ConCap: Practical Network Traffic Generation for (ML- and) Flow-based Intrusion Detection Systems

Miel Verkerken[†], Laurens D'hooge[†], Bruno Volckaert[†], Filip De Turck[†], Giovanni Apruzzese[¶]
[†]*Ghent University – imec*, [¶]*University of Liechtenstein*, [¶]*Reykjavik University*,
{name.surname}@{[†]ugent.be, [¶]uni.li}

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

We tackle such a challenge in this paper. We propose ConCap, a practical tool meant to facilitate experimental research on NIDS. Through ConCap, a researcher can set up an isolated and lightweight network environment and configure it to produce network-related data, such as packets or NetFlows, that are automatically labeled—hence ready for fine-grained experiments. ConCap is rooted on open-source software and is designed to foster experimental reproducibility across the scientific community by sharing just one configuration file. Through comprehensive experiments on 10 different network activities, further expanded via in-depth analyses of 21 variants of two specific activities and of 100 repetitions of four other ones, we empirically verify that ConCap produces network data resembling that of a real-world network. We also carry out experiments on well-known benchmark datasets as well as on a real "smart-home" network, showing that, from a cyber-detection viewpoint, ConCap's automatically-labeled NetFlows are functionally equivalent to those collected in other environments. Finally, we show that ConCap enables to safely reproduce sophisticated attack chains (e.g., to test/enhance existing NIDS). Altogether, ConCap is a solution to the "data problem" that is plaguing NIDS research.

***Index Terms*—dataset, assessment, netflow, training, nids, cve, machine learning, replication, container, kubernetes, labeling**

I. Introduction

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

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

Simply put, the stark reality is that *(i)* data is required to support a paper's claims, but *(ii)* high-quality data is hard to come by in the NIDS context—especially from the viewpoint of an academic researcher [2, 8, 10]. To aggravate this problem, modern NIDS increasingly rely on data-driven techniques, such as machine learning (ML). Therefore, *labeled* data is necessary to properly evaluate the pros and cons of state-of-the-art NIDS [2]. Finally, despite the benefits that open-source datasets (under the assumption that they are correctly labeled) can provide to a researcher [11], exclusive reliance on such "static benchmarks" prevents one from *generating new data*. This impairs the assessment of existing NIDS against recent, and more sophisticated threats—such as multi-step attacks [12, 13], or attacks exploiting recently-discovered vulnerabilities [14]. We aim to rectify such a "data problem."

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

To develop ConCap, we assembled open-source technologies, such as container-related frameworks, networking utilities, and network flows (NetFlow [2, 19]) extractors. Our design *facilitates experimental reproducibility*: to (automatically) generate labeled NetFlows pertaining to one (or more) attacks, ConCap only requires the researcher to define a "scenario," i.e., a file describing the activities of the "attacker" and "target" host(s), as well as the characteristics of the overarching network channel. In this way, other researchers can replicate the same experiments just by running the same "scenario" in their own version of ConCap, thereby *removing* *the need to share data*—which can be quite big in size (e.g., the CICIDS18 [9] dataset is larger than 400GB). Such a property intrinsically fosters scientific reproducibility—which is lacking even in top-tier security venues [2, 20].

We empirically validate the realism of ConCap's generated data. Through extensive experiments wherein we compare the network traffic produced by a physical bare-metal environment with that generated by ConCap, we find that there are negligible differences (at both the packet- and NetFlow-level) between these two setups. We deeply investigate such differences, and confirm that they are due to the well-known "unpredictable" behavior of modern networks [2, 21]. Inspired by such a finding, we go one step further and examine if the data generated by ConCap is deterministic—an aspect that received limited attention from related research (e.g., [22]). Our analysis, entailing 100 executions of the same set of four distinct network activities, reveals that ConCap's output cannot be claimed to be fully deterministic—as is the case for real-world networks, too. However, through other experiments on well-known benchmarks and on a real "smart-home" network, we show that, ML-wise, the data generated by ConCap is functionally equivalent to that of other network setups.

Altogether, our evaluations demonstrate that ConCap represents a convenient tool to address the "data problem" that affects NIDS-related research. For instance, we also show how to use ConCap to carry out security assessments of NetFlow- and ML-based NIDS against recent CVE; as well as how to replicate a sophisticated attack chain, resembling nine MITRE ATT&CK tactics, and generate the corresponding (labeled) traffic. Finally, we also show that ConCap is lightweight: it takes less than 3 seconds to power-up on a laptop, and its memory footprint takes less than 40MB.

Contributions. After positioning our paper within extant literature and motivating our work (in Section §II), we:

- Propose ConCap, a system for generating ad-hoc network traffic (§III), particularly suited to *create and label* malicious datapoints, facilitating network security research; we extensively describe and justify our implementation choices—rooted in open-source software and reproducibility (§IV).
- Empirically demonstrate (§V) that ConCap generates network-traffic data resembling that of bare-metal setups, and prove that experiments run via ConCap are not fully deterministic—a behavior expected by real-world networks.
- Show that (§VI), from an ML viewpoint, the (labeled) NetFlows produced via ConCap are functionally equivalent to those originating from other setups—such as from benchmark datasets, or from a real "smart home" network. We also show that ConCap can generate data of sophisticated multi-step attacks and recent threats to test existing NIDS.

We also discuss our findings (§VII) and compare ConCap with related works (§VIII). We release all our resources [23].

II. Related Work and Motivation

We summarize the domain of NIDS (§II-A) and we outline the challenges of acquiring data for assessing NIDS (§II-B), representing the root of the problem tackled by our paper. Then, we motivate our technical (§II-C) contribution, which specifically focuses on NIDS analysing network flows (NetFlows [24]) due to the widespread usage of this datatype [2, 8, 25].

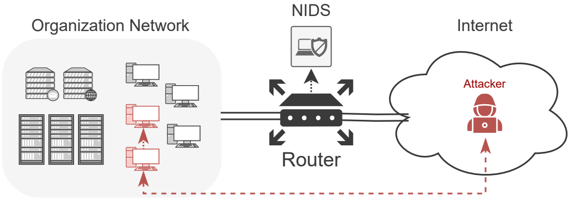

Fig. 1: **Exemplary deployment scenario of an NIDS.**

A. Network Intrusion Detection Systems (NIDS)

Modern networks are constantly under attack [26, 27], and NIDS represent the first line of defense against the ever-evolving cyberthreats [28]. The main goal of an NIDS[1] is to *detect* any given threat (e.g., a botnet-infected host, a remote DDoS attack, or an attacker who acquired access to an internal host) as early as possible so that proper mitigations can be enacted to mitigate the damage of an offensive campaign [31]. Fig. 1 shows a schematic of an NIDS deployment scenario.

The advent of data-driven technologies, such as AI/ML (which support both signature- and anomaly-based methods [2]), has been adopted by NIDS developers [32]. Yet, even state-of-the-art NIDSs struggle with the sheer amount of attacks that target current organizations [26]. Intriguingly, even though practitioners do appreciate the analytical capabilities of ML [33], modern security operation centers are overwhelmed with false alarms [34, 35]. Put simply, despite being an instrumental security tool, there is a constant need to improve NIDS—creating a fertile ground for research.

Since the seminal work by Denning [36], thousands of scientific papers have sought to propose (e.g., [37–40]), enhance (e.g., [41, 42]), or assess (e.g., [16, 43, 44]) NIDS. Despite all such efforts, a recent SoK [2] revealed that there is skepticism among industry experts with regard to the results claimed in research. Such doubts are well-founded: various recent papers (e.g., [6–8]) identified critical issues that are becoming endemic in research. The root cause of most such critiques is the *poor quality of the data* used to test these systems—which is a problem that has been known to affect this research domain since at least 2010 [21].

B. Research Challenge: Obtaining Data for NIDS

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

- *Real-world capture from the deployment environment.* This is ideally the best option since it guarantees that the data

[1] Borrowing from [2], we use "NIDS" in a broad sense, including also SIEM [29] or EDR [30] (which can be seen as extensions of NIDS).

resembles the one that will be generated by the monitored network. However, such an option may not be available: the researcher may not have access to such data due to privacy reasons, and infecting/attacking physical devices may not be acceptable in some organisations (even if for research [45]).

- *Synthetic capture from a custom environment.* This is a sensible alternative: by creating an ad-hoc network (e.g., via virtual machines [9]), a researcher has plenty of freedom to collect and generate any sort of data. However, the *benign* data may not be representative of the deployment environment. Moreover, even in such a setup, it is currently challenging to precisely distinguish benign from malicious data points: as shown in [6, 8], there is a risk of mistakenly "label" benign samples as malicious. Such errors can skew the results of the final assessment [46].
- *Reliance on benchmark datasets.* The last option is to use publicly available data, e.g., generated by other researchers in their own environments (either from physical or virtualized setups). This is a convenient option: it is exempt from privacy concerns and requires minimal technical expertise (since no simulation occurs). However, the validity of the corresponding evaluation will depend on whether the benchmark is a meaningful representation of the network wherein the NIDS is meant to be deployed.

For real-world deployments, it is paramount to evaluate the NIDS in the (real) network to be monitored by the NIDS: as highlighted by Sommer and Paxson [21], networks present immense variability. Hence, even if any given NIDS is shown to "work well" on data from a custom network, it is questionable whether the same NIDS works well also in other networks. This is known as the "generalisability" [47, 48] (or "transferability" [49]) problem of NIDS, which prevents the creation of plug-and-play NIDS (confirmed by practitioners [18]). However, *a research paper needs not to aim for real-world deployment to provide a significant contribution to the state of the art* [2]. Our work is rooted in this truth: we focus on improving future research on NIDS—which not necessarily requires assessments on "real" networks to be valuable.

C. Practical Generation of Network Data

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

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

- The researcher can create a simulated/virtual network, but doing so requires dealing with the labeling issues (for the malicious traffic), and is (likely) *limited to small-scale evaluations* due to the impossibility of recreating a large network environment [22].
- The researcher can exclusively rely on benchmark datasets, but this would *limit the evaluation to the data within the benchmark dataset* (e.g., even by manipulating the benchmark's datapoints, there is a risk of breaking domain constraints [50]), preventing exploration of new threats.
- To overcome the abovementioned limitations, the researcher can do a mix of the above [15, 16]: they can use "benign" data collected from a public dataset (potentially validated by prior work [6]), and then generate "malicious" data to use alongside the benign data to carry out a proper evaluation.

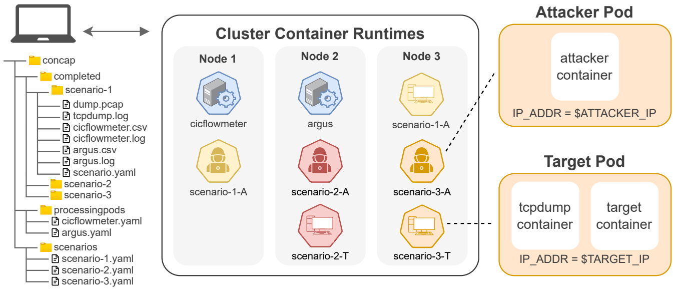

Fig. 2: **Overview of ConCap.** [left] `ConCap` configured with two NetFlow extractors and three scenarios, [mid] executing all scenarios simultaneously on the cluster. [right] A view of a running scenario's attacker and target pods.

We argue that the third option is the most enticing one. As of 2025, there are various publicly available datasets captured in large networks (see [11] for a list) whose data can be treated as "benign" for experimental purposes.[2] Hence, if one could generate "malicious" data so that *(i)* it is "correctly labeled," and *(ii)* it resembles the data generated by the host of the "benign" network, then one would be able to test a NIDS against a wide array of cyber threats—including new ones.

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

III. Our Proposed Tool: ConCap

We present `ConCap` (short for "Container Capture"), our tool to generate realistic network traffic for research in NIDS. After stating the design objectives (§III-A) and core principles (§III-B) of `ConCap`, we summarize its workflow (§III-C), schematically depicted in Fig. 2. We will discuss the low-level implementation of `ConCap` in the next section (§IV).

A. Objectives

The underlying purpose of `ConCap` is to simplify the operations involved in the generation and labeling of *malicious* network activities that resemble those in a *physical and real-world* network. `ConCap` seeks to fulfill three objectives:

- *Attack flexibility.* The tool must allow one to specify and reproduce a variety of actions that lead to network communications—including those denoting "malicious" behavior. The tool shall support both "simple" use cases (e.g., a machine attacking a different machine) as well as more "sophisticated" ones (e.g., a complex offensive campaign envisioning multi-step attacks across multiple hosts).
- *Data collection and labeling.* The tool must automatically: *(i)* capture network traffic related to the specified actions;

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

(ii) generate statistical metadata summarizing the communications, i.e., NetFlows; *(iii)* assign a label to these NetFlows.
- *Parallelism and control.* The tool must enable simultaneous execution of diverse experiments, and fine-grained control of the network conditions of each experiment—so to reproduce the behavior of any network, and enable "bulk" experiments.

To align these objectives with our overarching research goal (see §II-C), the entirety of ConCap's source code (which must leverage open-source libraries/frameworks) is publicly released [23]; moreover, the authors will make an effort to ensure long-term maintenance of ConCap's code repository.

B. Core Principles

We outline the core principles of ConCap, showing that our three objectives are embedded in its underlying design.

ConCap is rooted on the concept of "scenario", which serves as a customizable blueprint for an entire experiment. When defining a scenario, developers can specify: *(i)* the activities carried out by the involved hosts; *(ii)* the conditions of the overarching network environment; *(iii)* the fine-grained label to assign to the generated network traffic metadata.

ConCap executes each scenario in an isolated "containerized" environment that mimics the behavior of real, physical hosts, and allows for unlimited parallel execution of scenarios—subject to the resource constraints of the experimental testbed on which ConCap is executed.

A scenario in ConCap assumes a set of hosts: one designated as the "attacker", and one or more designated as the "target". These hosts are fully controllable by the developer through customizable container configurations and attack commands. ConCap captures the network packets (as a PCAP trace) exchanged between the attacker and the target(s), automatically extracts high-level NetFlow records, and then labels such NetFlows according to the scenario definition. Labeling accuracy is guaranteed by configuring the attacker host to only reproduce "malicious" actions, since the isolated environment ensures no background noise is present.

ConCap supports reproducing both simple attacks, as well as complex, multi-step attack chains. Such support is provided via the "'scenario" configuration file. In simple-attack scenarios, the attacker host executes a sequence of actions against a single host; for instance, the scenario in Listing 1 (in the Appendix B) shows a simple port-scan done by one "attacker" host against one "target" host. In complex-attack scenarios, however, it is possible to designate multiple "target" hosts—which can be either used to reproduce an attack against multiple hosts (e.g., an horizontal port scan; or a multi-chain attack in which each step requires the completion of the previous steps). The network traffic of each target is captured individually and labeled according to the specifications provided in the scenario file, enabling fine-grained analysis.

C. Architecture and Workflow

We present in Fig. 2 a schematic of a typical setup of ConCap, highlighting the most relevant logical units. We use Fig. 2 to explain the operational workflow and architecture of ConCap.

Input. On the left, the "scenarios" folder contains three YAML configuration files, each defining a given scenario to be executed by ConCap. An example scenario configuration is shown in Listing 1, highlighting the various options (i.e., attacker and target definition, network conditions, and labeling) that can be configured before executing any given experiment. For added flexibility, ConCap supports using different NetFlow generation tools. This can be specified through configuration files located in the "processingpods" folder, where the details of the desired tool are defined.

Execution. When executing a scenario, ConCap parses the configuration files and interacts with both the host machine and the cluster, consisting of one or more nodes. If multiple scenarios are specified, ConCap supports concurrent execution by distributing them across different nodes, enabling parallelism (see the central section of Fig. 2). ConCap follows the instructions in the scenario files to set up the attacker and target pods, apply custom network configurations, initiate traffic captures, trigger attack execution, and run processing pods for feature extraction and labeling. See the rightmost section of Fig. 2 for a detailed view of the attacker and a target pod during scenario execution.

Output. For each scenario, ConCap creates a dedicated folder containing all experiment outputs, including logs, the PCAP trace, and the labeled NetFlows (see left of Fig. 2). The reason why we focus (also) on NetFlows for ConCap is due to their widespread popularity for network-related experiments (both in research and in practice [2, 51]); for instance, most public benchmark datasets are also released in this format [11]. Such a design choice serves to facilitate future research.

IV. Creation and Functionalities of ConCap

We first explain and justify the key design choices followed to create ConCap (§IV-A). Then, we describe the "scenario file," which embeds the most original contributions of ConCap (§IV-B). Finally, we explain how ConCap enables also the simulation of sophisticated malicious activities, such as multi-step and multi-target attacks (§IV-C).

A. Design and Implementation Choices

We present and justify the elements that compose ConCap.

Isolated *Containerized* Environment. As explained (in §II-B), the major advantage of running network simulations is the ability to generate malicious network traffic without putting real-world networks at risk. Such simulations can leverage, e.g., virtual machines (as done, e.g., in [9]) or *containers*. Containers are a lightweight alternative to virtual machines [52], which are becoming very popular in related research (e.g., [22, 53, 54]) also for the ease of reproducibility [55] and reduced resource requirements [52]. For this reason, we used containers to develop ConCap. There exist many open-source solutions that can be used to deploy containerized applications [56]. For ConCap, we rely on Kubernetes [57], due to its widespread adoption (>110k GitHub stars [58]) and key advantages over alternatives, such as enabling deployment and management of workflows across multiple machines [59]—which is required to achieve our design goals of simultaneous execution of scenarios. For comparison, DetGen [22] relies on Docker [60], which by default does not support parallel scenario execution across multiple machines (we will compare ConCap with DetGen in §VIII-B). We stress, however, that Kubernetes alone *does not* provide the functionalities provided by ConCap (see §III-C): we simply use Kubernetes as the backbone.

Attacker and Target(s). In Kubernetes, a "pod" is the smallest deployable unit, functioning as a logical host that groups one or more containers with shared storage and networking resources. To ensure isolation and enable fine-grained control over each host involved in the "attack", we create ConCap so that the attacker and target(s) hosts are deployed in separate pods. This allows to configure parameters such as bandwidth and latency for each host. The attacker pod runs one container that simulates the behavior of the attacker's host. In contrast, each target pod runs two containers: one serving as the host targeted by the attacker, and another that captures the attacker-target network communications (via tcpdump, which can capture all traffic of the target host since containers within the same pod share networking resources).

NetFlow format. ConCap supports any type of NetFlow generation software, including novel ones (e.g., [19]). Choosing a given NetFlow tool is done by editing a dedicated configuration file (shown in Listing 2 in the Appendix B). In our proposed implementation of ConCap, we have integrated CICFlowMeter and Argus, which are popular in research and open source [2, 24]. Recent surveys [2, 19] revealed that most papers on ML-NIDS recently published in various (including top-tier) security venues rely on NetFlow for their analyses, motivating our focus on NetFlow.

Reproducibility. An implicit objective of ConCap, from a scientific viewpoint, is to facilitate the reproducibility of network traffic generation experiments. Many existing public network traffic datasets lack detailed metadata/instructions about how the traffic was produced, often providing only high-level descriptions of the experimental setup [8]. This lack of transparency makes it difficult to perform an in-depth analysis of such datasets and prevents researchers from linking individual network traffic to their specific causes or originators, which is critical for tasks such as automated detailed labeling. Reproducibility in this context means not only re-running the same scenario and obtaining consistent traffic captures and labels, but also *sharing complete experimental configurations in a way that others can reliably replicate the results on different systems*. To address this, ConCap encapsulates all aspects of the traffic generation experiment in a reusable scenario file. These files define the behavior of the attacker and target(s), their environment, and the network conditions, serving as a blueprint of the experiment. By versioning and sharing these files alongside the resulting datasets, ConCap enables other researchers to reproduce the traffic generation process with minimal manual setup and to understand the context behind the captured network traffic. Moreover, the framework enforces consistency by automating the full execution pipeline–from deployment and traffic capture to flow extraction and labeling–ensuring that experiments can be rerun with similar results.

B. Customisability (Scenario definition)

Among ConCap's greatest advantages is its customisability—which we claim as our major original contribution. Indeed, we are not aware of any tool which provides the same degree of flexibility (we discuss related work in §VIII). Such advantages are enabled by our custom implementation of the *scenario* file, for which we have provided a snippet in Listing 1. In what follows, we provide more details on the functionalities provided by the scenario file. Recall that the scenario file follows a YAML notation which requires to specify four components: the attacker, the target, the network, and the label.

Host configuration. The components describing the "attacker" and "target" hosts require a *name* and the *containerImage* to deploy the containers that simulate their behavior. Additionally, we designed ConCap so that it is possible to specify the conditions of the computing runtime (e.g., *CPU* and *memory*): this is crucial for attacks that disrupt the availability of the target host (e.g., DoS). The "attacker" component also has the *atkCommand* (used to start the attack), and an optional *atkTime* parameter that controls the attack duration (in seconds): intuitively if $atkTime > 0$, then the attack will stop after the provided number of seconds; otherwise, the attack will continue until it naturally terminates. We also allow to configure the startup probe of the "target" component, which determines when the target is ready, as well as the filter used by tcpdump for the packet capture (PCAP). Additionally, the attacker and/or target(s) can be run in privileged mode. This is useful for tools or services that need elevated permissions (e.g., raw socket access) or to simulate sophisticated attacks.

Networking. The *network* component defines the networking environment, enabling to specify, e.g., the bandwidth, latency, and packet loss of the communication channel. Such a functionality is useful to, e.g., simulate various conditions—not only for covering a wide array of "benign" network environments, but also to reproduce circumstances which can conceal traces of malicious activities (e.g., [53]). We are not aware of any open-source traffic-generation tool that enables (by default) specific configurations of the network conditions at the host level. This component, which we implemented via the open-source Traffic Control [61], also allows to determine if the generated traffic follows a specific distribution.

Labeling. The *label* component describes the labeling logic applied to the NetFlows (generated after processing the PCAP file). This can be configured globally, and individually per host. Given that ConCap enables precise control of the entire attack workflow (i.e., attacker and target(s) host and network conditions), the assigned labels ensure the resulting NetFlows are associated with the correct ground truth (since they all share the same "malicious" generative process). Moreover, since certain cyberattacks can be part of more sophisticated offensive campaigns (e.g., a port-scan can be part of a lateral-movement operation [62]), and since such use-cases are supported in ConCap, it is possible to assign more granular labels. For instance, in Listing 1 the "category" is "portscan" whereas the subcategory is "nmap": to better identify the corresponding malicious traffic in a lateral-movement context,

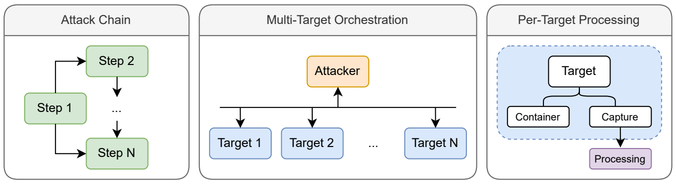

Fig. 3: **Multi-target scenario with ConCap.** [left] ConCap supports advanced multi-step attack chains, [mid] performed over multiple targets. [right] The traffic is captured and processed per target, enabling per-target labeling.

it would have been possible to specify "lateral-movement" as "category", and "port-scan" as "subcategory".

C. Sophisticated attacks (multi-step and multi-host)

In its most simple form, ConCap can be used to generate network traffic entailing one attacker and one target host. However, we designed ConCap so to also enable simulation of more complex network activities, such as multi-host attacks (e.g., an attacker host seeking to scan multiple target hosts) or multi-step attacks (e.g., an attacker that seeks to brute force an SSH server after finding out that there is an active SSH service running on such a host) in a reproducible and structured way. Here, we describe how we enabled support of multi-step/host attacks in ConCap (which we also claim as an original technical contribution).

Challenges. Simulating such advanced scenarios is nontrivial. Multi-step attacks often involve transferring state between phases (e.g., discovered credentials [62]) combined with dynamic network reconfiguration (e.g., proxy routing or port forwarding [63]). Additionally, these attacks may require accessing internal services only reachable after a successful compromise of a given gateway, demanding precise orchestration of timing, sequencing, and conditional logic.

Solutions. To address these challenges, ConCap introduces several mechanisms. Scenario definitions support dynamic environment variable substitution—instead of being constrained by hard-coded variables. Moreover, ConCap's built-in startup-readiness probes ensure that attacks are launched only when the targets are fully operational. Scenario-wide and per-target labels are automatically merged to annotate captured flows based on their context within the attack chain. Fig. 3 illustrates the fully-automated process enabled by ConCap. On the left, the developer defines an arbitrary and flexible attack chain using the "atkCommand" in the scenario file that specifies the attacker's behavior. In the middle, ConCap automatically deploys the pods, applies the network configuration, and manages pod lifecycles and readiness. On the right, it performs per-target traffic capture, flow extraction, and labeling. This workflow is repeated for each custom-defined step of the attack chain. In a sense, ConCap acts as a supervisor guaranteeing that each step is executed at the right time, thereby enabling smooth simulation of multi-step attacks—potentially requiring information acquired after intermediate steps. An end-to-end demonstration of using ConCap to reproduce a multi-step/multi-target attack chain, resembling acknowledged MITRE ATT&CK tactics, is provided in the Appendix D-B.

TAKEAWAY. ConCap is the first open-source tool for generating realistic, labeled network traffic for NIDS research. It offers attack flexibility, automated NetFlow generation and labeling, and parallel scenario execution with fine-grained control. A short demo is available in our repository [23].

V. REAL-WORLD VALIDATION OF CONCAP

The design choices we followed to develop ConCap (e.g., the usage of containers, or the usage of Traffic Control) indicate that ConCap is likely to generate traffic that mimics that of a physical real-world network. In what follows, we empirically confirm this hypothesis—a validation that has not been done in most prior works proposing traffic-generation tools (e.g., [64]). Specifically, we ask ourself our first research question (RQ1):

RESEARCH QUESTION #1: Does ConCap generate network traffic that resembles that of a physical network?

Motivation. For instance, while implementing ConCap, we may have made some mistakes; or it is possible that the usage of containers may lead to different traffic (w.r.t. that of a physical network). It could also be that there are some intrinsic differences: in which case, it is instructive to understand their root cause. Indeed, we are not aware of prior works on containers empirically testing such an hypothesis. Hence, our assessment serves to gauge whether our implementation of ConCap is correct, and that ConCap meets our overarching research goal (see §II-C); as well as studying potentially unknown properties of containers from a network perspective.

Approach. To answer RQ1, we carry out an intuitive set of experiments: we compare the network data generated by ConCap to that of a physical "bare-metal" setup—across a wide range of network activities. Such comparisons entail both quantitative and qualitative assessments, both at the network-packet level, as well as at the network-flow level. In particular, to comprehensively address RQ1, we perform three different experiments. First, as a starting point, we execute ten network activities, and see if there are any substantial differences from a quantitative viewpoint (§V-A). Second, we conduct an in-depth analysis focused on examining over twenty different variants of two specific activities types (§V-B). Third, as an ancillary experiment, we study the extent to which the traffic generated by ConCap can be considered to be deterministic (§V-C).

Common Setup and Workflow. The experiments discussed in this section entail two distinct environments.

- *Physical network.* Two "bare-metal" physical hosts, each having six Intel Core i5-9400, 32GB RAM running Ubuntu 20.04.6 and connected by a 1 Gbit switch.
- ConCap. A Kubernetes cluster (v1.29.0) with 1 control plane and 3 worker nodes. Each node has 16GB RAM, four Intel Xeon E5-2640v4, running Ubuntu 22.04.3. The machines are interconnected by a 10 Gbit switch.

For each experiment, we follow a similar procedure. We first consider the physical network. We instruct one bare-metal host, acting as the "attacker", to carry out each of the attacks envisioned in the experiment; whereas the other host acts as the "target" and is configured to enable the successful execution

TABLE I: **Broad analysis of various network activities.** We qualitatively compare the number of packets and NetFlows generated by our bare-metal and ConCap environments. Network traffic on bare-metal servers and ConCap.

Tool	Number of Packets		CICFlowMeter Flows		Argus Flows	
	Bare-metal	ConCap	Bare-metal	ConCap	Bare-metal	ConCap
ping	20	20	1	1	10	10
dig	10	10	5	5	5	5
mysql	37	27	1	1	1	1
curl/ftp	4910	1675	2	2	2	2
nmap	5	5	2	2	2	2
nmap (-sV)	128	103	10	10	11	11
patator (SSH)	30 960	31 485	680	680	680	680
patator (FTP)	2093	1860	70	70	70	70
slowloris	21 300	21 305	1500	1500	1500	1500
wfuzz	2310	2086	15	15	15	15

of the attack (e.g., if the attack is an SSH bruteforcing, the "target" will be running an SSH server). The corresponding traffic is captured (as PCAP) on the "target" host via tcpdump, and we also generate the corresponding network flows (via CICFlowMeter and Argus). Then, we consider the ConCap environment: we configure ConCap so that the "attacker" and "target" hosts resemble (in terms of computational power, software, and commands executed) of the "bare metal" machines; this ensures that the only "variable" across our experiments is the overarching environment (i.e., ConCap or the physical setup). The configuration files are in our repository [23].

A. Broad Analysis (many different network activities)

We begin our quest to answer RQ1 with a broad, preliminary analysis of quantitative nature. Specifically, we use our environments to reproduce ten network activities of various type, capture the corresponding traffic and generate the NetFlows, and quantitatively compare the overall number of packets and NetFlows generated by both environments.

Network Activities. For a comprehensive assessment, we consider activities of both "benign" and "malicious" nature. Four are clearly malicious: wfuzz (an exemplary HTTP fuzzying attack [65]); slowloris (an exemplary DoS attack [66]); as well as two variants of patator (an exemplary bruteforcing attack), one focused on SSH- and another on FTP-bruteforcing. Whereas the other activities are common network-related commands, not necessarily of a malicious nature: ping [67], dig [68], mysql [69], curl [70] over ftp [71], as well as two variants of nmap [72]. Such a broad set of activities is hence a solid basis to investigate RQ1 at a high level.

Results. We execute each activity with "default" options (we report the exact commands and configurations in the Appendix B-D). For each activity, we capture the corresponding packets and generate the respective NetFlows. We report in Table I the results of this experiment. Specifically, we report the total number of packets, as well as the total number of NetFlows (generated both via CICFlowMeter and Argus) of each activity for the physical and ConCap environments. We can already see that the number of NetFlows is a perfect match, which is a promising result in the context of answering RQ1 positively. Regarding the number of packets, we see that only three activities (ping, nmap, dig) had an equal number of packets. In the next experiment, we will better examine two activities for which we found differences: the SSH variant of patator, and inmapn its service/version probe variant.

B. In-depth Analysis (multiple variants of two activities)

Our previous experiment (§V-A) revealed that even though ConCap produced nearly-identical traffic (quantitatively-wise) to that of a physical network environment, some network activities (e.g., ssh-patator and nmap) presented some differences. Here, we better examine these phenomena.

1) ***Methodology****:* For a deep understanding, we expand our previous assessment and consider 21 variants of the aforementioned activities—up from 2 (i.e., the default configurations). Specifically, for nmap, we consider the 12 possible combinations of the scanning options: -Pn -sS -sV -sT -sU. Whereas for ssh-patator we test 9 combinations by varying: persistent=0/1 -RL=0/1 -T=1/5/10. Note that, for both patator and nmap, our variants also include the default option—which we repeat for completeness. After capturing the PCAP trace and generating the NetFlows, we first examine differences (between ConCap and the physical environment) at the packet level: to this end, we use Wireshark [73] for qualitative assessments, and then quantitatively inspect the number of packets generated by each variant of our considered activities. Finally, we quantitatively examine differences at the NetFlow level.

2) ***Packet-level analysis****:* After qualitatively inspecting the PCAP traces, we found that, across the 12 nmap and the 9 patator variants, the bytes in the individual packets generated by ConCap *exactly match those of the physical network*—except for expected differences in headers (e.g., MAC and IP addresses, high ports, checksums). We also observed some variations in the *window size* and *maximum segment size* which affect the amount of data that can be handled by the receiver: such (minimal) differences are due to the physiological diversity of each network, and their existence is evidence that the packets generated by ConCap can also present a degree of uniqueness which is intrinsic to physical real-world networks.

Next, we focus on the *total number of packets* exchanged for each activity. We report the results in Tables IIa (for nmap) and IIb (for patator). The leftmost column reports the specific options for each attack, whereas the second and third columns report the packets exchanged by the "target" and "attacker" host during the attack for both the physical and ConCap environment (we also report the NetFlows, covered in §V-B3).

- nmap. For the port scan, there is an almost perfect match. The differences occur only when the attacker probes the service running on the open port (option -sV). This is expected: the -sV option induces the server to provide the index HTML page of the Apache webserver, which is 10,918 bytes. In ConCap, such an exchange requires 2 packets, whereas the same payload requires 8 packets in the physical network—due to small differences in the network conditions such as TCP window scale [74, 75].
- patator. For the SSH brute force, we observe differences of ≈10% in the number of packets. In this case, the difference is due to the TCP/IP stack handling data acknowledgment on the different hardware setups. Additional testing in another replication experiment on a second real network showed similar but different deviations in the number of ACKs. The TPC RFC [76] allows for this nondeterministic behavior of

TABLE II: **Network traffic on bare-metal servers and ConCap.** For both network activities, the number of NetFlows is identical and the number of network packets has minimal variations (as expected in realistic networks).

Attack Options	Number of Packets		CICFlowMeter Flows		Argus Flows	
	Bare-metal	ConCap	Bare-metal	ConCap	Bare-metal	ConCap
-Pn -sS	5	5	2	2	2	2
-Pn -sS -sV	122	103	10	10	11	11
-Pn -sT	6	6	2	2	2	2
-Pn -sT -sV	127	107	10	10	11	11
-Pn -sU	4	4	3	3	4	4
-Pn -sU -sV	4	4	3	3	4	4
-sS	13	13	5	5	6	6
-sS -sV	137	113	13	13	15	15
-sT	14	14	5	5	6	6
-sT -sV	133	115	13	13	15	15
-sU	12	12	6	6	8	8
-sU -sV	12	12	6	6	8	8

-Pn = Treat host as online
-sS / -sT / -sU = TCP SYN, TCP Connect or UDP scan
-sV = Probe open ports to determine service and version info

(a) Nmap port scan

Attack Options	Number of Packets		CICFlowMeter Flows		Argus Flows	
	Bare-metal	ConCap	Bare-metal	ConCap	Bare-metal	ConCap
P=0 RL=0 T=1	95 194	78 309	3400	3400	3400	3400
P=1 RL=0 T=1	33 103	29 698	578	578	578	578
P=1 RL=1 T=1	38 726	35 332	578	578	578	578
P=0 RL=0 T=5	95 170	78 426	3400	3400	3400	3400
P=1 RL=0 T=5	33 554	30 293	595	595	595	595
P=1 RL=1 T=5	39 066	35 823	595	595	595	595
P=0 RL=0 T=10	94 941	78 347	3400	3400	3400	3400
P=1 RL=0 T=10	35 399	31 506	680	680	680	680
P=1 RL=1 T=10	40 772	37 384	680	680	680	680

P = Persistent, RL = Rate-Limit, T = Threads

(b) Patator SSH Bruteforce

"delayed ACKs" which send fewer than one ACK segment per data segment received and is even expected by the official specification [74, 77] to increase efficiency in the Internet and the hosts. However, as demonstrated by our qualitative analysis, the payload is the same.

In summary, any packet-level differences are due to inherent and unpredictable characteristics of the network, which do not affect the contents of the communication payloads.

*3) **NetFlow-level analysis**:* First, we carry out a quantitative comparison focusing on the number of NetFlows exchanged between the "attacker" and "target" host for each activity—reported in Tables IIa and IIb. Notably, there is always a perfect match: despite differences in packet counts, the number of NetFlows is consistent when the PCAP is processed by the same NetFlow software (CICFlowMeter and Argus follow a different logic to create NetFlows[3]). Then, we analyze the distributions of NetFlow features (for simplicity, we focus only on CICFlowMeter) across the different attacks. Fig. 4 presents side-by-side comparisons of the *mean packet length* distribution for all variations of our attacks. We also report the plots for *all 30 NetFlow features* in Fig. 6 (in the Appendix D). We observe that all features exhibit similar distributions, or any differences can be explained via our packet-level analyses (e.g., more packets sent with empty payload lead to a decrease in the mean packet length).

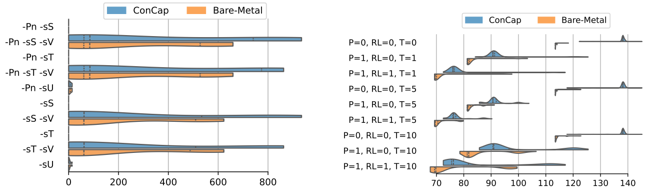

Fig. 4: **NetFlow feature distribution of *mean packet length* for traffic generated by ConCap and a pair of physical hosts.** The leftmost plot is for nmap and the rightmost is for patator. Additional plots in Fig. 6.

> **ANSWER TO RQ1.** Our packet- and NetFlow-level analyses revealed that the network traffic generated by ConCap resembles that of a physical network. Deviations are due to expected differences in the network channel, which are impossible to control—but do not affect the payload content.

C. Determinism of ConCap's generated traffic

In our previous experiments, we found an intriguing inconsistency. Indeed, by comparing the number of packets generated via the (default) nmap and ssh-patator commands in the first experiment with those of the equivalent veriant (i.e., nmap -Pn -sS -sV and ssh-patator P=1 RL=0 T=10) of the second experiment (cf. Table I with Tables II), we see minor deviations—despite the command and the setup being identical (for both ConCap and the physical network). While it is known that real-world networks exhibit immense variability [2, 21], and hence such differences may be expected, we do not know if the same also holds for ConCap. Therefore, we ask ourselves our second research question (RQ2):

> **RESEARCH QUESTION #2:** To what extent is the traffic generated by ConCap deterministic?

Motivation. There are three reasons why RQ2 is worthy of being investigated. First, to our knowledge, it is still unknown if the traffic generated by containers built on Kubernetes-related frameworks is deterministic; for instance, the authors of DetGen [22] tested this hypothesis for Docker, which is different software. Second, because among ConCap's aims is that of fostering reproducibility (§IV-A), and it is hence crucial to examine the degree of similarity across multiple executions of the same scenario via ConCap. Third, because it enables one to measure the extent to which ConCap can approximate the "unpredictable" behavior of real-world networks.

Setup. To answer RQ2, we created four scenarios (similar to those used in §V-A and §V-B) with increasing complexity in the traffic exchanged between the attacker and target: a simple ping scan (10 ICMP requests), a basic port scan (nmap -sS), a full port scan (nmap -A -T4), and an ssh brute-force attack (patator -P=1 -RL=0 -T=10). We capture the packets and generate the NetFlows for each scenario. We repeat each scenario 100 times. The intention is to study the degree of similarity across all of these repetitions. We also carry out the exact same operations (repeating them 10 times) on the bare-metal servers to compare with a physical setup approximating a real-world network, which should not be deterministic.

Results. Table III reports the results (mean and std) across our trials for both setups, detailing the number of packets,

[3] **Bugfix:** we encountered an unexpected discrepancy in the NetFlows generated by Argus. We reached out to the developers, and they confirmed that there was a bug in *their* implementation, and we helped fixed their code. After fixing the code, the number of NetFlows is identical.

TABLE III: **RQ evaluation.** We repeat the scenarios with ConCap 100 times.

Environment	Attack	Packets		Number of Flows	
		Count	Sum of Bytes	CICFlowMeter	Argus
Bare-Metal	Ping scan	20 ± 0	1960 ± 0	1 ± 0	10 ± 0
	Basic Port Scan	13 ± 0	736 ± 0	5 ± 0	6 ± 0
	Full Port Scan	2751 ± 88	271 551±6235	1091 ± 43	1093 ± 43
	SSH Bruteforce	30 935 ± 37	5 060 797±2417	680 ± 0	679 ± 0
ConCap	Ping scan	20 ± 0	1960 ± 0	1 ± 0	10 ± 0
	Basic Port Scan	13 ± 0	694 ± 0	5 ± 0	6 ± 0
	Full Port Scan	2504 ± 6	239 401±4631	1098 ± 1	1092 ± 1
	SSH Bruteforce	26 960±354	4 759 703±23 353	680 ± 0	679 ± 0

byte count, and the number of NetFlows for CICFlowMeter and Argus. We analyse these results by focusing on ConCap.

- We see no variation for the ping and basic port scan (nmap -sS) at both the packet- and NetFlow level. This shows that *the network connection is reliable*: these scenarios involve sending a single request and receiving a single response (or none). Variation would only occur if the network between the attacker and target were unreliable, causing packet duplication, loss, or corruption.
- In the complex scenarios (nmap -A -T4 and patator -P=1 -RL=0 -T=10), some variation is observed at the packet level, and in the case of the full port scan (nmap -A -T4), even at the NetFlow level. Similar to the realistic traffic assessment, packet-level variation arises from differences in how much data can be transmitted in a single packet and how this data is acknowledged. The flow-level variation in the full Nmap scan is caused by the aggressive mode (-T4) used by Nmap, which can overload the target.

By comparing the results of ConCap with those of the physical network, we see a consistent behavior at the NetFlow-level. The differences are only for the Full Port Scan and amount to less than 0.1%, which are not significant. In contrast, more pronounced differences exist for the Full Port Scan and SSH Bruteforce attacks from a packet-level viewpoint. In these cases, ConCap generates an average of ≈9% less packets and ≈9% less bytes than the physical setup.

> **Answer to RQ2.** Traffic generated by ConCap is deterministic content-wise, but the nondeterministic nature of networking results in small variations in packets and bytes. Variations are due to how data is acknowledged (e.g., using a separate TCP packet) and should not impact the outcome of a scientific experiment. ConCap does not just simulate network traffic, it generates it such that it resembles physical networks—whose real-world behavior is unpredictable [21].

VI. Applications: ConCap for Research

Here, we show the utility of ConCap via proof-of-concept experiments involving noteworthy practical applications—especially from an ML viewpoint. First, we show that ConCap can be used to replicate prior research (§VI-A). Then, we show that ConCap enables creation of ML-based NIDS from scratch (§VI-B). Finally, we show how to use ConCap for assessments of existing NIDS against unseen attacks (§VI-C).

Disclaimer: the following "proof of concept" experiments are just meant to demonstrate some applications of ConCap. We do not claim generalizable results, nor seek to cover all possible cases.

A. Replicability of Prior Research

Recall that our previous experiments (§V) revealed some small differences (at least at the packet level) in the data generated by ConCap w.r.t. of a physical network. Hence, we ask ourselves: are such differences "significant" from the viewpoint of ML-based network-intrusion detection?

Goal. We explore this question by attempting to reproduce the results of prior work focusing on ML-based NIDS. Specifically, we consider the experiments carried out by Engelen et al. [5] and Liu et al. [6], which involved assessment of supervised ML-based classifiers analysing network flows created via a (fixed) version of CICFlowMeter (the same one we used for ConCap). Specifically, the NetFlows pertain to the well-known CICIDS17 and CICIDS18 benchmark datasets [9]; such NetFlows have been rigorously labeled by the authors of [5, 6]. Both CICIDS17 and CICIDS18 include, among others, network-traffic data generated with ssh-patator (which we used in §V). Our intention is to train various ML models on the NetFlows of ssh-patator included in CICIDS17 and CICIDS18, and then testing such classifiers on the NetFlows of ssh-patator generated via ConCap, and then comparing the results: our expectation is that the classifiers trained on either "version" (i.e., CICIDS17/18, or ConCap) of ssh-patator shall always be able to detect the other version—thereby demonstrating that the data generated by ConCap is functionally equivalent (ML-wise) to that of prior research.

Setup. We follow the instructions provided in [5, 6], and retrieve the (fixed) version of CICIDS17 and CICIDS18 (more details in Appendix C-B). Then, for each dataset, we partition the NetFlows according to their classes. For this experiment, we are only interested in the ssh-patator class (which, in both CICIDS17 and CICIDS18, was launched with the default options), which we denote as $\mathcal{P}$; and the benign class, which we denote as $\mathcal{B}$. Then, we take the NetFlows generated by ConCap with the default ssh-patator (see §V-A), which we denote as $\overline{\mathcal{P}}$. For a broad assessment, we consider five different ML-based classifiers: Random Forest (RF), Decision Tree (DT), XGBoost (XGB), Support Vector Machine (SVM), as well as a Deep Neural Network (DNN); these classifiers have been tested in [5, 6], typically obtaining near-perfect performance. Then, for each dataset, we train each classifier on 80% of $\mathcal{P}$ and 80% of $\mathcal{B}$, and test it on the remaining 20% of $\mathcal{B}$ (for the false positive rate, fpr), the remaining 20% of $\mathcal{P}$, and all of $\overline{\mathcal{P}}$ (computing the true-positive rate, tpr for both sets). We then repeat this process, but by switching $\mathcal{P}$ with $\overline{\mathcal{P}}$. We repeat this procedure 5 times for statistical robustness. Note that such a workflow complies with the best practices of ML-based assessments in cybersecurity and ML-NIDS [2, 4]. The experimental source-code is in our repository [23].

Results. We report the results of our tests in Table IV. The table shows the fpr, which is always less than 0.001 (a result consistent with prior work [5, 6]), even when the classifiers were trained with ConCap's generated data. Then, looking at the tpr, we can see that the results on $\overline{\mathcal{P}}$ align with those on $\mathcal{P}$, both on CICIDS17 and CICIDS18, and irrespective of whether the training was done on 80% of $\overline{\mathcal{P}}$

TABLE IV: **Reproducibility of prior work.** We show ConCap-generated data is functionally equivalent to that of existing benchmark datasets.

	Train Set	$\mathcal{B}$+$\mathcal{P}$			$\mathcal{B}$+$\overline{\mathcal{P}}$		
	Test Set	$\mathcal{B}$	$\mathcal{P}$	$\overline{\mathcal{P}}$	$\mathcal{B}$	$\mathcal{P}$	$\overline{\mathcal{P}}$
CICIDS17	DT	<0.001	0.997	0.917	<0.001	0.980	1.000
	RF	0.000	0.998	1.000	0.000	0.986	1.000
	XGB	<0.001	0.998	0.869	<0.001	1.000	1.000
	SVM	<0.001	0.993	0.907	<0.001	0.993	1.000
	DNN	0.000	0.998	0.920	<0.001	0.996	1.000
CICIDS18	DT	0.000	1.000	0.859	<0.001	1.000	1.000
	RF	0.000	1.000	0.859	<0.001	1.000	1.000
	XGB	0.000	1.000	0.859	<0.001	0.984	1.000
	SVM	0.000	1.000	0.907	<0.001	1.000	1.000
	DNN	0.000	1.000	0.907	<0.001	1.000	1.000

or $\mathcal{P}$. Indeed, despite not being "perfectly" aligned[4] the tpr always shows that these attacks can be detected, since the tpr is always above 0.85. Importantly, this result is shared across all models/classifiers. Thus, we can say that, from an ML viewpoint, and at least according to these experiments, the (labeled) NetFlows generated via ConCap can be used to replicate prior research and derive the same conclusions.

B. Development of ML-based NIDS for the Real World

We consider the case in which a researcher may want to develop an ML-based NIDS *from scratch*. We assume that the researcher has access to a real-world network and that the hosts of such a network are not compromised. In this context, we wonder: can ConCap be used to develop an ML-NIDS that detects malicious traffic generated by real-world networks?

Goal. This experiment is conceptually similar to the one discussed in the previous subsection (§VI-A). At a high level, we will use ConCap to generate malicious network activities, and do the same on a real-world setup; and then see if using the malicious NetFlows generated via ConCap to train an ML-based classifier yields a model that can detect the real-world version of the malicious network activities—and vice versa. The difference, however, is that we are not relying on benchmark datasets here: this experiment is carried out on a real-world "smart home" network.[5] Our intention is twofold: first, verify if the results we achieved in our previous experiment also hold here—thereby reinforcing the claim that the data generated by ConCap is unlikely to alter the conclusions of a ML-based experiment; second, examine how well the model trained with ConCap's NetFlows can detect their real-world variant (which technically represents the "true attack" that the ML-NIDS should protect against).

Setup. For this experiment, we consider a real, physical network encompassing ≈50 active hosts (note: this network is different from the one of the "bare-metal" servers used in §V). These hosts include a mix of IoT devices, gaming consoles, laptops, as well as smartphones. We captured ≈20GB of network traffic in this network, which we verified to be clean of malicious activities. To capture the malicious activities, we consider two hosts (i.e., two laptops) deployed in this network, one acting as the "target" and the other as the "attacker". To align this experiment with the one in §VI-A, we consider the ssh-patator as our attack. Hence, we deploy an SSH server on the "target" host, and use the "attacker" host to launch ssh-patator (with its default options) against the SSH server. We captured the corresponding PCAP trace (on the "target" host), extracted the NetFlows (via CICFLowMeter) and manually labeled them. We then follow the same workflow as in §VI-A for the ML-based experiments, using the same notation. These experiments are also available in our repository [23] and more details on the overarching real-world network and packet capture are provided in the Appendix C-A.

TABLE V: **Real-world equivalence.** Network traffic generated with ConCap is compatible with a real-world "smart home" network.

Train Set	$\mathcal{B}$+$\mathcal{P}$			$\mathcal{B}$+$\overline{\mathcal{P}}$		
Test Set	$\mathcal{B}$	$\mathcal{P}$	$\overline{\mathcal{P}}$	$\mathcal{B}$	$\mathcal{P}$	$\overline{\mathcal{P}}$
DT	<0.001	1.000	0.899	<0.001	>0.999	>0.999
RF	0.000	1.000	0.899	0.000	1.000	1.000
XGB	0.000	>0.999	0.889	<0.001	1.000	1.000
SVM	0.000	1.000	1.000	<0.001	1.000	1.000
DNN	<0.001	1.000	1.000	<0.001	1.000	1.000

Results. The results of this experiment are in Table V. We can see a similar trend to that shown in the previous experiment (cf. Table IV with Table V). In particular, the models trained with ConCap-generated data ($\mathcal{B}$+$\overline{\mathcal{P}}$ columns) have a near-zero fpr and a near-perfect tpr, both on the malicious NetFlows generated via ConCap (i.e., $\overline{\mathcal{P}}$), which is expected; and on those generated by the physical hosts (i.e., $\mathcal{P}$). Moreover, such a good performance also encompasses the case wherein the models are trained only with real-world data ($\mathcal{B}$+$\mathcal{P}$ columns). Altogether, these finding further confirm that ConCap can be used to carry out ML- and NetFlow-based experiments (thereby reinforcing our conclusions related to §VI-A); and also show that ConCap can be used to develop ML-based NIDS meant to be deployed in the real world.

C. Security Assessment of Existing (ML-based) NIDS

Our previous experiments showed that ConCap's generated data can be used to develop ML-based NIDS, showing that, by training classifiers on some malicious data of a "known" attack, it is possible to detect future instances of the same attack. Here, we show how ConCap can be used to produce new knowledge. Specifically, we ask ourselves: how can ConCap be used to test existing NIDS against "unseen" attacks?

Threat Model. We consider a defender that uses an ML-NIDS developed by using either the CICIDS17 or CICIDS18 dataset (in their fixed version [5, 6]); such an assumption serves for broad coverage of diverse use cases, since these datasets are well-known "benchmarks", encompass a variety of attacks, and are used even in recent and top-tier publications (see, e.g., [8]). Differently from our previous experiments (in §VI-A) here we use all of the malicious data (not just that of ssh-patator) in these datasets to develop our models, leading to much larger, and therefore more effective, detectors. The attacker seeks to evade the detection by using recent exploits, whose corresponding network traffic has not been included in the training dataset of the detector used by the defender.

[4] We conjecture that such minor difference may stem—besides from unpredictable network effects—from potential labeling inaccuracies in the (fixed) CICIDS17 or CICIDS18, since these tasks have been done manually (whereas ConCap does so automatically); they can also be due to different machines (the creators of CICIDS17/18 do not provide low-level hardware details [9]).

[5] Yes, we captured traffic from a network regularly used by real people, and infected hosts deployed in such a network. We obtained permissions by the users/residents. We discuss ethical considerations at the end of the paper.

Without loss of generality, we consider three real and recently published exploits, taken from the common vulnerabilities and exposures (CVE) database: CVE-2024-47177 [78] (related to OpenPrinting Cups), CVE-2024-36401 [79] (related to GeoServer), CVE-2024-2961 [80] (related to GNU C Library). The attacker hence exploits these CVEs, and hopes that such malicious activities are not detected by the defender.

Approach. We train five ML-based classifiers (RF, DT, SVM, DNN, XGB) on the whole CICIDS17 and CICIDS18 datasets; we confirm that such ML-NIDS obtain the same performance as prior work [5, 6]. Then, we use ConCap to generate network-traffic data of the three considered CVEs. We setup ConCap accordingly, so that the target host can be "exploited" via the commands included in the CVE and executed on the attacker host. To expand our coverage (and also to show the capabilities of ConCap), we perform the captures by varying the overarching network conditions, specifying different values for *delay*, *loss*, *corrupt*, and *duplicate*, totaling over 320 combinations. Indeed, from a practical viewpoint, ConCap makes such repetitions trivial to carry out, and it is sensible to assume that an attacker may exploit any given CVE randomly, i.e., when the network conditions may be subject to bursts or noise. Overall, we capture 350k packets, corresponding to 4800, 640, and 1280 NetFlows for CVE-2024-47177, CVE-2024-36401, CVE-2024-2961, respectively. We then test our 10 models (5 per dataset) on these NetFlows.

Results. We report the results in Table VI (Appendix D). Our models are unable to reliably recognize the CVE NetFlows as malicious (the *tpr* is always below 0.6, and below 0.25 in most cases). Such an insight can be used as a signal that the ML-NIDS must be retrained—which is a trivial task since it is simply necessary to enhance the training dataset with the data generated by ConCap and re-train the ML models by, e.g., leveraging the well-known adversarial training technique [2]. (In Appendix D-B, we also demonstrate a full-fledged multi-step and multi-host attack by using ConCap.)

VII. Discussion and Critical Analyses

We evaluate the computing requirements needed to run ConCap (§VII-A), discuss the limitations of our research (§VII-B), and draw lessons learned from our experiments (§VII-C).

A. Computing Requirements and Operational Runtime

We show that ConCap can seamlessly run also on commodity hardware (e.g., laptops) by measuring its operational runtime.

Approach. We test ConCap by performing various experiments on two setups: a multi-node Kubernetes cluster (the same used in §V), and a laptop with an Intel i7-1265CPU and 23GB RAM (more hardware details in the Appendix D-A). Specifically, we consider the 12 combinations of nmap (also used in §V-B). Note that what we are interested in is the background utilization of resources (CPU and RAM) as well as the overall initialization time, i.e., the time required before the "attacker" host begins issuing its commands. Anything beyond this step does not depend on ConCap (e.g., if the attacker wants to DDoS the target, or if the scenario requires setting up a target running resource-intensive services, then such requirements are unrelated to ConCap's resource utilization).

Results. We report the results in Table VIII (in the Appendix D-A, which also provides more details). On average, less than 3 seconds elapse (on both setups) before the nmap command is launched. We also see that, on both setups, the containers running nmap requires less than 2MiB of RAM (across all of our experiments); whereas ConCap's memory footprint is less than 40MB. In terms of CPU, the utilization varies (which is expected, since it depends also on the dynamic allocation of the machine) but, on average, the max-CPU used was 10% for the cluster and 6% for the laptop. We can hence conclude that ConCap is also intrinsically lightweight.

B. Scope and Limitations

We advance the state of the art by providing a tool, ConCap, to foster future research on NID. Our contribution are aimed at *research*: real-world deployments are outside our scope (as we clearly remarked in §II). Although we did rely on real-world assessments, such assessments primarily served to prove the utility of ConCap for this specific purpose.

As for limitations, we acknowledged ConCap is not fully deterministic: while this property can be a strength (e.g., by running the same scenario multiple times, it is possible to collect more "realistic" data) it can also be a weakness (e.g., for reproducibility of some experiments expecting to achieve perfect matches). However, such a limitation is expected due to the non-deterministic nature of modern networks/protocols (see §V-B2). Second, ConCap's labeling is addressed to network flows, and cannot hence be used for labeling of other datatypes which can be used for NIDS purposes (e.g., [81]; yet, as we argued (in §IV-A) the majority of papers on NIDS rely on network flows for their analyses. Third, and related to the experiments carried out in this work, we believe the answer to our major research questions (in §V) to be correct and also generalizable (for instance, even the authors of DetGen [22] obtained similar results to ours, albeit with different software). In contrast, we acknowledge that the experiments in §VI cannot be used to cover all possible use-cases: this is why we emphasized that the experiments in that section serve as a "proof of concept" to show potential applications of ConCap.

Finally, we emphasize that ConCap can only generate data according to the specifications provided by the developer.

C. Lessons Learned

Let us summarize the major takeaways of our work.

First, ConCap automatically generates network-traffic data conforming to "any" attack. Such data resembles that of a physical network, including its non-deterministic nature (§V). We *manually verified all data labeled* by ConCap, and the labeling was always correct: the NetFlows always pertained to the malicious commands specified in the configuration file. Such a verification is confirmed by our experiment showing that such data is functionally equivalent to that of benchmarks/real-world networks (§VI-A and §VI-B).

Given the above, ConCap can be used in a variety of ways (some of which shown in §VI). For instance, through ConCap, it is possible to produce new knowledge (or test new hypotheses) without the need (and risk) to carry out experiments entailing malicious activities in real-world networks. Crucially, and as a byproduct of the experiments in §VI-C, ConCap enables early-assessments of existing NIDS (not-necessarily reliant on ML methods) against recent threats. For instance, through ConCap, the owners of an NIDS can safely test their systems by *(i)* taking any new CVE and *(ii)* running it on ConCap, and then *(iii)* either replay the PCAP trace or submit the NetFlows to the NIDS. Depending on the outcome, it is then possible to *enhance* the NIDS by, e.g., using adversarial training [2] with ConCap's generated data.

Finally, through ConCap, future research does not need to "share data" (which can require abundant storage space), rather "sharing the scenario configuration file" (typically less than 1KB in size) and the experimental details (e.g., the CVE used) is sufficient to reproduce prior results.

Q&A: We clarify some remarks (e.g., fitness to SaTML and realism of ConCap's data) made by reviewers in the Appendix A-C

VIII. Related Work

We are not aware of any open-source tool that fulfills the same goals as ConCap. Here, we first summarize related literature (§VIII-A), then compare ConCap with the two closest works we could find (§VIII-B), and finally suggest avenues in which ConCap can benefit future related research (§VIII-C).

A. Literature Summary

Some orthogonal works propose testbeds that do not provide the functionalities of ConCap: e.g., Gotham [82] cannot ensure fine-grained labeling of malicious datapoints, whereas I2DT [10] can only inject packets in a PCAP.

Closely related works are DetGen [22] and SOCBED [83]; we tried to find more works on "generation of realistic network traffic" by looking at the accepted papers in various top-tier conferences (NDSS, IEEE S&P, USENIX Sec, ACM CCS) since 2019 (similarly to [4, 84]). We found that the only related work was netUnicorn [85]. Then, we expanded our search via snowballing [86] and (recursively) looked for all peer-reviewed papers cited by [22, 85], and identified four related works [64, 87–90]. Let us position ConCap within these related works.

PINOT [87] and netUnicorn [85], can generate benign traffic but raise security risks for generating malicious traffic (see §II-B). To generate benign traffic at scale, netMosaic [88, 89] harnesses public code repositories to automatically capture network traffic for a wide range of applications, but does not allow fine-grained labeling of malicious datapoints. Finally, Zhou et al. [64] propose to use foundational models to artificially "augment" any given network-traffic dataset; however, it is unclear if such methods can produce data resembling a physical setup (there is no real-world assessment in [64]).

B. Low-level Comparisons

We compare ConCap with DetGen [91] and SOCBED [92].

ConCap vs DetGen. First, DetGen [22] does not allow parallelism by design. Second, DetGen does not generate and label the NetFlows of any given experiment, thereby forcing the developer to do so manually—which is error-prone [5]. Finally, DetGen does not allow the same degree of flexibility provided by ConCap. To provide evidence of this claim, we looked at the public repository of DetGen (available at [91]), inspected its source code, and observed the following: *(i)* we did not find any way to set the available CPU/RAM of the attacker/target; *(ii)* a predefined timeout is required for every scenario, potentially stopping the scenario before successful completion; *(iii)* comments in the code state that the attack sometimes starts before traffic is captured, highlighting the lack of fine-grained control over the scenario execution by DetGen. Although we tried to compare the traffic generated by both approaches, we were unable to run any of their example scenarios (due to deprecated software/runtime errors that we could not troubleshoot with the provided documentation).

ConCap vs SOCBED. First, SOCBED primarily targets log-data collection: even though it can create PCAPs, it does not *(a)* extract NetFlows by default, or *(b)* label them. The latter is crucial: manual post-hoc labeling of attack traffic is unreliable due to the background noise generated by the concurrent simulation of benign and malicious behaviors. Second, SOCBED relies on a *fixed* network topology designed for long-running experiments running *virtual machines* on a single host. Their sample scenario requires around one hour to complete, including a 15-minute setup time. In contrast, ConCap supports fast (startup time of seconds), parallel execution of isolated attack scenarios across multiple hosts. Finally, we found no example to configure network characteristics, such as per-host bandwidth or delay, in SOCBED.

Replication: We show in the Appendix B-B and B-C how to replicate one of DetGen's and SOCBED scenarios with ConCap.

C. Future Work

Future research can use ConCap to investigate open problems in NIDS [93, 94], such as: robustness to concept drift [95, 96], explainability [33, 97, 98], false alarms [34, 35, 99, 100], or development of novel detection techniques (not necessarily relying on NetFlows, such as [37, 101–103]). ConCap can technically also be used for generation of *benign* labeled traffic (useful for, e.g., traffic classification [104–108]). To further demonstrate the broad applicability of ConCap, we use its data to test other detection approaches [98, 109], based on deep learning or transformers, in Appendix D-C.

IX. Conclusions

Our paper is a stepping stone for future research in NID.

With ConCap, we enable researchers—especially those without access to real-world networks——to conduct realistic security assessments in networking contexts. ConCap automatic (and correct) labeling capabilities remove the burden of carrying out manual annotation. Moreover, ConCap open-source imprint fosters reproducibility and transparency.

Acknowledgments

The authors would like to thank the anonymous SaTML'26 reviewers for the feedback received, which substantially improved our work. Part of this research was funded by the Hilti Corporation. This research was also partially supported by the BOSA-AIDE project funded by the Belgian SPF BOSA with reference number 06.40.32.33.00.10. This work originated from a research stay at the University of Liechtenstein and was supported by the Research Foundation – Flanders (FWO) under grant number V450224N.

References

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, “Survey of intrusion detection systems: techniques, datasets and challenges,” *Cybersecurity*, 2019.

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

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

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

[5] G. Engelen, V. Rimmer, and W. Joosen, “Troubleshooting an intrusion detection dataset: the cicids2017 case study,” in *IEEE Security and Privacy Workshops (SPW)*, 2021.

[6] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, “Error prevalence in NIDS datasets: A case study on cic-ids-2017 and cse-cic-ids-2018,” in *IEEE CNS*, 2022.

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

[8] R. Flood, G. Engelen, D. Aspinall, and L. Desmet, “Bad Design Smells in Benchmark NIDS Datasets,” in *IEEE EusoS&P*, 2024.

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

[10] C. G. Cordero, E. Vasilomanolakis, A. Wainakh, M. Mühlhäuser, and S. Nadjm-Tehrani, “On generating network traffic datasets with synthetic attacks for intrusion detection,” *ACM TOPS*, 2021.

[11] P. Bönninghausen, R. Uetz, and M. Henze, “Introducing a comprehensive, continuous, and collaborative survey of intrusion detection datasets,” in *Cyber Security Experimentation and Test Workshop*, 2024.

[12] J. Navarro, A. Deruyver, and P. Parrend, “A systematic survey on multi-step attack detection,” *Computers & Security*, 2018.

[13] M. Corporation, “Mitre att&ck is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations.” 2013. [Online]. Available: https://attack.mitre.org

[14] “CVE: Common Vulnerabilities and Exposures,” https://www.cve.org/, 2025.

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

[16] G. Apruzzese, L. Pajola, and M. Conti, “The cross-evaluation of machine learning-based network intrusion detection systems,” *IEEE Transactions on Network and Service Management*, 2022.

[17] J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, “A survey on network simulators, emulators, and testbeds used for research and education,” *Computer Networks*, 2023.

[18] G. Apruzzese et al, “The role of machine learning in cybersecurity,” *ACM Digital Threats: Research and Practice*, 2022.

[19] M. Verkerken, M. Callewaert, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, “Rustiflow: Bridging the gap between security research and practice using ebpf-based network flow extraction,” in *WTMC (IEEE EuroS&P)*, 2025.

[20] D. Olszewski, A. Lu, C. Stillman, K. Warren, C. Kitroser, A. Pascual, D. Ukirde, K. Butler, and P. Traynor, “”get in researchers; we're measuring reproducibility”: A reproducibility study of machine learning papers in tier 1 security conferences,” in *ACM CCS*, 2023.

[21] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *IEEE S&P*, 2010.

[22] H. Clausen, R. Flood, and D. Aspinall, “Traffic generation using containerization for machine learning,” in *Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security*, 2019.

[23] “Repository of this paper,” https://github.com/idlab-discover/ConCap.

[24] G. Vormayr, J. Fabini, and T. Zseby, “Why are my flows different? a tutorial on flow exporters,” *IEEE CSUR*, 2020.

[25] L. Dias, S. Valente, and M. Correia, “Go with the flow: Clustering dynamically-defined netflow features for network intrusion detection with dynids,” in *IEEE NCA*, 2020.

[26] ENISA, “Enisa threat landscape,” ENISA, Tech. Rep., 2023. [Online]. Available: https://www.enisa.europa.eu/topics/cyber-threats/threats-and-trends

[27] CloudFlare, “DDoS threat report for 2023 Q4,” CloudFlare, Tech. Rep., 2024. [Online]. Available: https://blog.cloudflare.com/ddos-threat-report-2023-q4/

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

[29] Comodo, “Difference between siem and ids,” Tech. Rep., 2023. [Online]. Available: https://web.archive.org/web/20240829025823/https://www.comodo.com/difference-between-siem-and-ids.php

[30] PaloAlto, “What is the difference between edr vs. siem?” Tech. Rep., 2024. [Online]. Available: https://web.archive.org/web/20240829025838/https://www.paloaltonetworks.com/cyberpedia/what-is-edr-vs-siem

[31] E. Bertino and I. Karim, “Ai-powered network security: Approaches and research directions,” in *Proceedings of the 8th International Conference on Networking, Systems and Security*, 2021.

[32] N. Kshetri, “Economics of artificial intelligence in cybersecurity,” *IEEE IT Professional*, 2021.

[33] J. Mink, H. Benkraouda, L. Yang, A. Ciptadi, A. Ahmadzadeh, D. Votipka, and G. Wang, “Everybody's got ML, tell me what else you have: Practitioners' perception of ML-based security tools and explanations,” in *IEEE S&P*, 2023.

[34] B. A. Alahmadi, L. Axon, and I. Martinovic, “99% false positives: A qualitative study of {SOC} analysts' perspectives on security alarms,” in *Proc. USENIX Security Symp.*, 2022.

[35] M. Vermeer, N. Kadenko, M. van Eeten, C. Gañán, and S. Parkin, “Alert Alchemy: SOC Workflows and Decisions in the Management of NIDS Rules,” in *ACM CCS*, 2023.

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

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

[38] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, “Flowlens: Enabling efficient flow classification for ml-based network security applications.” in *NDSS*, 2021.

[39] M. Piskozub, F. De Gaspari, F. Barr-Smith, L. Mancini, and I. Martinovic, “Malphase: Fine-grained malware detection using network flow data,” in *ACM AsiaCCS*, 2021, pp. 774–786.

[40] C. Fu, Q. Li, M. Shen, and K. Xu, “Realtime robust malicious traffic detection via frequency domain analysis,” in *ACM CCS*, 2021.

[41] R. Sommer and V. Paxson, “Enhancing byte-level network intrusion detection signatures with context,” in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003.

[42] F. Araujo, G. Ayoade, K. Al-Naami, Y. Gao, K. W. Hamlen, and L. Khan, “Improving intrusion detectors by crook-sourcing,” in *Annual Computer Security Applications Conference*, 2019.

[43] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, “Ai/ml for network security: The emperor has no clothes,” in *ACM CCS*, 2022.

[44] I. Corona, G. Giacinto, and F. Roli, “Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues,” *Information sciences*, 2013.

[45] C. Wang, P. Zheng, J. Gui, C. Hua, and W. U. Hassan, “R+ r: From claims to crashes: A systematic re-evaluation of graph-based network intrusion detection systems,” in *ACSAC*, 2025.

[46] T. Krauß, J. Stang, and A. Dmitrienko, “Verify your labels! trustworthy

predictions and datasets via confidence scores," in *USENIX SEC*, 2024.

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

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

[49] M. Catillo, A. Del Vecchio, A. Pecchia, and U. Villano, "Transferability of machine learning models learned from public intrusion detection datasets: the cicids2017 case study," *Software Quality Journal*, 2022.

[50] R. Sheatsley, B. Hoak, E. Pauley, Y. Beugin, M. J. Weisman, and P. McDaniel, "On the robustness of domain constraints," in *ACM SIGSAC conference on computer and communications security*, 2021.

[51] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright, "Deepcoffea: Improved flow correlation attacks on tor via metric learning and amplification," in *IEEE S&P*, 2022.

[52] A. M. Potdar, D. Narayan, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020.

[53] M. Catillo, A. Pecchia, A. Repola, and U. Villano, "Towards realistic problem-space adversarial attacks against machine learning in network intrusion detection," in *International Conference on Availability, Reliability and Security*, 2024.

[54] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, 2015.

[55] D. Moreau, K. Wiebels, and C. Boettiger, "Containers for computational reproducibility," *Nature Reviews Methods Primers*, vol. 3, no. 1, p. 50, 2023.

[56] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Concurrency and Computation: Practice and Experience*, 2020.

[57] T. L. Foundation, "Kubernetes - production-grade container orchestration," [Online]. Available: https://kubernetes.io/.

[58] Kubernetes, "Kubernetes github," [Online]. Available: https://github.com/kubernetes/kubernetes.

[59] O. Bentaleb, A. S. Belloum, A. Sebaa, and A. El-Maouhab, "Containerization technologies: Taxonomies, applications and challenges," *The Journal of Supercomputing*, vol. 78, no. 1, pp. 1144–1181, 2022.

[60] D. inc., "Docker: Accelerate how you build, share, and run applications," 2024, [Online]. Available: https://docker.com.

[61] A. N. Kuznetsov, "tc(8) — linux manual page." [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html.

[62] G. Apruzzese, F. Pierazzi, M. Colajanni, and M. Marchetti, "Detection and threat prioritization of pivoting attacks in large networks," *IEEE transactions on emerging topics in computing*, vol. 8, no. 2, pp. 404–415, 2017.

[63] M. Angelini, S. Bonomi, S. Lenti, G. Santucci, and S. Taggi, "Mad: A visual analytics solution for multi-step cyber attacks detection," *Journal of Computer Languages*, vol. 52, pp. 10–24, 2019.

[64] G. Zhou, X. Guo, Z. Liu, T. Li, Q. Li, and K. Xu, "Trafficformer: An efficient pre-trained model for traffic data," in *IEEE Symposium on Security and Privacy 2025*, 2025.

[65] X. Mendez, "wfuzz - web application fuzzer," https://github.com/xmendez/wfuzz.

[66] G. Yaltirakli, "Slowloris," 2015. [Online]. Available: https://github.com/gkbrk/slowloris

[67] "ping - send ICMP or ICMPv6 ECHO REQUEST packets to network hosts," https://man.freebsd.org/cgi/man.cgi?ping(8).

[68] "dig — DNS lookup utility," https://man.openbsd.org/dig.1.

[69] "mysql — The MySQL Command-Line Client," https://dev.mysql.com/doc/refman/8.4/en/mysql.html.

[70] "curl, transfer a URL," https://curl.se/docs/manpage.html.

[71] "ftp, Internet file transfer program," https://linux.die.net/man/1/ftp.

[72] "nmap," https://nmap.org/.

[73] "wireshark," https://www.wireshark.org/.

[74] R. 813, "WINDOW AND ACKNOWLEDGEMENT STRATEGY IN TCP," MIT Laboratory for Computer Science, Tech. Rep., 1982. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc813

[75] R. 7323, "TCP Extensions for High Performance," Internet Engineering Task Force (IETF), Tech. Rep., 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7323

[76] R. 793, "Transmission Control Protocol," DARPA, Tech. Rep., 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc793

[77] R. 1122, "Requirements for Internet Hosts – Communication Layers," Network Working Group, Tech. Rep., 1989. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1122

[78] VulHub, "CVE-2024-47177," https://github.com/vulhub/vulhub/blob/master/cups-browsed/CVE-2024-47177, 2024.

[79] ——, "CVE-2024-36401," https://github.com/vulhub/vulhub/blob/master/geoserver/CVE-2024-36401, 2024.

[80] ——, "CVE-2024-2961," https://github.com/vulhub/vulhub/tree/master/php/CVE-2024-2961, 2024.

[81] A. Venturi, M. Ferrari, M. Marchetti, and M. Colajanni, "Arganids: a novel network intrusion detection system based on adversarially regularized graph autoencoder," in *ACM/SIGAPP Symposium on Applied Computing*, 2023.

[82] X. Sáez-de Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Gotham testbed: a reproducible iot testbed for security experiments and dataset generation," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[83] R. Uetz, C. Hemminghaus, L. Hackländer, P. Schlipper, and M. Henze, "Reproducible and adaptable log data generation for sound cybersecurity experiments," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 690–705.

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

[85] R. Beltiukov, W. Guo, A. Gupta, and W. Willinger, "In search of netunicorn: A data-collection platform to develop generalizable ml models for network security problems," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2217–2231.

[86] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014.

[87] R. Beltiukov, S. Chandrasekaran, A. Gupta, and W. Willinger, "Pinot: Programmable infrastructure for networking," in *Proceedings of the Applied Networking Research Workshop*, 2023, pp. 51–53.

[88] P. I. Khan, S. Guthula, R. Beltiukov, R. Schmid, T. Bühler, A. Gupta, L. Vanbever, and W. Willinger, "Harnessing public code repositories to develop production-ready ml artifacts for networking," in *Proceedings of the 2024 Applied Networking Research Workshop*, 2024, pp. 100–102.

[89] T. Bühler, R. Schmid, S. Lutz, and L. Vanbever, "Generating representative, live network traffic out of millions of code repositories," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, 2022, pp. 1–7.

[90] S. Guthula, N. Battula, R. Beltiukov, W. Guo, and A. Gupta, "netfound: Foundation model for network security," *arXiv preprint arXiv:2310.17025*, 2023.

[91] R. F. David Aspinall, Henry Clausen, "DetGen GitHub repository," https://github.com/detlearsom/DetGen.

[92] F. F. department of Cyber Analysis & Defense (CA&D), "SOCBED GitHub repository," https://github.com/fkie-cad/socbed.

[93] F. Ceschin, M. Botacin, A. Bifet, B. Pfahringer, L. S. Oliveira, H. M. Gomes, and A. Grégio, "Machine learning (in) security: A stream of problems," *ACM DTRAP*, 2024.

[94] A. E. Cinà, K. Grosse, A. Demontis, B. Biggio, F. Roli, and M. Pelillo, "Machine learning security against data poisoning: Are we there yet?" *Computer*, 2024.

[95] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection," in *ACM workshop on artificial intelligence and security*, 2021.

[96] X. Wang, "Enidrift: A fast and adaptive ensemble system for network intrusion detection under real-world drift," in *Annual Computer Security Applications Conference*, 2022.

[97] D. Bhusal, R. Shin, A. A. Shewale, M. K. M. Veerabhadran, M. Clifford, S. Rampazzi, and N. Rastogi, "Sok: Modeling explainability in security analytics for interpretability, trustworthiness, and usability," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, 2023, pp. 1–12.

[98] F. Wei, H. Li, Z. Zhao, and H. Hu, "{xNIDS}: Explaining deep learning-based network intrusion detection systems for active intrusion responses," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4337–4354.

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

[100] C. Fu, Q. Li, K. Xu, and J. Wu, "Point cloud analysis for ml-based malicious traffic detection: Reducing majorities of false positive alarms," in *ACM CCS*, 2023.

[101] M. Sharif, P. Datta, A. Riddle, K. Westfall, A. Bates, V. Ganti, M. Lentzk, and D. Ott, "Drsec: Flexible distributed representations for efficient endpoint security," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3609–3624.

[102] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 139–139.

[103] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "{PROGRAPHER}: An anomaly detection system based on provenance graph embedding," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[104] A. Azab, M. Khasawneh, S. Alrabaee, K.-K. R. Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," *Digital Communications and Networks*, 2024.

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

[106] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *USENIX NSDI*, 2022.

[107] V. Rimmer, T. Schnitzler, T. Van Goethem, A. Rodríguez Romero, W. Joosen, and K. Kohls, "Trace oddity: Methodologies for data-driven traffic analysis on tor," *PETS*, 2022.

[108] S. Schäfer, A. Löbel, and U. Meyer, "Accurate real-time labeling of application traffic," in *IEEE LCN*, 2022.

[109] M. Luay, S. Layeghy, Y. Pandey, G. Kulatilleke, and M. Portmann, "Multimodal llms for zero-shot intrusion detection using netflow visualisations," in *IEEE LCN*, 2025.

[110] "ssh-patator," https://github.com/lanjelot/patator.

[111] "danielmiessler/SecLists," https://github.com/danielmiessler/SecLists/tree/master.

[112] G. Apruzzese, P. Laskov, and A. Tastemirova, "Sok: The impact of unlabelled data in cyberthreat detection," in *EuroS&P*, 2022.

[113] L. D'hooge, M. Verkerken, B. Volckaert, T. Wauters, and F. De Turck, "Establishing the contaminating effect of metadata feature inclusion in machine-learned network intrusion detection models," in *DIMVA*, 2022.

[114] "Medusa," https://github.com/jmk-foofus/medusa.

[115] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE transactions on emerging topics in computational intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[116] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21 954–21 961, 2017.

[117] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, "Throwing darts in the dark? detecting bots with limited data using neural data augmentation," in *IEEE S&P*, 2020.

[118] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting http-based application layer dos attacks on web servers in the presence of sampling," *Computer Networks*, vol. 121, pp. 25–36, 2017.

Appendix A
Ethics, Open Science, and Clarifications

Following the best scientific practices, and to also comply with the CfP, in this appendix we: provide some ethical comments (Appendix A-A); describe the technical content we will publicly release (Appendix A-B); and elaborate our response to some valid remarks that we received in previous versions of this paper (Appendix A-C), while also disclosing our "responsible usage" of compute.

A. Ethical Remarks

We make the following ethical considerations.

Prevention of harm. For our research, we carried out experiments entailing "network attacks" (i.e., ssh bruteforcing [110]) on a real-world network. The authors received permission to do so, and no software/hardware was damaged as a result of these operations.

Data Confidentiality. For our research, we have collected data from two physical networks: that of the bare-metal setup, and the real-world one encompassing ∼50 hosts. We will release the full data (PCAP and NetFlows) for the bare-metal setup. For the other network, we will only provide the (labeled) NetFlows, which do not leak private data (we will anonymise the IP addresses and ports). We received permission by the owners to share this data publicly.

Respectfulness. We emphasize that our intention is not to "point the finger" against any prior work. We found a bug in the current implementation of DetGen [91] that prevented us from proceeding in our comparison (see §VIII). We contacted the developers of DetGen, informing them of the issue. They responded and confirmed that the current code is outdated and does not function properly. We are currently corresponding with them, offering our help to fix the problem.

B. Data Availability

All necessary artifacts to replicate the results in the paper will be released together with `ConCap` allowing researchers to create their own traffic generation experiments. Specifically, all our resources are available at: https://github.com/idlab-discover/ConCap [23] (an anonymised version of this repository was also provided for peer review). The repository contains the following resources:

- The source code of `ConCap` (which is based on open-source libraries)
- The containers used for our experiments.
- The PCAP and (labeled) NetFlows generated by `ConCap` that we used for our experiments.
- The PCAP trace and NetFlows of the bare-metal setup.
- The labeled NetFlows of our real-world network of 50 hosts; (we cannot release the PCAP for privacy).
- The notebooks of our experiments (for reproducibility).
- The additional data (PCAP and labeled NetFlows) used to showcase that `ConCap` can be used to generate "new" data conforming to recent attacks.

We have the permission to share all of the above.

C. Clarifications on our Research

We provide additional clarifications on four aspects: the labeling accuracy of `ConCap`, the realism of `ConCap`'s generated data, the necessity to provide ad-hoc configuration files to run `ConCap`, the resources used to carry out our experimental evaluation, and fitness of our work to the SaTML community. To facilitate understanding, we organize this appendix in a "Question-and-Answer" format.

How does `ConCap` ensure that the NetFlows are correctly labeled? This is clarified in Section III-B. The major difficulty in labeling network data (and, in our case, NetFlows) is that it is difficult precisely distinguish malicious datapoints among the myriad of activities carried out by modern machines. For instance, the labeling of most popular benchmarks is done by *(i)* generating some network traffic in synthetic environments, *(ii)* creating the corresponding NetFlows, and *(iii)* apply coarse labeling strategies—such as "assign the label $maliciousLabel to all NetFlows generated by $IPaddress between $startTime and $endTime using ports $dPort and $sPort" (see [8]). Such strategies may inevitably lead to labeling as malicious also benign/background traffic, and they may also underestimate the actual malicious traffic (e.g., there may be malicious activities related to a given malware that entail ports different from $dPort or $sPort"). In contrast, the labeling applied by ConCap *does not have such a problem by design*: thanks to its isolated environment and to the fact that the "attacker" host runs only the (supposedly malicious) commands specified in the configuration file, it is guaranteed that the packets generated by the attacker host will include only the network activities that pertain to the provided command(s). In our experiments, we have verified that this holds true (see §VII-C); moreover, the fact that the ConCap's generated-and-labeled data yielded ML-NIDS that have the same performance as in prior work (see experiment in §VI-A) further confirms that ConCap provides accurate labeling—by design and without requiring human intervention.

ConCap generates network traffic in a synthetic environment. How can such traffic be realistic? From a technical viewpoint, network packets have no notion of "real-world" or "synthetic" environment. Therefore, the payload (i.e., the actual data that flows over a network and that is exchanged between two endpoints) should be identical whether the traffic capture occurs in a "synthetic" or "physical" testbed. However, networks—irrespective of "where" they are—are characterized by having immense variability [21]. For instance, two organizations having the exact same hosts which carry out the exact same activities can have different network traffic data because, e.g., they may have different bandwidth. Thanks to ConCap's configuration file, it is possible to specify parameters of the network channel that can lead to a better approximation of the real-world network behavior. Note, however, that from the viewpoint of the malicious payload, we have empirically verified (in §V) that there are no differences.

ConCap requires manual effort to define the configuration files. Wouldn't this offset the benefit of providing automatic traffic generation and labeling? We respectfully disagree. The analysis done by Flood et al. [8] (and, previously, also by Liu et al. [6] and by Engelen et al. [5]) show that manual labeling requires extensive effort. In contrast, with ConCap, it is just necessary to, e.g., read the documentation of a given CVE and setup the hosts accordingly to produce a "configuration file" that can be used as a blueprint to carry out essentially endless variants of a given malicious activity. For instance, for the attacks we captured in §VI-C (and also in Appendix D-B), we only took ≈2 hours (we did them in the timespan between a "rebuttal phase" of a security conference, and we can provide evidence of this). Such effort is objectively smaller than that required to, e.g., set up a full-fledged network environment (even via virtual machines), capture the traffic, generate the NetFlows, and precisely label such NetFlows.

What is the memory footprint of your research? Our experiments did not use a lot of compute (we report the training times of our models in Appendix D) and ConCap requires a negligible amount of computing resources to run (see §VII-A). We do not believe more experiments are necessary to prove any of our major claims: doing so (e.g., to show how LLMs can benefit from ConCap's data) would be a waste of resources for the purpose of this specific paper.

Can you clarify how ConCap represents a significant contribution to the SaTML research community? First, we acknowledge that our paper's primary contribution, ConCap, is primarily an engineering effort. However, as shown by the EuroS&P'24 Best Paper Award [8] (and also by other recent works, e.g., ACSAC'25 [45]), the ML-NIDS research community is in desperate need of "better datasets". Our contribution, ConCap, is our response to such a need, given that it solves the labeling issue while generating valid data for ML-NIDS research experiments. Moreover, we stress that the development of ConCap goes beyond simply deploying containers. Configuring and executing reproducible, labeled network experiments end-to-end—especially in the context of *security* research—is not supported by Kubernetes out of the box. For instance, Kubernetes has plugins for networking of containers, but none provides support for traffic shaping (e.g., controlling bandwidth or latency). Moreover, Kubernetes does not natively support packet capture, NetFlow generation, or any form of labeling—the latter being the major problem affecting existing NIDS datasets [8]. While some of these steps may seem trivial in isolation, executing them reliably and reproducibly across scenarios is a known challenge. To sum up, ConCap is a tool to "unlock" new research to address the many open problems in the ML-NIDS domain: without "trustworthy" (which we intend as "correctly-labeled") data, it is difficult to provide convincing solutions. ConCap therefore enables *trustworthy data curation* in a way that is *safe to practically use* (since the capture occurs in a safe setup), both of which being themes that align with SaTML's vision.

Appendix B
ConCap Configuration

We provide information for practical use of ConCap. First, the full configuration files for a ConCap scenario and NetFlow extractor are given with all the possible configuration options. Then, a step-by-step guide is provided to replicate a scenario from DetGen [22].

A. NetFlow Exporter Configuration

Automatic NetFlow extraction in ConCap is set-up by processing configuration files. A flow extractor is created for each file, which exports NetFlows from the scenario's network traces. An example configuration file in Listing 2 for *Argus* has 3 configuration options: a "name", "containerImage", and "command". The name and container image are used to deploy the NetFlow exporter container. The command is responsible for processing the network capture file and outputting the extracted network flows as a CSV file.

Listing 1: A ConCap scenario configuration file describing a full port scan via nmap against an Apache webserver.

```
attacker:
  name: nmap
  image: instrumentisto/nmap:latest
  atkCommand: nmap $TARGET_IP -A -T4
  atkTime: 30s
  cpuRequest: 100m
  memRequest: 100Mi
  cpuLimit: 500m
  memLimit: 500Mi
target:
  name: httpd
  image: httpd:latest
  filter: host $ATTACKER_IP and host $TARGET_IP and not arp
  cpuRequest: 100m
  memRequest: 100Mi
  cpuLimit: 500m
  memLimit: 500Mi
network:
  bandwidth: 100mbit
  queueSize: 100ms
  limit: 10000
  delay: 0ms
  jitter: 0ms
  distribution: normal
  loss: 0%
  corrupt: 0%
  duplicate: 0%
  seed: 0
labels:
  label: 1
  category: port-scan
  subcategory: nmap
  scenario: nmap_A_T4
```

Listing 2: Argus Processing Definition for ConCap

```
name: argus
containerImage: ghcr.io/idlab-discover/concap/argus:
    latest
command: ”argus -r $INPUT_FILE -S 60s -w - | ra -r -
      -c , > $OUTPUT_FILE”
```

B. Configuration of a DetGen Scenario in ConCap

To demonstrate the flexibility of ConCap, we have created a step-by-step guide to implement one of the predefined scenarios from DetGen [22]. We selected the well-documented “capture-020-nginx” scenario, which uses “siege”, an HTTP load testing and benchmarking tool, to target the “nginx” HTTP server and reverse proxy. In this guide, we show how to replicate the “capture-020-nginx” scenario in ConCap using processing pods and scenario definitions.

NetFlow Configuration DetGen does not support automated NetFlow generation, thus we skip the processing pods.

Scenario Configuration The ConCap scenario is detailed in Listing 3. Here, the attacker is configured to use *siege* for 10 seconds with 10 simulated users, each making 1,000 requests to the reverse proxy’s index page. The IP address of the target is assigned through environment variable expansion. Since there is no official Docker image for *siege*, we could have opted for one of the many community-built images. However, to demonstrate the simplicity of creating a custom image, we provide a minimal Dockerfile in Listing 4, which we then push to our GitHub Container Registry. For the target, we use the official *nginx* DockerHub image. To execute the scenario twice, we duplicate the scenario definition file.

Listing 3: The replicated capture-020-nginx scenario in ConCap.

```
attacker:
  name: siege
  image: ghcr.io/idlab-discover/concap/siege:
      ubuntu18
  atkCommand: siege -c 10 -r 1000 -v http://
      $TARGET_IP
  atkTime: 10s
target:
  name: nginx
  image: nginx:1.13.8-alpine
```

Listing 4: A minimal container image for running *siege*.

```
FROM ubuntu:18.04
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update && \
    apt-get -y install siege && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists

ENTRYPOINT ["siege"]
```

C. Configuration of a SOCBED Scenario in ConCap

We further show the coverage of ConCap by providing the configuration files that enable reproduction of the experimental setup of SOCBED [83].

To this end, we report in Listing 6 the configuration that reproduces a portscan, i.e., the first step of the attack chain in SOCBED. We also provide the corresponding PCAP and (labeled) NetFlow data in our repository (note: this data is *not* provided by SOCBED). Furthermore, in our repository [23], we report additional configuration files (and corresponding data) for other scenarios in SOCBED.

We observe that creating all these configuration files required us less than one day of work: we merely investigated the public repository of SOCBED [92] to infer the low-level network and command details, required to devise our own configuration files.

D. Scenario Details for Broad Analysis ConCap

Each scenario evaluates a specific network activity or tool using its default configuration unless noted otherwise. Where applicable, we highlight the use of additional command options. All scenario files, including execution and configuration details, are available in our repository for full reproducibility.

Wfuzz This scenario simulates an HTTP fuzzing attack where the attacker attempts to discover hidden web resources by injecting path variations. The ”common” wfuzz wordlist was used as an additional parameter.

Slowloris This activity performs a Slowloris denial-of-service attack, in which the attacker opens many half-completed HTTP connections to exhaust the server’s resources. The attack was configured to run for 300 seconds, beyond the tool’s default behavior.

Patator FTP This scenario emulates an FTP brute-force attack using Patator. A password list with the 200 most used

Listing 5: A ConCap multi-target scenario file defining a multi-step attack chain against three targets. The scenario combines global with target-specific network and labeling configuration.

```
type: multi-target
name: multi-step-attack
attacker:
  name: advanced-attacker
  image: ghcr.io/idlab-discover/concap/advanced-
       attacker:1.0.0
  atkCommand: ./multi-step-attack.sh $TARGET_IP_0
       $TARGET_IP_1 $TARGET_IP_2
  cpuRequest: 100m
  memRequest: 250Mi
targets:
  - name: openssh
    image: ghcr.io/idlab-discover/concap/openssh-
         server:password-24.04
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      step: ”bruteforce-ssh”
    network:
      bandwidth: 10Mbit
      queueSize: 100ms
      delay: 100ms
  - name: db
    image: ghcr.io/idlab-discover/concap/mysql:1.0.0
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      step: ”exfiltration”
    network:
      bandwidth: 1Gbit
      queueSize: 100ms
      delay: 1ms
  - name: libssh
    image: vulhub/libssh:0.8.1
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      step: ”exploit-cve”
network:
  bandwidth: 100Mbit
  queueSize: 100ms
  delay: 5ms
labels:
  label: 1
  category: ”advanced-lateral”
```

Listing 6: A configuration of ConCap to support one of the scenarios envisioned in SOCBED which includes a portscan.

```
type: multi-target
name: socbed-example
attacker:
  name: attacker
  image: instrumentisto/nmap
  atkCommand: nmap $TARGET_IP_1 $TARGET_IP_2
       $TARGET_IP_3 $TARGET_IP_4 $TARGET_IP_5 -n --
       disable-arp-ping -sU -sV
  cpuRequest: 200m
  memRequest: 200Mi
targets:
  - name: dmz-server
    image: vulnerables/web-dvwa
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      server: ”dmz-server”
  - name: log-server
    image: kibana:9.2.1
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      server: ”log-server”
  - name: internal-server
    image: dperson/samba
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      server: ”internal-server”
  - name: client-1
    image: dockurr/windows
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      server: ”client-1”
  - name: client-2
    image: dockurr/windows
    cpuRequest: 200m
    memRequest: 200Mi
    labels:
      server: ”client-2”
network:
  bandwidth: 100Mbit
  queueSize: 100ms
  delay: 5ms
labels:
  label: 1
  category: ”socbed-example”
```

passwords of 2023 [111] was explicitly supplied, and failure responses matching the message ”530 Login incorrect.” were ignored using Patator's ignore option.

Ping A standard ICMP echo test is performed with 10 packets sent to the target host. The number of packets was explicitly specified in the command.

Patator SSH This scenario represents a brute-force attack against an SSH server. Both a username and password list were provided, top usernames shortlist and top 200 passwords from 2023 [111], and the failed login attempts are filtered based on the message ”Authentication failed.”

Nmap Version Scan In this scan, Nmap was used with additional options for version detection, SYN scan, and no DNS resolution. The scan targeted ports 79 and 80.

Nmap This scenario runs a more basic Nmap scan, using a SYN scan and disabling DNS resolution. It targeted the same ports (79 and 80) but did not include version detection.

MySQL The attacker connects to a MySQL database and issues a query to read entries from a users table. Default MySQL authentication is used (user: root, password: root), and the server was configured to listen on all interfaces.

Curl/FTP This scenario involves an authenticated file download over FTP. The attacker connects with a predefined username and password to retrieve a specific file (bible.txt). No additional options were used beyond basic authentication and output specification.

Dig (DNS) The attacker sends a series of DNS queries for five popular domain names (e.g., google.com, facebook.com) to the target DNS server on a non-standard port (63). The use of a custom port deviates from the default behavior.

Appendix C
Datasets (existing and new ones)

We provide details on the various "datasets" considered in our paper. Specifically, we first provide information on the real-world capture (used in §VI-B), then we describe how we preprocessed the benchmark datasets CICIDS17 and CICIDS18 (used in §VI-A and §VI-C), and finally we provide low-level details of our "new" dataset containing labeled NetFlows related to attacks not contained in currently available benchmarks (mentioned in §VI-C).

A. Real-word Network Capture: Description

For the experiments in §VI-B, we used data captured in a real-world network representing a "smart home". Here, we provide more details of this network environment, and the captured PCAP trace and corresponding NetFlow data.

Network Overview The network environment encompasses 40–50 physical devices. Such devices entail: smartphones, laptops / desktops, gaming consoles; as well as various IoT devices (smart speakers, lightbulbs) and media appliances (e.g., smart TV). All these devices are connected to a router through a WiFi 5 or 2.4 interface. The router is connected to the internet through a 50Mbps download speed and 5Mbps upload speed. The devices within the network are kept up-to-date with security patches and their owners are security experts, hence it is safe to assume that the traffic is "benign" (and even if some traffic is "malicious", it is of a different class than patator meaning that our conclusions are not affected by such circumstances). With regards to the devices used to simulate the attack in this network, they were two laptops both running Ubuntu 18.04; the "target" mounts an Intel i7 7700H with 32GB of RAM; the "defender" mounts an Intel N4100 with 8GB of RAM.

Network Traffic (PCAP). Three sets of network captures were performed, two for benign background traffic and one related to the patator ssh-bruteforce attacks. The background traffic was first captured on the 6th of November 2023 and a second time on the 26th of August 2024, the same day the malicious attacks were executed and captured. The total file size for the benign traffic of the August 26th capture is of 6GB for 8M packets, whereas the attack captures only measured 45MB for 241k packets. For the (benign) trace captured on November 2023, the size is of 17GB for 17M packets

NetFlow. NetFlows were extracted from the network captures using CICFlowMeter. The benign capture from November 2023 contained 30,304 unique NetFlows, while the benign capture from August 2024 contained 49,188 unique NetFlows. The malicious capture, on the other hand, included a total of 7,990 unique NetFlows.

B. Preprocessing of Benchmark datasets

In this appendix, we provide an overview of the data preprocessing steps undertaken to prepare the network traffic data for analysis. This includes the extraction of NetFlow features using the latest version of CICFlowMeter, data cleaning procedures, and a description of the real world network captures used in our experiments.

NetFlow Extraction The fixed versions of the datasets CICIDS17 and CICIDS18 were released in October 2022. Since then, CICFlowMeter, the tool used to extract the NetFlow features from the network traffic traces, has received over 30 new commits fixing, changing, and adding NetFlow features. To benefit from these updates over the last two years, we replicated the work done by Liu et al. [6] by extracting the NetFlows using the current version (Aug '24) of CICFlowMeter and labeling the flows based on their fixed logic. The "attempted" NetFlows are removed from our dataset.

Data Cleaning. The dataset cleaning steps on the NetFlows performed in our experiments follow the best practices described in previous work [4, 112, 113]. First, the meta-data and spurious features are removed: "id", "Flow ID", "Src IP", "Dst IP", "Timestamp", "FWD Init Win Bytes", and "Bwd Init Win Bytes". Second, the source and destination ports are mapped to their IANA port categories and subsequently encoded as 0, 1, or 2 for respectively *well-known*, *registered*, and *dynamic*. Third, all NetFlows with missing values are removed. Last, all duplicate NetFlows are removed.

Implementation. The features and hyperparameters used to develop our ML models are provided in our repository.

C. Generation of (new) CVE Data with `ConCap`

We generated new labeled "malicious" data using `ConCap` by executing three recent CVE-based attack scenarios, ensuring none of them overlap with the datasets mentioned in Flood et al. [8]. These CVEs represent diverse vulnerabilities, each exploited to demonstrate the flexibility of `ConCap` in handling modern attacks.

- **CVE-2024-47177** [78]: This vulnerability affects OpenPrinting Cups-Browsed versions 2.0.1 and earlier, where improper handling of the FoomaticRIPCommandLine parameter in PostScript Printer Description (PPD) files allows remote code execution. Attackers can exploit this by creating a malicious IPP (Internet Printing Protocol) server that sends crafted printer information to a vulnerable Cups-Browsed instance, enabling arbitrary command execution on the affected system.
- **CVE-2024-36401** [79]: In GeoServer versions prior to 2.25.1, 2.24.3, and 2.23.5, unauthenticated remote code execution is possible through unsafely evaluated property name expressions. Specially crafted input can exploit these unsafe evaluations in multiple OGC request parameters, allowing attackers to execute arbitrary code on a vulnerable GeoServer installation.
- **CVE-2024-2961** [80]: This vulnerability in the GNU C Library (versions 2.39 and earlier) affects the iconv() function, which may overflow the output buffer by up to 4 bytes when converting strings to the ISO-2022-CN-EXT character set. Attackers can exploit arbitrary file read vulnerabilities in PHP applications to escalate to remote code execution by leveraging the iconv() issue, potentially crashing the application or executing code.

The target environments were constructed using Vulhub, an open-source collection of pre-built vulnerable docker environments. The attacker environments were built using official

Docker images, combined with the necessary tools and exploit code for each CVE. These environments were then defined into a `ConCap` scenario together with the attack command and assigned a unique label with the corresponding CVE for automatic traffic labeling. The three scenarios, one for each CVE, are then repeated 320 times in different network environments with a unique set of values for *delay, loss, corrupt,* and *duplicate*. Each of these scenarios is repeated for five, two, and four host-space perturbations respectively for CVE-2024-47177, CVE-2024-36401, and CVE-2024-2961.

TABLE VI: **Testing existing ML-NIDS against unseen CVEs with `ConCap`.** We develop benchmark ML-NIDS using CICIDS17 (baseline tpr=0.999 with fpr=0.001) and CICIDS18 (baseline tpr=0.999 with fpr=0.001). Then, we use `ConCap` to generate and label NetFlows of three CVEs (we report the # Packets and # NetFlows of each capture) involving completely different attacks than those used to "train" the ML-NIDS, and we test them against all of our models (RF, DT, SVM, DNN, HGB). The results show the average tpr (and std. dev.) across all models.

	Traffic Statistics		CICIDS17	CICIDS18
Attack	**# Packets**	**# NetFlows**	tpr (std. dev)	tpr (std. dev)
CVE-2024-47177	44658	4800	0.205 (0.389)	0.550 (0.396)
CVE-2024-36401	10372	640	0.199 (0.399)	0.214 (0.378)
CVE-2024-2961	391350	1280	0.112 (0.224)	0.011 (0.021)

Appendix D
Additional Results and Experiments

We presents detailed results for the interested reader.

First, we provide the training time of the models evaluated on the real-world network in Table VII. Finally, Fig. 6 shows an additional 30 NetFlow features compared for the patator brute force attack between traffic generated by `ConCap` and on a real network as described in §V.

We provide the tables with the standard deviation of our results (useful to carry out statistical tests) in our repository [23].

A. Runtime and Computing Results

We report in Table VII the training time of the ML models used for the experiments in §VI. These experiments have been carried out on a machine running Windows 11 Pro (Build 26100) with a Intel 12th Gen Core i7-1265U CPU 1.80 GHz (10 cores, 12 threads) and 32GB RAM.

Then, we report in Table VIII the results of the runtime assessment of `ConCap` (covered in §VII-A). The laptop used in these tests runs MicroK8s v1.32.3 in a virtualized Linux environment on Windows Subsystem for Linux 2 (WSL2). The host system features an Intel 12th Gen Core i7-1265U CPU 1.80 GHz (10 cores, 12 threads), 32GB RAM, and Windows 11 Pro (Build 26100), WSL2 has access to 12 logical CPUs and 23 GiB of RAM. During our tests, 12 variations of nmap scenarios are executed in a single `ConCap` run configured to run three scenarios concurrently. `ConCap` shows minimal differences in initialization time and resource usage across the two setups. Interestingly, initialization time is slightly lower on the laptop despite its more limited resources. We attribute this to the elimination of network communication delays between the machine running `ConCap` and the cluster executing the scenarios, as the microK8s setup is entirely local. Initialization time includes spawning the attacker and target containers, configuring the scenario network environment, and activating packet capture on the targets, as well as associated network delays. We also measured the resource consumption of the processing pods: both argus and cicflowmeter. While argus remains lightweight, cicflowmeter consistently exhibits an order of magnitude higher CPU and memory usage compared to the scanning scenarios themselves—highlighting the relative cost of traffic processing versus generation. Although both environments perform similarly, the Kubernetes cluster provides horizontal scalability, enabling the execution of an increased number of scenarios in parallel when needed.

TABLE VII: **Training times for the models for the assessment of the real-world network.** Avg time (sec) and std. dev. over 5 training folds.

Train Set	Benign + P=1	Benign + P=0
DT	0.3 ± 0.0	0.2 ± 0.0
RF	3.1 ± 0.1	4.1 ± 0.2
XGB	3.0 ± 0.4	3.9 ± 0.6
SVM	0.5 ± 0.0	0.9 ± 0.1
DNN	9.2 ± 0.5	12.3 ± 0.5

(a) Real-world

	Train time
DT	6.7 ± 1.3
RF	40.8 ± 4.2
XGB	15.1 ± 3.8
SVM	4.1 ± 0.9
DNN	17.2 ± 2.0

(b) CICIDS17

	Train time
DT	40.4 ± 1.8
RF	547.8 ± 6.3
XGB	118.8 ± 7.4
SVM	101.3 ± 10.9
DNN	574.5 ± 45.8

(c) CICIDS18

TABLE VIII: **Performance evaluation of `ConCap` running various port scan scenarios on a multi-node K8s cluster versus microK8s on a WSL-enabled laptop.** The results show minimal differences in initialization time and resource usage between the two environments. However, the K8s cluster offers horizontal scalability, allowing for greater resource availability and enabling more parallel scenario executions.

	Cluster			**Laptop**		
Scenario	Init Time (s)	CPU (mcpu)	Mem (MiB)	Init Time (s)	CPU (mcpu)	Mem (MiB)
nmap-pn-ss	2.202	4.10	1.67	2.300	2.77	0.68
nmap-pn-ss-sv	2.696	20.94	1.67	2.155	2.67	2.00
nmap-pn-st	2.622	7.89	1.67	1.866	4.37	1.63
nmap-pn-st-sv	2.737	3.33	1.67	2.189	2.53	2.00
nmap-pn-su	2.747	3.33	1.67	2.095	4.52	1.67
nmap-pn-su-sv	2.724	19.76	1.67	2.869	1.00	1.67
nmap-ss	2.679	3.00	1.67	2.300	1.00	1.67
nmap-ss-sv	4.682	10.88	1.67	2.155	11.03	2.00
nmap-st	2.286	4.67	1.67	1.866	4.40	1.67
nmap-st-sv	3.996	15.17	1.67	2.189	8.71	2.00
nmap-su	3.767	8.00	1.67	2.095	3.00	1.67
nmap-su-sv	2.536	1.00	1.67	2.869	8.00	1.67
argus	—	18.19	0.00	—	14.88	0.00
cicflowmeter	—	119.85	2.98	—	109.10	10.95
`ConCap`	—	33.18	35.29	—	29.95	33.76
Avg Init Time	2.97 s			2.25 s		

B. Reproducing Multi-step and Multi-host attacks

We conclude our practical demonstrations by showing how to use `ConCap` to carry out experiments entailing sophisticated

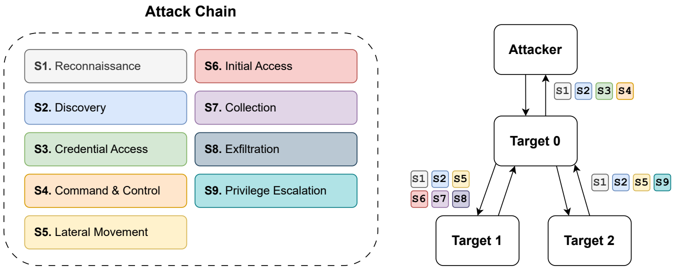

Fig. 5: **Using ConCap to reproduce complex attack chains envisioned in MITRE ATT&CK.** The attacker first compromises an exposed target before performing lateral movement to two internal hosts.

attacks. Specifically, we demonstrate that ConCap enables automatic reproduction of multi-host and multi-step attacks.

Threat Model. We assume a network consisting of four or more hosts, each having different privileges and containing different information. An attacker has obtained control of one host, and wants to expand their control and steal sensitive information. However, the attacker does not know the network topology, and must hence carry out reconnaissance activities to identify vulnerabilities that can be exploited to reach the intended goal. In other words, we envision an attack chain spanning multiple stages and hosts, which can be mapped to a range MITRE ATT&CK tactics [13]. Altogether, the attack encompasses the following stages (expressed via the well-known MITRE ATT&CK terminology): Gather Victim Network Information, Network Service Discovery, Brute Force, Internal Proxy, SSH Lateral Movement, Data Collection from Local System, Exfiltration Over C2, and Exploitation for Privilege Escalation. Our intention is showing how to use ConCap to generate (and automatically label) NetFlows of the entire attack chain—in an end-to-end fashion.

Implementation. To replicate the setting envisioned in our threat model, we defined the scenario in Listing 5 (in the Appendix). Such a scenario embeds the use case, schematically depicted in Fig. 5, in which an attacker first performs a port scan against an SSH server using nmap [72], then launches a brute-force SSH attack using Medusa [114]. Upon discovering valid credentials, the attacker uses SSH to set up a SOCKS proxy through the compromised host and scans additional internal hosts (a MySQL database and a vulnerable libssh server). Then, the attacker forwards internal ports via SSH tunneling to the local machine, allowing authenticated access and data exfiltration against the MySQL target and enabling the attacker to exploit a known vulnerability against libssh.

Considerations. In practice, when ConCap processes such a scenario, all of the aforementioned operations are automatically carried out. We also note that each target in this scenario is separately labeled with its function (e.g., "brute-force-ssh", "exfiltration", or "cve") and configured with specific network constraints (e.g., bandwidth and delay). At the end of the entire experiment, the OpenSSH server received 3635 network packets corresponding to 184 CICFlowMeter and 185 argus NetFlows. The MySQL server received 280 packets corresponding to 109 CICFlowMeter and 110 argus NetFlows. The libSSH server received 246 packets corresponding to 106 CICFlowMeter and 106 argus NetFlows. All NetFlows (which we provide in our repository [23]) are labeled according to the specifications of the scenario. Notably, by sharing this scenario (i.e., Listing 5), future researchers can inspect the entire traffic-generation process: this is important for open science and to address the skepticism around NIDS research [8].

C. Testing additional ML-based NIDS with ConCap

In our paper (§VI), we showed that ConCap can be used to test "simple" ML-based (e.g., RF or SVM) NIDS. Here, we expand our assessment by showing that ConCap's generated data can be used also by other families of ML-driven NIDS.

Please note that our goal is merely to show that ConCap's data can be used to train and test such additional methods: we do not seek to *(a)* propose new methods, *(b)* outperform/tweak existing ones, or *(c)* benchmark prior work.

Considered models. There are hundreds of papers proposing, or evaluating, methods to detect network intrusions via ML [2]. For the sake of this additional demonstration, we consider the methods considered in the recent USENIX'23 paper xNIDS [98]. Specifically, this work considers two families of deep-learning–based methods for intrusion detection: one reliant on *AutoEncoders* (drawn from [37, 115]), which we denote as AE-IDS; and another one reliant on *Recurrent Neural Networks* (drawn from [116, 117]), which we denote as RNN-IDS. Note that, altogether, the papers proposing these methods have thousands of citations on Google Scholar (as of December 2025) or are published in top-tier venues (e.g., S&P'22, NDSS'18). Then, to provide yet-another perspective, we also consider transformer-based approaches which empower *multimodal large-language models* (LLMs), inspired by the recent [109].

Implementation and Datasets. We take the code from [98] (which is publicly available) as a blueprint for our AE-IDS and RNN-IDS models; whereas, for LLMs, we consider: ChatGPT 5.1, and Gemini Flash 2.5 (both of which being the best freely-available commercial models as of December 2025). For both of these LLMs, inspired by [109], we will assess them with a zero-shot prompting strategy (with the prompt in Listing 7), as well as in an "zero-shot-with-augmentation" fashion (with the prompt in Listing 8).[6] For the evaluation dataset, we consider the same used in §VI-A, i.e., a mix of CICIDS17 and the data generated by ConCap referring to the ssh-patator attack. Such a setup is valid for the sake of our demonstration, given that the approaches we considered have been evaluated in completely different setups (e.g., the "outdated" NSL-KDD or the CIC-DoS2017 [118]).

Evaluation and Results. We replicate the experiments in §VI-A. Specifically, we train each model (AE-IDS and RNN-IDS) on 80% of the benign data in CICIDS17, as well as

[6] Note that the approach in [109] uses *two-dimensional plots* as input to the LLM, whereas we use ConCap's raw data. This is because we want to show that ConCap can be used off-the-shelf to test LLMs. However, there is nothing preventing one from generating the same two-dimensional plots of [109] (which simply show the source and destination IP for the x- and y-axes, and points indicate the amount of *exchanged bytes*—all of which being metrics derivable from ConCap's data.

on 80% of the data generated by `ConCap` for ssh-patator (launched with default options); and then we test it on the remaining 20% (benign and malicious). We also consider doing a mixed experiment by testing each model on 20% of the ssh-patator included in CICIDS17; as well as by creating another variant trained on 80% of the ssh-patator included in CICIDS17 and testing it on 20% of the ssh-patator generated by `ConCap`. For the LLMs, the zero-shot prompt does not have any training (barring that to develop the LLM itself), whereas the augmented-zero-shot prompt embeds a fine-tuning step done on the same data used to train the AE-IDS and RNN-IDS. We repeat our experiments five times to ensure consistency. Let us discuss the results of our experiments:

- *AE-IDS:* This experiment confirms our conclusions drawn from §VI-A. Specifically, when trained[7] on `ConCap`'s generated data (which is always malicious), the AE-IDS achieves tpr=0.999 on the testing partition of `ConCap`'s generated data, and tpr=0.987 on the testing partition of malicious data of the same attack included in CICIDS17; conversely, when trained on the malicious data of CICIDS17, the AE-IDS achieves tpr=0.955 on the testing partition of CICIDS17, and tpr=0.952 on the testing partition of `ConCap`'s generated data. The fpr is 0.478 on the former case, and 0.209 in the latter case: such an underwhelming result is because our AE-IDS uses the same thresholding mechanism used in xNIDS [98] (note that AutoEncoders are not classifiers: to use them in a classification task, one must specify a threshold on the reconstruction error. We used the one of xNIDS, which was derived on a different dataset. To improve the fpr, one can simply change the threshold). The different fpr is because the number of datapoints in the training set is different across the two experiments, but what is crucial is that, for this detector, the data generated by `ConCap` is essentially equivalent to that of CICIDS17, since the classification results on these two groups are statistically the same (verified with a t-test: $p < .05$).
- *RNN-IDS:* For this experiment, we report that, after 2 hours of training, the underlying RNN did not reach convergence. Therefore, given our previous experiment, as well as those in our main paper (§VI-A) showing that simple models (such as RF or SVM) can achieve near-perfect detection accuracy while requiring seconds to train (see Table VIIb), we conclude that using RNN-IDS on our dataset is not a wise choice and stopped this experiment. However, what matters is that the RNN-IDS was being trained, meaning that `ConCap`'s data was "compatible" with that expected in RNN-IDS, thereby validating the goal of these experiments (note that SaTML'26 call for papers explicitly states to be mindful of the computing resources spent in training ML models, and we believe that additional training would not have changed our conclusions).
- *ChatGPT:* When assessed in the "zero-shot" prompt, the model always classified each datapoint as benign. This is expected: the malicious datapoints are just repeated SSH attempts, which not necessarily imply malicious intent. However, by using the "augmented-zero-shot" prompt (which uses `ConCap`'s generated labelled data to "teach" the LLM that quick bursts of SSH connections are malicious), the model achieves a perfect classification accuracy (i.e., tpr=1.000 and fpr=0.000). We inspected the output provided by ChatGPT, and found that, to derive such a result, the model learned heuristics that enabled a clear separation of the benign from malicious datapoints. (note that these results align with those in [109]).
- *Gemini:* The results for Gemini 2.5 match those of ChatGPT.

Listing 7: Zero-shot prompt used to evaluate LLMs with `ConCap`'s data.

```
You are a cybersecurity analyst working in a
    Security Operations Center (SOC). You are
    analyzing a CSV file that contains NetFlow
    records.

Data Description:
Each row in the CSV represents a network flow with
    features such as: Src Port, Dst Port, Protocol,
    Flow Duration, Number of packets Your task is to
     classify each row as either: 0 = Benign 1 =
    Malicious

Your Tasks:
Determine whether the NetFlow record shows
    suspicious behavior. Output only 0 or 1 for each
     row (0 = benign, 1 = malicious) on a separate
    row and write to csv file.
```

Listing 8: Augmented zero-shot prompt used to evaluate LLMs with `ConCap`'s data. We augment the prompt in Listing 7 by instructing the LLM to use labeled data (in the X_train.csv file) as a guide for its inference.

```
You are a cybersecurity analyst working in a
    Security Operations Center (SOC). You are
    analyzing a CSV file that contains NetFlow
    records (X_test_shuffled.csv).

Data Description:
Each row in the CSV represents a network flow with
    features such as: Src Port, Dst Port, Protocol,
    Flow Duration, Number of packets Your task is to
     classify each row as either: 0 = Benign 1 =
    Malicious

Your Tasks:
Determine whether the NetFlow record shows
    suspicious behavior. Base your predictions on
    the provided labeled NetFlows (X_train.csv and
    y_train.csv). Output only 0 or 1 for each row (0
     = benign, 1 = malicious) on a separate row and
    write to csv file.
```

The code of these experiments is in our repository [23].

Takeaway. Our experiments show that `ConCap` can be used to develop and assess multiple families of ML-based NIDS. Moreover, our results further confirm that `ConCap`'s generated data can be functionally equivalent to that generated in other environments (e.g., that of CICIDS17).

[7] Training the AE-IDS required ≈10 minutes on our system.

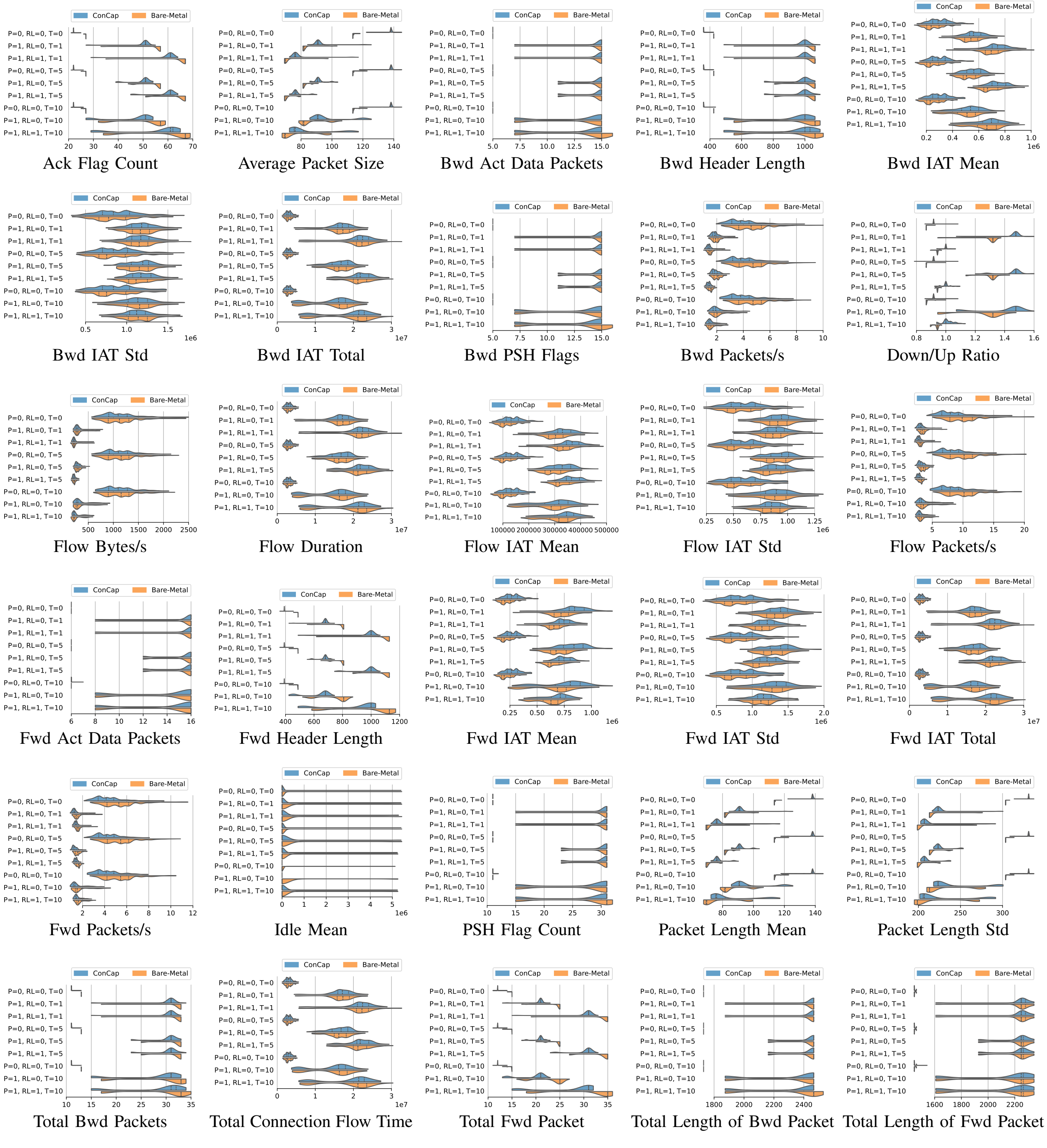

Fig. 6: Comparison of NetFlow feature distributions of a Patator brute-force SSH attack between ConCap and a real network.