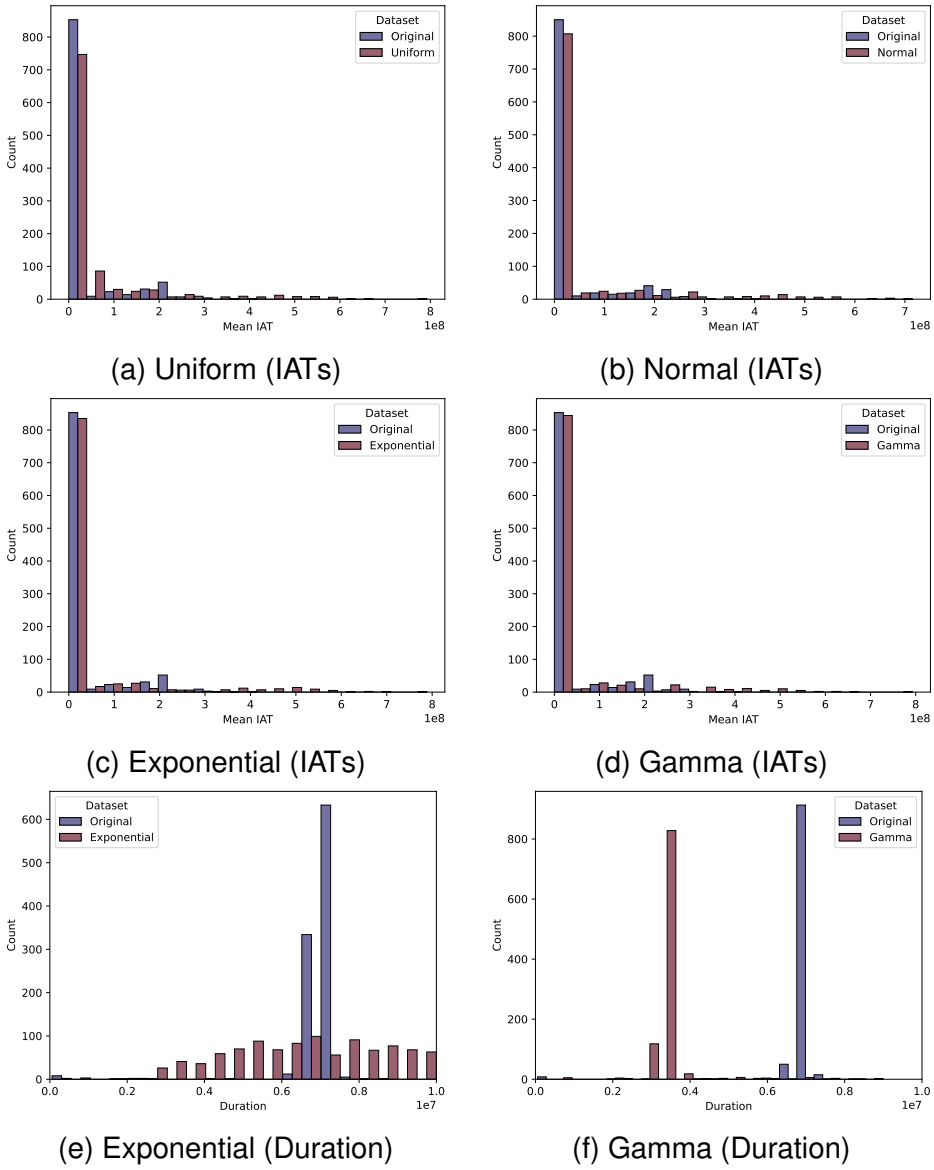
As seen in Figures 7.5 (a)–(d), except for the uniform model, all distributions duplicate the extreme skew of the original IATs. However, as in Figures 7.5 (e)–(f), we see that flow durations vary considerably. Although left-shifted, the gamma model is most comparable to the original data. We quantify this by normalising the distributions by their means and calculating the Wasserstein distance [65] between the original and generated durations (lower is better). The gamma model scores best by far followed by the exponential model at **0.056** and **0.233** respectively. The discrepancy between the gamma model and the real data — a difference of approximately 0.3 seconds — is caused by our wide time buckets to prevent state explosion biasing longer IATs.

7.5.2 Specification-based Adversarial Perturbations

To demonstrate PackGen's ability to produce traffic with targeted perturbations, we use it alongside Vehicle to generate evasive traffic according to feature-level perturbation strategies, as in Figure 7.1. Our feature set is straightforward: we extract the directions, IATs, sizes and TCP flags of the first 10 packets of a flow, as well as the protocol and the time since the previous flow with the same hosts. We train our models to detect the DoS traffic from Section 7.5.1.

We consider three attacker models encoded as Vehicle specifications by bounding features in the same manner as Listing 7.3. We define three attackers based on the features they can modify:

- Attacker 1: Packet *sizes*
- Attacker 2: Packet *times*
- Attacker 3: Packet *sizes and times*



(a) Uniform (IATs) (b) Normal (IATs)

(c) Exponential (IATs) (d) Gamma (IATs)

(e) Exponential (Duration) (f) Gamma (Duration)

Figure 7.5: Histograms comparing the timing characteristics of Original data versus our PackGen data depending on distribution choice. We omit Duration histograms for the Uniform and Normal distributions as they did not overlap with the Original data.

We enforce that attackers can only modify packets in the forward direction and select feature bounds such that attackers can perturb packet sizes by approximately 20% and delay packets by approximately one second. We compare the effectiveness of this process to PGD [142], a state-of-the-art adversarial attack. We use two networks consisting of one layer of 128 nodes: a *base* network which has been trained normally and a *robust* network, which has been trained adversarially.

Table 7.2: Attacks are replayable (✓) if we successfully create evasive PackGen flows.

Network	Strategy	Adv. Acc. (PGD/Vehicle)	Replayable?
Base	Sizes	1.000/1.000	✓
Robust	Sizes	0.8350/0.8350	✓
Base	Times	1.0000/1.0000	✓
Robust	Times	0.8400/0.8550	✓
Base	Both	1.0000/1.0000	✓
Robust	Both	0.8550/0.8550	✓

Results Despite our restrictive threat models, we find adversarial examples for all networks. Furthermore, PackGen successfully recreates PCAP files containing flows with the same characteristics as the input counterexample, allowing these attacks to be replayed or incorporated into standard NIDS pipelines for testing.

Based on our results, PGD and Vehicle are both proficient at finding adversarial/-counterexamples. However, Vehicle expresses threat models and attacker limitations using an easy-to-understand DSL. We also note that PGD is an approximate method and is not guaranteed to find evasive flows if they exist whereas Vehicle is exhaustive, leading to better performance for Attacker 2 (modifying packet times). We present an intuitive explanation as to why this happens in Figure 7.6: PGD perturbs features downwards in the direction where the gradient is steepest. However, when considering bounded feature perturbations, the worst-case minima that PGD is “trying to reach" may be beyond the specified limits. Thus, PGD effectively ignores other potential counterexamples that Vehicle can uncover. We note that, from a security perspective, there is little difference between an arbitrary and a worst-case counterexample. Therefore, the exhaustive method of verification-based counterexample generation provides stronger security guarantees.

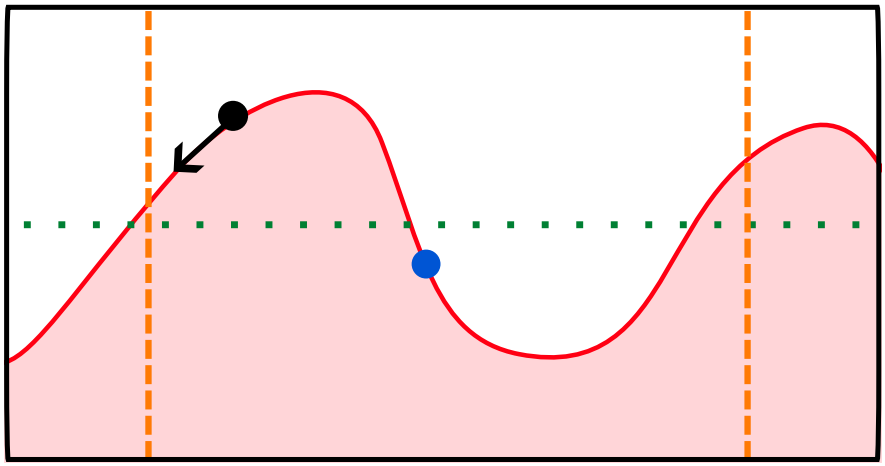


Figure 7.6: PGD encourages the input (●) to follow the steepest gradient downwards, effectively ignoring the possible counterexample found by Vehicle (●). The green dotted line represents the model decision boundary while the orange dotted lines represent feature limits. Note that Vehicle does not necessarily find the counterexample at a local minima.

7.6 Conclusion

We introduced PackGen, a specification-driven traffic generator and, in conjunction with Vehicle, used it to produce replayable adversarial attacks on NIDS. We demonstrated that the Vehicle specification language can flexibly define problem-space constraints and threat models in an easy-to-interpret manner.

PackGen has several limitations compared with other packet generation frameworks. It works best when trained on a dataset where each flow has some underlying similarity to generate focused data, for instance, a series of HTTP connections. Furthermore, our feature set allowed us to directly parse counterexamples to modify the corresponding Markov chain. Producing traffic for more complex features where this cannot be done is left to future work. Finally, finding counterexamples using Vehicle is considerably slower than attacks such as PGD and does not scale to very large networks; the trade off is the greater versatility, readability and stronger guarantees of our specification-based approach. Hybrid techniques may provide greater balance between these downsides.

Chapter 8

Conclusion

This thesis explored techniques to improve trust in ML-based NIDS via two strands of research: (1) scrutinising NIDS benchmark datasets and (2) neural network verification. In Chapters 3 and 4 we built a framework for analysing, critiquing and quantifying bad data design smells that are suggestive of issues with the underlying data generation process, identifying six issues that are general in the most popular, recent benchmark datasets. For each of these, we provided a corresponding heuristic measure to quantify their prevalence, demonstrating numerically that, for example, 64% of classes in CIC IDS 2018 suffer from profoundly repetitive traffic. We apply these measures to other domains and see that issues are far less extreme, indicating that these issues are particular to NIDS datasets. Critically, via a involved literature overview and four in-depth case studies, we also highlight how these bad data design smells impact downstream research and how unjustified assumptions regarding NIDS data quality are endemic. We continued this in Chapter 5, designing a metric to place the input complexity of NIDS datasets in a wider context, juxtaposing them with both established benchmark image datasets and sources of real-world traffic. Altogether, these chapters present a cohesive methodology for interrogating NIDS datasets, allowing researchers and practitioners to more readily pinpoint potential problems and limitations of their data.

In Chapter 6 we change tack, presenting — to our knowledge — the first application of neural network verification techniques to the NIDS domain to improve model inference along a number of axes such as stronger generalisation guarantees — improving cross-dataset generalisation accuracy by 35% — better explainability and adversarial robustness. We do this by enforcing that models adhere to global domain constraints reflecting desirable properties, such as associating volumetric attacks with repetitive

traffic. We also demonstrate how to constructively use verifier counterexamples to rank threat models by severity according to feature fragility. Finally, in Chapter 7, we further exploit these counterexamples to close the problem/feature space gap and generate evasive, feature-space traffic by developing a bespoke traffic generation framework. Not only does our technique produce raw PCAP traffic that can be tested against a full NIDS pipeline, we demonstrate the advantages of verification-based counterexamples by outperforming PGD, a state-of-the-art gradient-based adversarial attack.

Trusting Datasets Despite a substantial body of research, there are still fundamental questions about dataset design for network security. In contrast to fields such as computer vision and natural language processing, best practices for dataset creation and feature extraction are comparatively underdeveloped, resulting in the poor data standards highlighted in this thesis, and degrading trust in results on models tested on them. Achieving parity with more mature ML domains remains a difficult task. This thesis provides an initial foundation for mitigating these issues by systematising the analysis of benchmark NIDS datasets, surfacing key flaws, and proposing recommended practices. Critically, we emphasise the importance of thorough dataset analysis in NIDS design. Researchers should justify features and model architectures used by directing relating choices to attack properties and structure.

Trusting Models Interpretability and transparency are vital for trustworthy network security models. By exploring how verification can enforce domain constraints, this thesis enables security practitioners to identify model blind spots and improve robustness. Similarly, as our specification-driven training, in effect, generates additional, specification-compliant data, well-defined constraints are also a tool to resolve dataset shortcomings, indirectly improving dataset trust.

Our two-pronged approach to dataset quality and formal specification offers a blueprint for more reliable and confident deployment of ML-based NIDS. It ensures that security-critical decisions are driven by models whose strengths and vulnerabilities are both well-understood and systematically addressed, narrowing the gap between research and its tangible application in protecting networks.

Moreover, we emphasise that all neural networks used in this thesis are shallow and straightforward, yet they consistently surpass the performance of state-of-the-art research models. This success stems from shifting the research focus away from engineering complex architectures. Instead, we prioritise good data practices and enhancing performance by augmenting data via complementary tooling, such as Marabou. We argue that this outcome strongly indicates that data quality is a critical limiting factor in NIDS research.

8.1 Future Work

There are several compelling directions for future work, some of which we outline here.

Improved Rigour for NIDS Features Currently, there is little consensus on NIDS feature selection and aggregated flow-based features have remained the de facto standard since KDD Cup 1999. In contrast, other machine learning applications to security have seen an iterative progression of features. For instance, early ML-based software vulnerability detection used simple *n*-gram features [178] whereas more recent research relies on complex embeddings equipped with a graph structure [242] to better reflect the program's control flow graph. This thesis explored the advantages of alternative features which, for instance, enabled our specifications in Section 6. We also show the shortcomings of naive application of flow statistics in Section 4.4.

Automated Specification Generation In Chapter 6, we required domain knowledge and a human-in-the-loop to write meaningful specifications. This introduces a source of bias that could be negated by developing an automated process that extracts sensible specifications from training data. Neural network certification may provide a starting point for this process, as prior work shows that maximal certification boundaries can be automatically learned from data [221]. By narrowing these bounds via counter-example generation, we might be able to convert these 'certified' bounds to 'verified' bounds in a fairly straightforward manner.

Application to Real-World Settings Throughout this thesis, we relied extensively on synthetically generated data produced using the emulation-based DetGen framework [49] which, as highlighted in Chapter 3, can offer limited research value. We exercised considerable caution when using DetGen by varying different aspects of the generation process to maintain realistic traffic patterns. To minimise data leakage, we also segmented the training/test splits of our datasets according to these variations. Despite these precautions, our experiments were conducted in a laboratory setting, meaning that certain real-world concerns, such as the appropriate ratio of attacks to benign traffic, remain unaddressed. Consequently, while our synthetic framework enables controlled experimentation, the results may not fully translate to unpredictable real-world environments. Indeed, a real-world traffic monitor would likely only have a partial view of most flows, depending on how packets are routed through the network. A comprehensive security overview of a network would require multiple capture sinks working in tandem to reconstruct flows. In this setting, the applicability of almost all research NIDS is unknown and solving this issue requires further study into flow reconstruction from multiple network vantage points, learning methods that can handle incomplete or noisy traffic data, and scalable feature extraction techniques for real-world deployment.

8.2 Future Outlook

I believe concrete improvements to both NIDS data and models are possible and that this thesis presents a research path forward. First and foremost, strengthening NIDS dataset quality is no doubt a good thing and will form a fundamental aspect of deploying NIDS to real-world environments. Better data generation frameworks and better NIDS data measurements are also vital, allowing researchers to more concretely reason about model training and inference. For instance, practitioners may use bespoke metrics to finetune training data systematically, or data emulators to produce arbitrary test sets to fully catalogue false positive rates across a spectrum of deployment scenarios. Crucially, a data-driven focus lays the foundation for novel model architectures and detection methods, enhancing real-world trust and reliability. I am confident that shifting research focus to these neglected aspects of NIDS can lead to major strides and potentially revitalise the field.

Appendix A

Overview of Surveyed Papers in Chapter 3

For our paper overview selection, we employed the following criteria:

- The paper uses at least one of the 7 datasets that we have covered here. Note that we looked through papers that cited the dataset's official paper, potentially missing incorrect citations.
- The aim of the paper is to detect or classify malicious activity found in one of the 7 datasets, or perform some kind of adversarial attack (e.g. adversarial examples, evasion attacks) against a model trained on one of the 7 datasets. For this reason, we excluded systematisations of knowledge [13, 15, 154] and papers that criticise other aspects of Machine Learning approaches for Network Intrusion Detection [9, 72]
- The paper was published at one of the venues listed in Table A.1. As mentioned at the beginning of Section 3.3, we expanded our initial list of 7 top security target venues to increase dataset coverage.

We evaluated dataset(s) usage as follows:

- *Feature Variation (FV)* - We marked the assumption as *present* if the paper does not mention any discussion or analysis of the variability (or lack thereof) of features in a class. This also extends to discussion or analysis on whether the distribution of features within a certain malicious class is, due to a lack of intra-class variation, significantly different from that of benign traffic, rendering the classification task trivial.

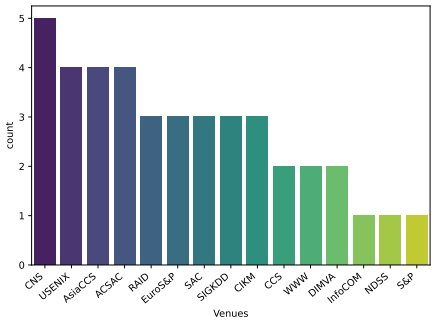
Acronym	Venue Full Name
USENIX	USENIX Security Symposium
S&P	IEEE Symposium on Security and Privacy
CCS	ACM SIGSAC conference on computer and communications security
NDSS	Network and Distributed System Security Symposium
EuroS&P	IEEE European symposium on security and privacy
ACSAC	Annual Computer Security Applications Conference
AsiaCCS	ACM Asia conference on computer and communications security
SAC	ACM/SIGAPP Symposium on Applied Computing
CNS	IEEE Conference on Communications and Network Security
DIMVA	Detection of Intrusions and Malware, and Vulnerability Assessment
InfoCOM	IEEE International Conference on Computer Communications
WWW	World Wide Web Conference
CIKM	Conference on Information and Knowledge Management
KDD	ACM SIGKDD Conference on Knowledge Discovery and Data Mining

Table A.1: List of venues considered for our selection of papers that use one of the 7 NIDS datasets treated in this work.

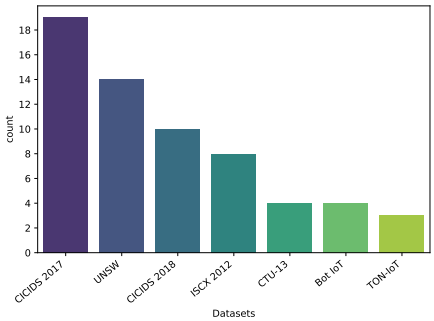
- *Attack Variation (AV)* - We marked the assumption as *present* if the paper does not mention any discussion or analysis of the number of distinct interactions in a class, whether the attack was repetitive in nature, or whether the attack was set up in a very simplistic way.

- *Highly Dependent Features (HDF)* - We marked this category as *present* if the paper does not perform any *semantic* post-hoc analysis on the most important features according to their trained model, namely whether these features are realistically able to characterise an attack or whether the feature's value is merely spuriously correlated to a certain attack. In order to mark this category as *not present*, we required that the paper's semantic analysis is built upon expert knowledge, which usually necessitates some level of manual PCAP analysis.

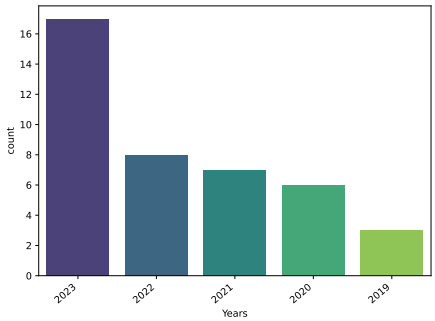
- *Wrong labels or Unclear Ground Truth (W/U)* - We marked the assumption as *present* if papers accepted the dataset's labels without questioning the validity of the ground truth. We marked it as *not present* if the paper performs PCAP analysis to verify the ground truth (e.g. inspecting their model's false positives and false negatives).



((a)) Plot of venue counts in paper overview.



((b)) Plot of dataset citation counts in paper overview.



((c)) Plot of release year in paper overview.

Appendix B

Treatment of *CTU-13* in Chapter 4

Due to the granular labelling of *CTU-13*, treating each label as a separate class, as we do for the other datasets, is infeasible. Instead, we combine conceptually similar labels based on their protocol and purpose, determined via the provided labels. For instance, for *Neris 2*, we combine all malicious, established TCP connections into a single class, reducing 73 labels — many associated with only one flow — to a single class. The exact combinations chosen are available in the accompanying GitHub repo. In Table 4.1, we provide our results for the primary attack associated with a scenario, based on volume. We describe the associated malicious behaviours in Table B.1.

Class	Attack	Class	Attack
Neris 1	Injected Ad Traffic	Murlo	UDP C&C
Neris 2	Injected Ad Traffic	Neris 3	Injected Ad Traffic
Rbot 1	PortScan	Rbot 3	ICMP Flood
Rbot 2	UDP Flood	Rbot 4	ICMP Flood
Virut 1	SMTP Proxy	NSIS	UDP C&C
Donbot	Attempted TCP Spam	Virut 2	SMTP Proxy

Table B.1: Scenario and associated malicious behaviour.

Class	WL_C	HDF_C	Class	WL_C	HDF_C
Neris 1	**1.0**	0.73	Murlo	**1.0**	0.5
Neris 2	**0.95**	0.74	Neris 3	**0.89**	0.69
Rbot 1	**0.18**	**1.0**	Rbot 3	**0.95**	**1.0**
Rbot 2	0.03	**1.0**	Rbot 4	0.01	**1.0**
Donbot	0.01	**1.0**	NSIS	**1.0**	0.76

Table B.2: Comparative measures with *Background* traffic.

CTU-13 provides a large amount of *Background* traffic, a subset of which has been filtered and labelled as *Normal* traffic. In the main body, we use this *Normal* traffic for our comparative measures, WL_C and HDF_C. However, in some cases, based on *CTU-13*'s granular labelling, it is apparent that the *Background* traffic could provide a more challenging classification task, impacting these measures. Where applicable, we rerun our comparative tests using the *Background* flows. We present our results in Table B.2. Our other measures are not affected by this change.

Bibliography

[1] CICFlowMeter Repo. https://github.com/ahlashkari/CICFlowMeter. Accessed: 2021-12-15.

[2] DIRB: Web content scanner. https://dirb.sourceforge.net/. [Online; accessed 20-March-2025].

[3] KDD Cup 1999 Dataset. http://kdd.ics.uci.edu/\databases/kddcup99/kddcup99.html. Accessed: 2022-03-07.

[4] Maged Abdelaty, Sandra Scott-Hayward, Roberto Doriguzzi-Corin, and Domenico Siracusa. GADoT: GAN-based Adversarial Training For Robust DDoS Attack Detection. In *2021 IEEE Conference On Communications And Network Security (CNS)*, pages 119–127. IEEE, 2021.

[5] Naoki Abe, Bianca Zadrozny, and John Langford. Outlier Detection by Active Learning. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 504–509, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150459.

[6] Ahmed M. Alaa, Boris van Breugel, Evgeny S. Saveliev, and Mihaela van der Schaar. How Faithful is your Synthetic Data? Sample-level Metrics for Evaluating and Auditing Generative Models. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 290–306. PMLR, 2022.

[7] Bushra A. AlAhmadi, Enrico Mariconti, Riccardo Spolaor, Gianluca Stringhini, and Ivan Martinovic. BOTection: Bot Detection by Building Markov Chain Models of Bots Network Behavior. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, page 652–664, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367509. doi: 10.1145/3320269.3372202.

[8] Bushra A. Alahmadi, Louise Axon, and Ivan Martinovic. 99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2783–2800, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1.

[9] Huda Ali Alatwi and Charles Morisset. Realism Versus Performance For Adversarial Examples Against DL-based NIDS. In *Proceedings Of The 38th ACM/SIGAPP Symposium On Applied Computing*, pages 1549–1557. ACM, 2023.

[10] Suresh Kumar Amalapuram, Bheemarjuna Reddy Tamma, and Sumohana S Channappayya. SPIDER: A Semi-Supervised Continual Learning-based Network Intrusion Detection System. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 571–580. IEEE, 2024.

[11] Giuseppina Andresini, Feargus Pendlebury, Fabio Pierazzi, Corrado Loglisci, Annalisa Appice, and Lorenzo Cavallaro. Insomnia: Towards Concept-drift Robustness In Network Intrusion Detection. In *Proceedings Of The 14th ACM Workshop On Artificial Intelligence And Security*, pages 111–122. ACM, 2021.

[12] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110. USENIX, 2017.

[13] Giovanni Apruzzese, Pavel Laskov, and Aliya Tastemirova. SoK: The Impact of Unlabelled Data in Cyberthreat Detection. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 20–42. IEEE, 2022.

[14] Giovanni Apruzzese, Luca Pajola, and Mauro Conti. The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems. *IEEE Transactions on Network and Service Management*, 19(4):5152–5169, 2022.

[15] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 592–614. IEEE, 2023.

[16] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and Don'ts of Machine Learning in Computer Security. *CoRR*, abs/2010.09470, 2020.

[17] Bradley Ashmore and Lingwei Chen. HOVER: Homophilic Oversampling via Edge Removal for Class-Imbalanced Bot Detection on Graphs. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 3728–3732. ACM, 2023.

[18] Dina Ayesha S and Siddique AB. FS3: Few-Shot and Self-Supervised Framework for Efficient Intrusion Detection in Internet of Things Networks. In *Proceedings of the 39th Annual Computer Security Applications Conference*, pages 138–149. ACM, 2023.

[19] Maximilian Bachl, Fares Meghdouri, Joachim Fabini, and Tanja Zseby. Sparseids: Learning Packet Sampling With Reinforcement Learning. In *2020 IEEE Conference On Communications And Network Security (CNS)*, pages 1–9. IEEE, 2020.

[20] Stanley Bak, Changliu Liu, and Taylor Johnson. The Second International Verification Of Neural Networks Competition: Summary And Results. *arXiv preprint arXiv:2109.00498*, 2021.

[21] Teodora Baluta, Shiqi Shen, Shweta Shinde, et al. Quantitative Verification Of Neural Networks and its Security Applications. In *Proceedings Of The 2019 ACM SIGSAC Conference On Computer And Communications Security*, pages 1249–1264. ACM, 2019.

[22] Teodora Baluta, Zheng Leong Chua, Kuldeep S Meel, et al. Scalable Quantitative Verification for Deep Neural Networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 312–323. IEEE, 2021.

[23] Kent Beck, Martin Fowler, and Grandma Beck. Bad Smells in Code. *Refactoring: Improving the Design of Existing Code*, 1(1999):75–88, 1999.

[24] Roman Beltiukov, Wenbo Guo, Arpit Gupta, and Walter 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*, pages 2217–2231. ACM, 2023.

[25] Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are We Done With ImageNet? *arXiv preprint arXiv:2006.07159*, 2020.

[26] Siddharth Bhatia, Arjit Jain, Pan Li, Ritesh Kumar, and Bryan Hooi. MStream: Fast Anomaly Detection in Multi-Aspect Streams. In *Proceedings of the Web Conference 2021*, WWW '21, page 3371–3382, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3450023.

[27] Siddharth Bhatia, Arjit Jain, Shivin Srivastava, Kenji Kawaguchi, and Bryan Hooi. Memstream: Memory-based streaming anomaly detection. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 610–621, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390965. doi: 10.1145/3485447.3512221.

[28] Siddharth Bhatia, Mohit Wadhwa, Kenji Kawaguchi, Neil Shah, Philip S Yu, and Bryan Hooi. Sketch-based Anomaly Detection in Streaming Graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 93–104. ACM, 2023.

[29] Tim M Booij, Irina Chiscop, Erik Meeuwissen, Nour Moustafa, and Frank TH den Hartog. ToN_IoT: The Role of Heterogeneity and the Need for Standardization of Features and Attack Types in IoT Network Intrusion Data Sets. volume 9, pages 485–496. IEEE, 2021.

[30] Frederic Branchaud-Charron, Andrew Achkar, and Pierre-Marc Jodoin. Spectral Metric for Dataset Complexity Assessment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3215–3224. IEEE, 2019.

[31] Christopher Brix, mark Niklas Müller, Stanley Bak, et al. First Three Years of the International Verification of Neural Networks Competition. *International Journal on Software Tools for Technology Transfer*, 25(3):329–339, 2023.

[32] Rudy Bunel, P Mudigonda, Ilker Turkaslan, et al. Branch And Bound For Piecewise Linear Neural Network Verification. *Journal of Machine Learning Research*, 21(2020), 2020.

[33] Rudy R Bunel, Ilker Turkaslan, Philip Torr, et al. A Unified View Of Piecewise Linear Neural Network Verification. *Advances in Neural Information Processing Systems*, 31, 2018.

[34] Valentín Carela-Español, Tomasz Bujlow, and Pere Barlet-Ros. Is Our Ground-truth For Traffic Classification Reliable? In *International Conference On Passive And Active Network Measurement*, pages 98–108. Springer, 2014.

[35] Nicholas Carlini, Anish Athalye, Nicolas Papernot, et al. On Evaluating Adversarial Robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[36] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, Percy Liang, and John C. Duchi. Unlabeled Data Improves Adversarial Robustness. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

[37] Marco Casadio, Ekaterina Komendantskaya, Matthew L Daggitt, et al. Neural Network Robustness As A Verification Property: A Principled Case Study. In *International Conference On Computer Aided Verification*, pages 219–231. Springer, 2022.

[38] Marta Catillo, Andrea Del Vecchio, Antonio Pecchia, and Umberto Villano. A Critique On The Use Of Machine Learning On Public Datasets For Intrusion Detection. In *International Conference On The Quality Of Information And Communications Technology*, pages 253–266. Springer, 2021.

[39] Marta Catillo, Antonio Pecchia, and Umberto Villano. Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS? In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 132–136. IEEE, 2023.

[40] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-Sampling Technique. *J. Artif. Int. Res.*, 16(1): 321–357, Jun 2002. ISSN 1076-9757.

[41] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. Hopskipjumpattack: A Query-efficient Decision-based Attack. In *2020 IEEE Symposium on Security and Privacy (S&P)*, pages 1277–1294. IEEE, 2020.

[42] Olga Chen, Aaron D Jaggard, Catherine Meadows, and Michael C Shlanta. NExtSteP: An Extensible Testbed For Network Covert Channels. In *2020 IEEE Conference On Communications And Network Security (CNS)*, pages 1–9. IEEE, 2020.

[43] Yizheng Chen, Shiqi Wang, Yue Qin, et al. Learning Security Classifiers With Verified Global Robustness Properties. In *Proceedings Of The 2021 ACM SIGSAC Conference On Computer And Communications Security*, pages 477–494. ACM, 2021.

[44] Adriel Cheng. PAC-GAN: Packet Generation of Network Traffic Using Generative Adversarial Networks. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0728–0734. IEEE, 2019.

[45] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum Resilience Of Artificial Neural Networks. In *Automated Technology For Verification And Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pages 251–268. Springer, 2017.

[46] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic Data Repository at the {WIDE} project. In *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000.

[47] Rudi Cilibrasi and Paul MB Vitányi. Clustering by Compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[48] Henry Clausen and David Aspinall. Examining Traffic Microstructures to Improve Model Development. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 19–24. IEEE, 2021.

[49] Henry Clausen, Robert Flood, and David Aspinall. Traffic generation using containerization for machine learning. In *Proceedings of the 2019 Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security*, DYNAMICS '19, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450384902. doi: 10.1145/3464458.3464460.

[50] Joseph Clements, Yuzhe Yang, Ankur A Sharma, Hongxin Hu, and Yingjie Lao. Rallying Adversarial Techniques Against Deep Learning For Network Security. In *2021 IEEE Symposium Series On Computational Intelligence (SSCI)*, pages 01–08. IEEE, 2021.

[51] Carlos Garcia Cordero, Emmanouil Vasilomanolakis, Aidmar Wainakh, Max Mühlhäuser, and Simin Nadjm-Tehrani. On Generating Network Traffic Datasets With Synthetic Attacks For Intrusion Detection. *ACM Transactions on Privacy and Security (TOPS)*, 24(2):1–39, 2021.

[52] MITRE Corporation. MITRE ATT&CK®: The Adversarial Tactics, Techniques, and Common Knowledge Framework. `https://attack.mitre.org/`, 2025. Accessed: 25 February 2025.

[53] Andrea Corsini and Shanchieh Jay Yang. Are Existing Out-Of-Distribution Techniques Suitable for Network Intrusion Detection? In *2023 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2023.

[54] Andrea Corsini, Shanchieh Jay Yang, and Giovanni Apruzzese. On The Evaluation Of Sequential Machine Learning For Network Intrusion Detection. In *Proceedings Of The 16th International Conference On Availability, Reliability And Security*, pages 1–10. ACM, 2021.

[55] Matthew L Daggitt, Wen Kokke, Robert Atkey, Luca Arnaboldi, and Ekaterina Komendantskaya. Vehicle: Interfacing Neural Network Verifiers With Interactive Theorem Provers. *arXiv preprint arXiv:2202.05207*, 2022.

[56] Matthew L Daggitt, Wen Kokke, Robert Atkey, et al. Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs. *arXiv preprint arXiv:2401.06379*, 2024.

[57] Bernardo Damele, Miroslav Stampar, and the sqlmap Project. sqlmap: Automatic sql injection and database takeover tool. `https://github.com/sqlmapproject/sqlmap`, 2006–2025. Accessed: 2025-03-11.

[58] George Dantzig. *Linear Programming And Extensions*. Princeton University Press, 1963.

[59] Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. Supporting Standardization Of Neural Networks Verification With VNN-LIB And CoCoNet. In *Proceedings Of The 6th Workshop On Formal Methods For ML-Enabled Autonomous Systems*, volume 16, pages 47–58. EasyChair, 2023.

[60] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A Large-scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[61] Dorothy Denning and Peter G Neumann. *Requirements and Model for IDES-a Real-time Intrusion-Detection Expert System*, volume 8. SRI International Menlo Park, 1985.

[62] Dorothy E Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, (2):222–232, 1987.

[63] Alec F Diallo and Paul Patras. Adaptive Clustering-based Malicious Traffic Classification At The Network Edge. In *IEEE INFOCOM 2021-IEEE Conference On Computer Communications*, pages 1–10. IEEE, 2021.

[64] Alec F Diallo and Paul Patras. Sabre: Cutting Through Adversarial Noise With Adaptive Spectral Filtering And Input Reconstruction. In *2024 IEEE Symposium On Security And Privacy (SP)*, pages 76–76. IEEE Computer Society, 2023.

[65] Roland L Dobrushin. Prescribing a System of Random Variables by Conditional Distributions. *Theory of Probability & Its Applications*, 15(3):458–486, 1970.

[66] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, et al. Boosting Adversarial Attacks with Momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9185–9193. IEEE, 2018.

[67] Yutao Dong, Qing Li, Kaidong Wu, Ruoyu Li, Dan Zhao, Gareth Tyson, Junkun Peng, Yong Jiang, Shutao Xia, and Mingwei Xu. HorusEye: A Realtime IoT Malicious Traffic Detection Framework using Programmable Switches. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 571–588, 2023.

[68] R. Doriguzzi-Corin. LUCID Source Code. `https://github.com/doriguzzi/lucid-ddos`, 2020.

[69] Roberto Doriguzzi-Corin, Stuart Millar, Sandra Scott-Hayward, Jesus Martinez-del Rincon, and Domenico Siracusa. LUCID: A Practical, Lightweight Deep Learning Solution For DDoS Attack Detection. *IEEE Transactions on Network and Service Management*, 17(2):876–889, 2020.

[70] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly Detection And Diagnosis From System Logs Through Deep Learning. In *Proceedings Of The 2017 ACM SIGSAC Conference On Computer And Communications Security*, pages 1285–1298. ACM, 2017.

[71] Nandita Dukkipati, Matt Mathis, Yuchung Cheng, and Monia Ghobadi. Proportional Rate Reduction for TCP. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, pages 155–170. ACM, 2011.

[72] Laurens D'hooge, Miel Verkerken, Bruno Volckaert, Tim Wauters, and Filip 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*, pages 23–41. Springer, 2022.

[73] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting An Intrusion Detection Dataset: The CICIDS2017 Case Study. In *2021 IEEE Security And Privacy Workshops (SPW)*, pages 7–12. IEEE, 2021.

[74] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88:303–338, 2010.

[75] Filipe Falcão, Tommaso Zoppi, Caio Barbosa Viera Silva, Anderson Santos, Baldoino Fonseca, Andrea Ceccarelli, and Andrea Bondavalli. Quantitative Comparison of Unsupervised Anomaly Detection Algorithms for Intrusion Detection. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 318–327. ACM, 2019.

[76] Abdirisaq Farah, Martin Nielsen, and Emmanouil Vasilomanolakis. Improving Synthetic Network Attack Traffic Generation. In *9th International Workshop On Traffic Measurements For Cybersecurity*. IEEE, 2024.

[77] Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, et al. Complete Verification Via Multi-Neuron Relaxation Guided Branch-and-Bound. In *International Conference On Learning Representations*, 2022.

[78] Robert Flood. A Data-driven Toolset Using Containers to Generate Datasets for Network Intrusion Detection.

[79] Robert Flood and David Aspinall. Measuring the Complexity of Benchmark NIDS Datasets Via Spectral Analysis. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 335–341. IEEE, 2024.

[80] Robert Flood, Marco Casadio, David Aspinall, and Ekaterina Komendantskaya. Generating Traffic-Level Adversarial Examples from Feature-Level Spe- cifications. In *European Symposium on Research in Computer Security*. Springer, 2024.

[81] Robert Flood, Gints Engelen, David Aspinall, et al. Bad Design Smells In Benchmark NIDS Datasets. In *2024 IEEE 9th European Symposium On Security And Privacy (EuroS&P)*, pages 658–675. IEEE, 2024.

[82] Robert Flood, Marco Casadio, David Aspinall, and Ekaterina Komendantskaya. Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, 2025.

[83] Kevin L Fox, Ronda R Henning, Jonathan H Reed, Richard Simonian, et al. A Neural Network Approach Towards Intrusion Detection. In *Proceedings Of The 13th National Computer Security Conference*, volume 1, pages 125–134, 1990.

[84] Jun Gao, Weiming Hu, Zhongfei Zhang, Xiaoqin Zhang, and Ou Wu. RKOF: Robust Kernel-based Local Outlier Detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 270–283. Springer, 2011.

[85] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An Empirical Comparison of Botnet Detection Methods. *Computers & Security*, 45:100–123, 2014.

[86] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, et al. Ai2: Safety And Robustness Certification Of Neural Networks With Abstract Interpretation. In *2018 IEEE Symposium On Security And Privacy (SP)*, pages 3–18. IEEE, 2018.

[87] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut Learning In Deep Neural Networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

[88] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. An Evaluation Framework For Intrusion Detection Dataset. In *2016 International Conference On Information Science And Security (ICISS)*, pages 1–6. IEEE, 2016.

[89] Julie Gjerstad, Fikret Kadiric, Gudmund Grov, Espen Hammer Kjellstadli, and Markus Leira Asprusten. LADEMU: A Modular & Continuous Approach for Generating Labelled APT Datasets From Emulations. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2610–2619. IEEE, 2022.

[90] Julie Lidahl Gjerstad. Generating Labelled Network Datasets of APT with the MITRE CALDERA framework. Master's thesis, 2022.

[91] Mateusz Gniewkowski, Henryk Maciejewski, Tomasz Surmacz, and Wiktor Walentynowicz. Sec2vec: Anomaly Detection in HTTP Traffic and Malicious URLs. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 1154–1162. ACM, 2023.

[92] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6572`.

[93] Parikshit Gopalan, Vatsal Sharan, and Udi Wieder. PIDForest: Anomaly Detection via Partial Identification. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

[94] MAWI Working Group et al. Traffic Archive. *http://tracer. csl. sony. co. jp/mawi/*.

[95] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A Classification Of SQL-injection Attacks And Countermeasures. In *Proceedings Of The IEEE International Symposium On Secure Software Engineering*, volume 1, pages 13–15. IEEE, 2006.

[96] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. Deepaid: Interpreting and Improving Deep Learning-based Anomaly Detection in Security Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3197–3217. ACM, 2021.

[97] Dongqi Han, Zhiliang Wang, Ying Zhong, Wenqi Chen, Jiahai Yang, Shuqiang Lu, Xingang Shi, and Xia Yin. Evaluating And Improving Adversarial Robustness Of Machine Learning-based Network Intrusion Detectors. *IEEE Journal on Selected Areas in Communications*, 39(8):2632–2647, 2021.

[98] Zijun Hang, Yuliang Lu, Yongjie Wang, and Yi Xie. Flow-MAE: Leveraging Masked AutoEncoder for Accurate, Efficient and Robust Malicious Traffic Classification. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 297–314. ACM, 2023.

[99] Christoph Hardegen, Mike Petersen, Chukwuebuka Ezelu, Timo Geier, Sebastian Rieger, and Ulrich Buehler. A Hierarchical Architecture And Probabilistic Strategy For Collaborative Intrusion Detection. In *2021 IEEE Conference On Communications And Network Security (CNS)*, pages 128–136. IEEE, 2021.

[100] Ke He, Dan Dongseong Kim, and Muhammad Rizwan Asghar. Adversarial Machine Learning for Network Intrusion Detection Systems: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, 25(1):538–566, 2023.

[101] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained By A Two Time-scale Update Rule Converge To A Local Nash Equilibrium. *Advances in Neural Information Processing Systems*, 30, 2017.

[102] Tin Kam Ho and Mitra Basu. Complexity Measures Of Supervised Classification Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (3):289–300, 2002.

[103] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion Detection Using Sequences Of System Calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[104] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New Directions in Automated Traffic Analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3366–3383. ACM, 2021.

[105] Soumyadeep Hore, Jalal Ghadermazi, Diwas Paudel, Ankit Shah, Tapas K Das, and Nathaniel D Bastian. Deep Packgen: A Deep Reinforcement Learning Framework for Adversarial Network Packet Generation. *arXiv preprint arXiv:2305.11039*, 2023.

[106] Olakunle Ibitoye, Omair Shafiq, and Ashraf Matrawy. Analyzing Adversarial Attacks Against Deep Learning For Intrusion Detection In IoT Networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[107] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. AI/ML For Network Security: The Emperor Has No Clothes. In *Proceedings Of The 2022 ACM SIGSAC Conference On Computer And Communications Security*, pages 1537–1551. ACM, 2022.

[108] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Taul Schmitt, Francesco Bronzino, and Nick Feamster. Netdiffusion: Network Data Augmentation Through Protocol-constrained Traffic Generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 8(1):1–32, 2024.

[109] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N Cohen, and Adrian Weller. Synthetic Data–What, Why And How? *arXiv preprint arXiv:2205.03257*, 2022.

[110] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, et al. Policy Compression for Aircraft Collision Avoidance Systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2016.

[111] Anan Kabaha and Dana Drachsler Cohen. Verification Of Neural Networks' Global Robustness. *Proceedings of the ACM on Programming Languages*, 8 (OOPSLA1):1010–1039, 2024.

[112] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017.

[113] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The Marabou Framework for Verification and Analysis of Deep Neural networks. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*, pages 443–452. Springer, 2019.

[114] Anthony Kenyon, Lipika Deka, and David Elizondo. Are Public Intrusion Datasets Fit for Purpose Characterising the State of the Art in Intrusion Event Datasets. *Computers & Security*, 99:102022, 2020.

[115] Keysight. PerfectStorm Data Generation Tool. `https://www.keysight.com/gb/en/products/network-test/network-test-hardware/perfectstorm.html`, 2024.

[116] Aditya Khosla, Tinghui Zhou, Tomasz Malisiewicz, Alexei A Efros, and Antonio Torralba. Undoing the Damage Df Dataset Dias. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I 12*, pages 158–171. Springer, 2012.

[117] Isaiah J King, Xiaokui Shu, Jiyong Jang, Kevin Eykholt, Taesung Lee, and H Howie Huang. EdgeTorrent: Real-time Temporal Graph Representations for Intrusion Detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 77–91. ACM, 2023.

[118] Andrei N Kolmogorov. On Tables of Random Numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 369–376, 1963.

[119] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards The Development Of Realistic Botnet Dataset In The Internet Of Things For Network Forensic Analytics: Bot-IoT Dataset. *Future Generation Computer Systems*, 100:779–796, 2019.

[120] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (Canadian Institute for Advanced Research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, 5(4):1, 2010.

[121] Max Landauer, Maximilian Frank, Florian Skopik, Wolfgang Hotwagner, Markus Wurzenberger, and Andreas Rauber. A Framework For Automatic Labeling Of Log Datasets From Model-driven Testbeds For HIDS Evaluation. In *Proceedings Of The 2022 ACM Workshop On Secure And Trustworthy Cyber-Physical Systems*, pages 77–86. ACM, 2022.

[122] Maxime Lanvin, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Eric Totel. Towards Understanding Alerts Raised by Unsupervised Network Intrusion Detection Systems. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 135–150, 2023.

[123] Yann LeCun. The MNIST Database of Handwritten Digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[124] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, page 6, USA, 1998. USENIX Association.

[125] Laetitia Leichtnam, Eric Totel, Nicolas Prigent, and Ludovic Mé. Forensic Analysis of Network Attacks: Restructuring Security Events as Graphs and Identifying Strongly Connected Sub-Graphs. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 565–573. IEEE, 2020.

[126] Laetitia Leichtnam, Eric Totel, Nicolas Prigent, and Ludovic Mé. Sec2graph: Network Attack Detection Based on Novelty Detection on Graph Structured Data. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, pages 238–258. Springer, 2020.

[127] Linyi Li, Tao Xie, and Bo Li. SoK: Certified Robustness for Deep Neural Networks. In *2023 IEEE Symposium on Security and Privacy (S&P)*, pages 1289–1310. IEEE, 2023.

[128] Yangmin Li and Xi Yuan. An Extreme Semi-supervised Framework Based On Transformer For Network Intrusion Detection. In *2023 IEEE Conference On Communications And Network Security (CNS)*, pages 1–9. IEEE, 2023.

[129] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. COPOD: Copula-based Outlier Detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1118–1123. IEEE, 2020.

[130] Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative Adversarial Networks for Attack Generation Against Intrusion Detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 79–91. Springer, 2022.

[131] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA Off-line Intrusion Detection Evaluation. *Computer Networks*, 34(4):579–595, 2000.

[132] Changliu Liu, Tomer Arnon, Christopher Lazarus, et al. Algorithms For Verifying Deep Neural Networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

[133] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter 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)*, pages 254–262. IEEE, 2022.

[134] Miao Liu, Boyu Zhang, Wenbin Chen, and Xunlai Zhang. A Survey Of Exploitation And Detection Methods Of XSS Vulnerabilities. *IEEE access*, 7: 182004–182016, 2019.

[135] Shaohui Liu, Yi Wei, Jiwen Lu, and Jie Zhou. An Improved Evaluation Framework for Generative Adversarial Networks. *arXiv preprint arXiv:1803.07474*, 2018.

[136] Zhuang Liu and Kaiming He. A Decade's Battle on Dataset Bias: Are We There Yet?, 2024. URL `https://arxiv.org/abs/2403.08632`.

[137] Seth Lloyd. Measures of Complexity: A Nonexhaustive List. *IEEE Control Systems Magazine*, 21(4):7–8, 2001.

[138] Wai Weng Lo, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. E-Graphsage: A Graph Neural Network Based Intrusion Detection System for IoT. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2022.

[139] Alessio Lomuscio and Lalit Maganti. An Approach To Reachability Analysis For Feed-forward Relu Neural Networks. *arXiv preprint arXiv:1706.07351*, 2017.

[140] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[141] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. UGR '16: A New Dataset for the Evaluation of Cyclostationarity-based Network IDSs. *Computers & Security*, 73: 411–424, 2018.

[142] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, et al. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*, 2018.

[143] Matthew V Mahoney and Philip K Chan. Learning Nonstationary Models Of Normal Network Traffic For Detecting Novel Attacks. In *Proceedings Of The Eighth ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*, pages 376–385, 2002.

[144] Matthew V Mahoney and Philip K Chan. An Analysis Of The 1999 DARPA/Lincoln Laboratory Evaluation Data For Network Anomaly Detection. In *International Workshop On Recent Advances In Intrusion Detection*, pages 220–237. Springer, 2003.

[145] David E Mann and Steven M Christey. Towards A Common Enumeration Of Vulnerabilities. In *2nd Workshop On Research With Security Vulnerability Databases, Purdue University, West Lafayette, Indiana*, 1999.

[146] Robert A Martin, Steven M Christey, and Joe Jarzombek. The Case For Common Flaw Enumeration. In *Proceedings Of Workshop On Software Security Assurance Tools, Techniques, And Metrics*, number 500-265, 2005.

[147] Steven McCanne and Van Jacobson. The BSD Packet Filter: A New Architecture For User-level Packet Capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, page 2, USA, 1993. USENIX Association.

[148] Shibin Mei, Chenglong Zhao, Shengchao Yuan, and Bingbing Ni. Towards Bridging Sample Complexity and Model Capacity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 1972–1980, 2022.

[149] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An Ensemble Of Autoencoders For Online Network Intrusion Detection. *arXiv preprint arXiv:1802.09089*, 2018.

[150] Nour Moustafa. A New Distributed Architecture For Evaluating AI-based Security Systems At The Edge: Network TON_IoT Datasets. *Sustainable Cities and Society*, 72:102994, 2021.

[151] Nour Moustafa and Jill Slay. UNSW-NB15: A Comprehensive Data Set For Network Intrusion Detection Systems (UNSW-NB15 Network Data Set). In *2015 Military Communications And Information Systems Conference (MilCIS)*, pages 1–6. IEEE, 2015.

[152] Nour Moustafa and Jill Slay. The Evaluation of Network Anomaly Detection Systems: Statistical Analysis of the NB15 Data Set and the Comparison with the KDD99 Data Set. *Information Security Journal: A Global Perspective*, 25 (1-3):18–31, 2016.

[153] Nicolas M Müller and Karla Markert. Identifying Mislabeled Instances in Classification Datasets. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[154] Azqa Nadeem, Daniël Vos, Clinton Cao, Luca Pajola, Simon Dieck, Robert Baumgartner, and Sicco Verwer. SoK: Explainable Machine Learning for Computer Security Applications. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 221–240. IEEE, 2023.

[155] Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. Reliable Fidelity And Diversity Metrics For Generative Models. In *International Conference on Machine Learning*, pages 7176–7185. PMLR, 2020.

[156] NewEraCracker. Loic. `https://github.com/NewEraCracker/LOIC`, 2025. GitHub repository. Accessed: 25 February 2025.

[157] Quoc Phong Nguyen, Kar Wai Lim, Dinil Mon Divakaran, Kian Hsiang Low, and Mun Choon Chan. Gee: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 91–99. IEEE, 2019.

[158] Open Information Security Foundation (OISF). Suricata. `https://suricata.io/`, 2025. Accessed: 25 February 2025.

[159] Guansong Pang, Chunhua Shen, and Anton van den Hengel. Deep Anomaly Detection With Deviation Networks. In *Proceedings Of The 25th ACM SIGKDD International Conference On Knowledge Discovery & Data Mining*. ACM, 2019.

[160] Guansong Pang, Chunhua Shen, Huidong Jin, and Anton van den Hengel. Deep Weakly-Supervised Anomaly Detection. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1795–1807, 2023.

[161] Kexin Pei, Linjie Zhu, Yinzhi Cao, et al. Towards Practical Verification of Machine Learning: The Case of Computer Vision Systems. *arXiv preprint arXiv:1712.01785*, 2017.

[162] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *28th USENIX security symposium (USENIX Security 19)*, pages 729–746, 2019.

[163] CS Penrod and TJ Wagner. Another Look At The Edited Nearest Neighbor Rule. Technical report, TEXAS UNIV AT AUSTIN DEPT OF ELECTRICAL ENGINEERING, 1976.

[164] Jared M Peterson, Joffrey L Leevy, and Taghi M Khoshgoftaar. A Review And Analysis Of The Bot-IoT Dataset. In *2021 IEEE International Conference On Service-Oriented System Engineering (SOSE)*, pages 20–27. IEEE, 2021.

[165] Luca Pulina and Armando Tacchella. Challenging SMT Solvers To Verify Neural Networks. *AI Communications*, 25(2):117–135, 2012.

[166] Shebuti Rayana. ODDS Library. http://odds.cs.stonybrook.edu, 2016. Accessed: 2023-06-02.

[167] OJ Reeves. Gobuster: Directory & dns busting tool. `https://github.com/OJ/gobuster`. [Online; accessed 20-March-2025].

[168] Matthias Reif, Markus Goldstein, Armin Stahl, and Thomas M Breuel. Anomaly Detection by Combining Decision Trees and Parametric Densities. In *2008 19th International Conference on Pattern Recognition*, pages 1–4. IEEE, 2008.

[169] Maria Rigaki. Adversarial Deep Learning Against Intrusion Detection Classifiers, 2017.

[170] Maria Rigaki and Sebastian Garcia. Bringing a Gan to a Knife-fight: Adapting Malware Communication to Avoid Detection. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 70–75. IEEE, 2018.

[171] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A Survey of Network-based Intrusion Detection Data Sets. *Computers & Security*, 86:147–167, 2019.

[172] Martin Roesch et al. Snort: Lightweight Intrusion Detection For Networks. In *Lisa*, volume 99, pages 229–238, 1999.

[173] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. LabelMe: a Database and Web-based Tool for Image Annotation. *International Journal of Computer Vision*, 77:157–173, 2008.

[174] Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing Generative Models Via Precision And Recall. *Advances in neural information processing systems*, 31, 2018.

[175] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques For Training GANs. *Advances in Neural Information Processing Systems*, 29, 2016.

[176] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. "Everyone Wants To Do The Model Work, Not The Data Work": Data Cascades In High-Stakes AI. In *Proceedings Of The 2021 CHI Conference On Human Factors In Computing Systems*, pages 1–15, 2021.

[177] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards A Standard Feature Set For Network Intrusion Detection System Datasets. *Mobile networks and applications*, pages 1–14, 2022.

[178] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. Predicting Vulnerable Software Components via Text Mining. *IEEE Transactions on Software Engineering*, 40(10):993–1006, 2014.

[179] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially Robust Generalization Requires More Data. *Advances in Neural Information Processing Systems*, 31, 2018.

[180] Madeleine Schneider, David Aspinall, and Nathaniel D Bastian. Evaluating Model Robustness To Adversarial Samples In Network Intrusion Detection. In *2021 IEEE International Conference On Big Data (Big Data)*, pages 3343–3352. IEEE, 2021.

[181] Adrien Schoen, Gregory Blanc, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, and Ludovic Me. A Tale of Two Methods: Unveiling the Limitations of GAN and the Rise of Bayesian Networks for Synthetic Network Traffic Generation. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 273–286. IEEE, 2024.

[182] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An Open Large-Scale Dataset for Training Next Generation Image-text Models. *Advances in Neural Information Processing Systems*, 35:25278–25294, 2022.

[183] HyungBin Seo and MyungKeun Yoon. Generative Intrusion Detection and Prevention on Data Stream. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4319–4335, 2023.

[184] Joan Serrà, David Álvarez, Vicenç Gómez, Olga Slizovskaia, José F Núñez, and Jordi Luque. Input Complexity and Out-of-Distribution Detection with Likelihood-based Generative Models. *arXiv preprint arXiv:1909.11480*, 2019.

[185] Giorgio Severi, Simona Boboila, Alina Oprea, John Holodnak, Kendra Kratkiewicz, and Jason Matterer. Poisoning Network Flow Classifiers. In *Proceedings of the 39th Annual Computer Security Applications Conference*, pages 337–351, 2023.

[186] Iman Sharafaldin, Amirhossein Gharib, Arash Habibi Lashkari, and Ali A Ghorbani. Towards A Reliable Intrusion Detection Benchmark Dataset. *Software Networking*, 2018(1):177–200, 2018.

[187] Iman Sharafaldin, Amirhossein Gharib, Arash Habibi Lashkari, et al. Towards a Reliable Intrusion Detection Benchmark Dataset. *Software Networking*, 2018(1): 177–200, 2018.

[188] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward Generating A New Intrusion Detection Dataset And Intrusion Traffic Characterization. *ICISSP*, 1:108–116, 2018.

[189] Yam Sharon, David Berend, Yang Liu, Asaf Shabtai, and Yuval Elovici. Tantra: Timing-based Adversarial Network Traffic Reshaping Attack. *IEEE Transactions on Information Forensics and Security*, 17:3225–3237, 2022.

[190] Ryan Sheatsley, Nicolas Papernot, Michael Weisman, Gunjan Verma, and Patrick McDaniel. Adversarial Examples In Constrained Domains. *arXiv preprint arXiv:2011.01183*, 2020.

[191] Ryan Sheatsley, Blaine Hoak, Eric Pauley, et al. On the Robustness of Domain Constraints. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 495–515, 2021.

[192] Ryan Sheatsley, Nicolas Papernot, Michael J Weisman, Gunjan Verma, and Patrick McDaniel. Adversarial Examples for Network Intrusion Detection Systems. *Journal of Computer Security*, (Preprint):1–26, 2022.

[193] Ryan Sheatsley, Blaine Hoak, Eric Pauley, and Patrick McDaniel. The Space of Adversarial Strategies. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3745–3761, 2023.

[194] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. Tiresias: Predicting Security Events Through Deep Learning. In *Proceedings Of The 2018 ACM SIGSAC Conference On Computer And Communications Security*, pages 592–605, 2018.

[195] Tom Shenkar and Lior Wolf. Anomaly Detection for Tabular Data with Internal Contrastive Learning. In *International Conference on Learning Representations*, 2022.

[196] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection. *Computers & Security*, 31(3):357–374, 2012.

[197] Arumoy Shome, Luis Cruz, and Arie Van Deursen. Data Smells in Public Datasets. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, pages 205–216, 2022.

[198] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, et al. Beyond the Single Neuron Convex Barrier for Neural Network Certification. *Advances in Neural Information Processing Systems*, 32, 2019.

[199] Ankush Singla, Elisa Bertino, and Dinesh Verma. Preparing Network Intrusion Detection Deep Learning Models With Minimal Data Using Adversarial Domain Adaptation. In *Proceedings Of The 15th ACM Asia Conference On Computer And Communications Security*, pages 127–140, 2020.

[200] Nickolay Smirnov. Table for Estimating the Goodness of Fit of Empirical Distributions. *The Annals of Mathematical Statistics*, 19(2):279–281, 1948.

[201] Robin Sommer and Vern Paxson. Outside The Closed World: On Using Machine Learning For Network Intrusion Detection. In *2010 IEEE Symposium On Security And Privacy*, pages 305–316. IEEE, 2010.

[202] Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. Synthetic data–anonymisation groundhog day. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1451–1468, 2022.

[203] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre ATT&CK: Design and Philosophy. In *Technical report*. The MITRE Corporation, 2018.

[204] Danyu Sun, Joann Qiongna Chen, Chen Gong, Tianhao Wang, and Zhou Li. Netdpsyn: Synthesizing network traces under differential privacy. In *Proceedings of the 2024 ACM on Internet Measurement Conference*, pages 545–554, 2024.

[205] Girish Suryanarayana, Ganesh Samarthyam, and Tushar Sharma. Refactoring for Software Design Smells. *ACM SIGSOFT Software Engineering Notes*, 40, 2015.

[206] Cisco Systems. Netflow. `https://web.archive.org/web/20170222053806/https://pliki.ip-sa.pl/wiki/Wiki.jsp?page=NetFlow`, 2017. Archived from the original on 22 February 2017. Accessed on 25 February 2025.

[207] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties Of Neural Networks. *arXiv preprint arXiv:1312.6199*, 2013.

[208] Kymie MC Tan and Roy A Maxion. " Why 6?" Defining The Operational Limits Of Stide, An Anomaly-based Intrusion Detector. In *Proceedings 2002 IEEE Symposium On Security And Privacy*, pages 188–201. IEEE, 2002.

[209] Wesley Joon-Wie Tann, Jackie Jin Wei Tan, Joanna Purba, and Ee-Chien Chang. Filtering DDoS Attacks From Unlabeled Network Traffic Data Using Online Deep Learning. In *Proceedings Of The 2021 ACM Asia Conference On Computer And Communications Security*, pages 432–446, 2021.

[210] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A Detailed Analysis Of The KDD CUP 99 Data Set. In *2009 IEEE Symposium On Computational Intelligence For Security And Defense Applications*, pages 1–6. IEEE, 2009.

[211] Ankit Thakkar and Ritika Lohiya. A Review of the Advancement in Intrusion Detection Datasets. *Procedia Computer Science*, 167:636–645, 2020.

[212] Robert L Thorndike. Who Belongs in the Family. In *Psychometrika*. Citeseer, 1953.

[213] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating Robustness Of Neural Networks With Mixed Integer Programming. *International Conference on Learning Representations*, 2019.

[214] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[215] Antonio Torralba and Alexei A Efros. Unbiased Look at Dataset Bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.

[216] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, et al. Ensemble Adversarial Training: Attacks and Defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[217] Rafael Uetz, Marco Herzog, Louis Hackländer, Simon Schwarz, and Martin Henze. You Cannot Escape Me: Detecting Evasions of SIEM Rules in Enterprise Networks. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5179–5196, 2024.

[218] Boris Van Breugel, Trent Kyono, Jeroen Berrevoets, and Mihaela Van der Schaar. Decaf: Generating Fair Synthetic Data Using Causally-aware Generative Networks. *Advances in Neural Information Processing Systems*, 34:22221–22233, 2021.

[219] Andrea Venturi, Matteo Ferrari, Mirco Marchetti, and Michele Colajanni. ARGANIDS: A Novel Network Intrusion Detection System Based on Adversarially Regularized Graph Autoencoder. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 1540–1548, 2023.

[220] Junnan Wang, Liu Qixu, Wu Di, Ying Dong, and Xiang Cui. Crafting Adversarial Example to Bypass Flow-&ML-based Botnet Detector via RL. In *Proceedings of the 24th International Symposium on Research in Attacks, intrusions and defenses*, 2021.

[221] Kai Wang, Zhiliang Wang, Dongqi Han, Wenqi Chen, Jiahai Yang, Xingang Shi, and Xia Yin. BARS: Local Robustness Certification For Deep Learning Based Traffic Analysis Systems. In *NDSS*, 2023.

[222] Shiqi Wang, Huan Zhang, Kaidi Xu, et al. Beta-crown: Efficient Bound Propagation With Per-neuron Split Constraints For Neural Network Robustness Verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

[223] Wei Wang, Yiqiang Sheng, Jinlin Wang, Xuewen Zeng, Xiaozhou Ye, Yongzhong Huang, and Ming Zhu. HAST-IDS: Learning Hierarchical Spatial-temporal Features Using Deep Neural Networks To Improve Intrusion Detection. *IEEE access*, 6:1792–1806, 2017.

[224] Xian Wang. ENIDrift: A Fast and Adaptive Ensemble System for Network Intrusion Detection under Real-world Drift. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 785–798, 2022.

[225] Xiaolei Wang, Lin Yang, Dongyang Li, Linru Ma, Yongzhong He, Junchao Xiao, Jiyuan Liu, and Yuexiang Yang. MADDC: Multi-scale Anomaly Detection, Diagnosis and Correction for Discrete Event Logs. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 769–784, 2022.

[226] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting Intrusions Using Sequences Of System Calls: Alternative Data Models. In *Proceedings Of The 1999 IEEE Symposium On Security And Privacy (Cat. No. 99CB36344)*, pages 133–145. IEEE, 1999.

[227] Feng Wei, Hongda Li, Ziming Zhao, and Hongxin Hu. XNIDS: Explaining Deep Learning-based Network Intrusion Detection Systems for Active Intrusion Responses. In *32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA*, 2023.

[228] Dennis L Wilson. Asymptotic Properties Of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[229] Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark Barrett. Marabou 2.0: A Versatile Formal Analyzer Of Neural Networks, 2024.

[230] Haoze Wu, Omri Isac, Aleksandar Zeljić, et al. Marabou 2.0: A Versatile Formal Analyzer of Neural Networks, 2024.

[231] Renjie Wu and Eamonn Keogh. Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[232] Yang Wu, Xurui Li, Xuhong Zhang, Yangyang Kang, Changlong Sun, and Xiaozhong Liu. Community-Based Hierarchical Positive-Unlabeled (PU) Model Fusion for Chronic Disease Prediction. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 2747–2756, 2023.

[233] Junchi Xing and Chunming Wu. Detecting Anomalies in Encrypted Traffic via Deep Dictionary Learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 734–739. IEEE, 2020.

[234] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020.

[235] Dayu Yang and Hairong Qi. A Network Intrusion Detection Method Using Independent Component Analysis. In *2008 19th International Conference On Pattern Recognition*, pages 1–4. IEEE, 2008.

[236] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. CADE: Detecting And Explaining Concept Drift Samples For Security Applications. In *30Th USENIX Security Symposium (USENIX Security 21)*, pages 2327–2344, 2021.

[237] Dit-Yan Yeung and Calvin Chow. Parzen-window Network Intrusion Detectors. In *2002 International Conference On Pattern Recognition*, volume 4, pages 385–388. IEEE, 2002.

[238] Lun-Pin Yuan, Peng Liu, and Sencun Zhu. Recompose Event Sequences Vs. Predict Next Events: A Novel Anomaly Detection Approach For Discrete Event Logs. In *Proceedings Of The 2021 ACM Asia Conference On Computer And Communications Security*, pages 336–348, 2021.

[239] Zili Zha, An Wang, Yang Guo, Doug Montgomery, and Songqing Chen. BotSifter: An SDN-based Online Bot Detection Framework In Data Centers. In *2019 IEEE Conference On Communications And Network Security (CNS)*, pages 142–150. IEEE, 2019.

[240] Chaoyun Zhang, Xavier Costa-Pérez, and Paul Patras. Adversarial Attacks Against Deep Learning-based Network Intrusion Detection Systems And Defense Mechanisms. *IEEE/ACM Transactions on Networking*, 2022.

[241] J. Zhou. Security Conference Ranking. `http://jianying.space/conference-ranking.html`, 2023. Accessed: 2024-03-21.

[242] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics Via Graph Neural Networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[243] Bolor-Erdene Zolbayar, Ryan Sheatsley, Patrick McDaniel, Michael J Weisman, Sencun Zhu, Shitong Zhu, and Srikanth Krishnamurthy. Generating Practical Adversarial Network Traffic Flows Using NIDSGAN. *arXiv preprint arXiv:2203.06694*, 2022.