5GProvGen: 5G Provenance Dataset Generation Framework

Amr Abouelkhair, Kiarash Majdi, Noura Limam, Mohammad A. Salahuddin, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
{aabouelkhair, kmajdi, mohammad.salahuddin, n2limam, rboutaba}@uwaterloo.ca

***Abstract*—The softwarization and virtualization of the fifth-generation (5G) cellular networks bring about increased flexibility and faster deployment of new services. However, these advancements also introduce new vulnerabilities and unprecedented attack surfaces. The cloud-native nature of 5G networks mandates detecting and protecting against threats and intrusions in the cloud systems. Additionally, the evolving cyber-threat landscape and the growing reliance on cellular networks for mission-critical tasks reinforce the need for robust security systems, which should be capable of detecting stealthy and zero-day attacks.**

Recent developments in Provenance-based Intrusion Detection Systems (PIDS) address these requirements. These host-based systems aim to analyze provenance graphs derived from system calls to uncover any deviation from the expected benign behaviour of the host. Provenance graphs are structured as holistic representations of the dependencies and causal relationships between digital objects, and hence they fit well in the Service-based Architecture (SBA) of 5G networks. However, deploying PIDS requires substantial datasets of provenance graphs collected from the relevant hosts. In this work, we propose a framework to generate provenance graphs datasets for a 5G core network. We provide an example dataset and evaluate the state-of-the-art PIDS in protecting a 5G network core from various threats.

***Index Terms*—5G, provenance, dataset, service-based infrastructure, GNN**

I. Introduction

Fifth-generation (5G) cellular networks bring forth many advancements over previous generations. At the forefront of those advancements are network virtualization and softwarization, which introduce more flexibility and expedite service deployment. This is achieved primarily by converting many 5G core Network Functions (NFs) into software micro-services. Notably, the implementations of these NFs can be sourced from an array of vendors, including third parties, following the 5G Service-based Architecture (SBA). This significantly alters the circle of trust, increases the 5G core network (5GCN) attack surface and introduces new vulnerabilities.

The increased importance of 5GCNs in supporting mission-critical services, coupled with the evolution of advanced persistent cyber threats (APTs), highlights the necessity of accurate intrusion detection systems (IDS). A critical example is Operation Dianxun, which targeted numerous 5G providers worldwide with a privilege escalation APT [1]. Additional examples of such APTs are documented in the MITRE FiGHT attack taxonomy [2].

Furthermore, traditional network flow-based IDS commonly deployed in cellular networks are no longer sufficient. Detecting low and slow APTs from network traffic has become much more elusive [3]. APTs can take days and consist of multiple minor steps. With the enormous growth of traffic in 5GCNs, APTs can easily disappear within benign traffic, effectively neutralizing conventional IDS [4]. This inadequacy is largely due to conventional IDS lacking the contextual and temporal awareness between system events. Therefore, having IDS analyze system calls on the network core is essential.

To leverage contextual and temporal information about system events, researchers began structuring system call information into graphs, known as provenance graphs. This led to the evolution of a new class of Provenance-based Intrusion Detection Systems (PIDS). In PIDS, representing system events as graphs enables the use of more sophisticated Machine Learning (ML) models, particularly Graph Neural Networks (GNNs) [5].

The advancements of PIDS, combined with the developments in 5G networks, strongly support utilizing PIDS within 5GCN. With service providers renting compute resources, tenants start sharing physical hardware. In theory, each tenant has separate virtual resources, but virtual isolation is not always guaranteed. Therefore, if an APT is ongoing from one of the tenants or their customers, the host must detect the attack quickly and accurately, to prevent it from impacting other tenants. PIDS show a lot of promise in this area.

Unfortunately, there remains a need for a dataset that describes the benign behaviour of 5GCN for the PIDS to learn from. While there are many efforts to provide datasets for ML model training in 5G networks, these datasets predominantly focus on the networking perspective rather than a system perspective [6]–[8]. Most available datasets consist of labelled PCAP files containing network packets rather than system-level events.

Nevertheless, there are a few datasets that facilitate the training of ML models for provenance-based intrusion detection, some more sophisticated than others. However, none of them consider the 5G use case or provide a representation of benign activity on a host running 5G core services. Since most PIDS operate as anomaly detectors, there is a dire need to capture benign activity in 5GCN. Furthermore, while there are some frameworks for generating datasets whether based on network traffic packets or system logs [9], our survey of the literature has not identified a single comprehensive framework for generating provenance graphs for training PIDS, particularly in the context of 5G networks.

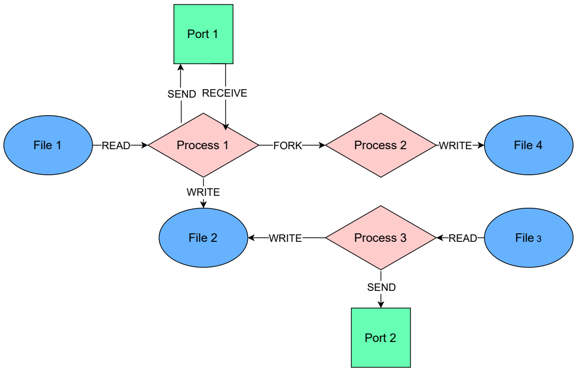

Fig. 1. An abstract example of a provenance graph

While there may be some individual components for provenance graph collection, with CamFlow [10] being the most notable, there remains a significant gap in generating comprehensive datasets. Additionally, installing CamFlow on most operating systems (OS) is non-trivial, making data collection less accessible than desired. The main contributions of our work are as follows:

- We design and develop a flexible framework, called 5GProvGen, to generate a dataset of provenance graphs for 5GCN. To the best of our knowledge, this is the first framework for generating provenance datasets.
- We generate a 5GCN provenance dataset using our framework, which includes an array of benign and malicious activities to train ML models for provenance-based PIDS. This is the first dataset of provenance graphs aimed at PIDS for detecting threats in 5GCN.
- We evaluate prominent, state-of-the-art PIDS on our generated 5GCN provenance dataset to showcase its efficacy in facilitating the detection of several APTs.

We make the source code for 5GProvGen, the dataset generation setup, and the provenance dataset for 5GCN publicly available[1].

The remainder of our paper is organized as follows. Section II summarizes the necessary background, covering the evolution of provenance graphs and PIDS. Section III discusses the related works on datasets and frameworks. Section IV details the design of our framework and the reasoning behind our design choices. Section V describes our dataset generation pipeline, while Section VI evaluates state-of-the-art PIDS on our dataset. Section VII instigates future research directions, and we conclude the paper in Section VIII.

II. Background

In this section, we provide a high-level overview of provenance graphs, their collection, and the intrusion detection systems that leverage provenance graphs.

A. Provenance graphs

Provenance was initially used to track data integrity in databases [10], [11]. Whole-system Provenance graphs were introduced as an iteration on provenance to capture states of system entities and file history [12]. These graphs offer a holistic view of the system from initialization through its life cycle, providing a significant amount of information that could be leveraged to enhance the performance of ML-based intrusion detection techniques [10].

System calls are the source of truth for provenance graphs. While system calls and provenance graphs contain similar information (e.g., system call type, source process) [13], provenance graphs organize the information to facilitate pattern recognition for ML models. In a provenance graph, each node represents a system entity, such as processes, files, or network ports. The edges represent interactions between these entities, such as reads, writes, and forks, along with the progression of such events over time. This format allows ML models to capture a more comprehensive contextual view of each system entity and their behaviour, resulting in a better prediction of their intentions [14]. Figure 1 shows an abstract example of a provenance graph.

B. Provenance collection

The generation of provenance graphs is the first step in utilizing PIDS. While different OS may provide built-in tools to monitor and record system calls, these calls are not formatted as provenance graphs, and converting them is non-trivial. Additionally, the system call information collected by the OS might not be comprehensive. We found that built-in tools (e.g., auditD) occasionally generate events that are not complete and do not indicate the source and destination of an edge. Furthermore, when such tools operate in the user space they lack the required access to information that is crucial to link system calls to their respective containers in a containerized environment.

An example of a built-in auditing tool is auditD. It is available as part of the Linux security module (LSM) in Linux-based OS [13]. Although it comes pre-installed with most Linux builds and is relatively easy to run, the Linux audit suite has its drawbacks. Running auditD causes significant overhead, a major consideration in a cloud environment that can impact providers' revenue [15]. Furthermore, while auditD is easy to configure and use for collecting system calls, the information collected is incomplete (e.g., missing information within each event), making the construction of provenance graphs from system logs a substantial effort. This is especially noticeable when compared to more comprehensive whole-system provenance collection tools. Whole-system provenance involves tracking and recording all system-level events and interactions, providing a detailed and high-fidelity history of system activities, states, and data flows, which aids in analyzing system behaviour over time [12].

[1] https://github.com/abouelkhair5/5GProvGen

TABLE I
DIFFERENT PIDS IN THE RECENT LITERATURE

Model	Technique Used	Output	Granularity	Publication year
Unicorn [16]	WL labeling Scheme + Histogram Sketching + Clustering	Timestamp	Temporal Graph Snapshot	2020
KAIROS [17]	TGN encoder + MLP Decoder	Edges	A summary graph of suspicious edges	2023
Flash [18]	Word2Vec + GNN Representation Learning	Nodes	Suspicious Nodes	2024

On the other hand, the flagship whole-system provenance collection tool is CamFlow. It offers an array of features unavailable in built-in tools. CamFlow is designed to capture provenance with each logged event containing all the necessary information to map it to an edge in the provenance graph without further preprocessing. Additionally, CamFlow is directly integrated into the system's kernel rather than the OS. This is especially useful in containerized environments, where it captures more information to determine which system calls belong to which container. This is crucial in 5GCN since most NFs are deployed inside containers. Finally, CamFlow offers support for distributed environments out of the box, which is important in a cloud environment where resources are not necessarily on a single host.

More sophisticated provenance collection tools have been released as part of The Defense Advanced Research Projects Agency (DARPA) Transparent Computing (TC) program, designed to minimize overhead and maximize coverage. However, these tools are not publicly available and, therefore, were not considered in this work.

C. PIDS

Detecting anomalies in whole-system provenance graphs is challenging due to the properties of these graphs. Provenance graphs are dynamic, with nodes and edges timestamped with creation and expiry time. They are also heterogeneous, containing multiple node and edge types. These characteristics complicate the use of many ML approaches that handle simpler graphs [19].

The heterogeneity of provenance graphs can be addressed by mapping different node and edge types to latent vector spaces, the resulting vectors, called *embeddings*, can then be used in the graph learning process [20]. An embedding is a mapping of high-dimensional data, such as nodes and edges in a graph, into a lower-dimensional vector space in a way that preserves the inherent properties and relationships of the original data [21]. Advances in Graph Neural networks (GNNs) have further improved the handling of such heterogeneity [22].

Capturing the temporal dynamics in provenance graphs has been more elusive. Initial attempts focused on simplifying the problem by reducing the granularity of the sub-graph to be evaluated [23]. The Weisfeiler-Lehman (WL) graph isomorphism test [24], an algorithm that iteratively relabels vertex labels based on their neighbourhoods, is used to determine whether two graphs are isomorphic. This is repeated until either stable labelling is achieved or a difference is detected between the graphs. For example, early PIDS considered temporal snapshots and labelled them with the WL-test [24]. The distribution of the most common labels in the provenance graph is then used as a baseline to determine any anomalous sub-graphs [16].

Recent advancements in GNNs, such as Graph Attention Networks (GATs) [25] and Temporal Graph Networks (TGNs) [26], have significantly enhanced the ability to capture temporal dynamics. The embeddings used by these models contain information not only about the events impacting each node but also about when they occur and how far apart they are. These models can analyze finer granularity interactions within the graph. By leveraging these advancements, newer PIDS can generate sophisticated embeddings for nodes and edges, improving the detection of suspicious activities [17], [18]. Table I shows some recent PIDS along with their employed approaches. We discuss their details in Section V-D.

III. RELATED WORKS

To date, no 5G provenance dataset is available for training ML models for threat detection. Most efforts in the 5G context have focused on Quality of Service (QoS) Key Performance Indicators (KPIs) rather than security-oriented datasets [6]–[8]. However, a few attempts have been made to generate 5G datasets aimed at network security. For instance, 5G NIDD [4] shares a similar motivation to our work. This work leveraged a 5G testbed that closely mirrors a production environment, and generated a rich dataset of benign behaviour. However, it only focused on providing network flows in PCAP files.

In the realm of provenance graphs, several major datasets have been leveraged for provenance-based research. Foremost among these are the datasets provided by DARPA as part of the TC Project. DARPA's TC project included numerous exercises, such as Engagement 3 (E3), Engagement 5 (E5), and Operationally Transparent Cyber (OpTC) [27]. Each exercise involved multiple groups, such as CADETS and THEIA, responsible for tracking and tagging system activity. These datasets were made publicly available by DARPA. However, none of the benign behaviors available in the DARPA dataset resemble the behavioral patterns in a 5G network.

Apart from DARPA's contributions, provenance datasets have been relatively sparse, often limited to simpler datasets released alongside specific models. For example, the authors of Unicorn [16], a prominent open-source provenance graph-based anomaly detector, published their dataset. Another example is Streamspot [28], where the published dataset has become a standard for evaluating provenance-based anomaly detection models due to its proprietary yet simple format. The widespread use of Streamspot's dataset facilitates analytical comparison across different intrusion detection approaches.

AutoCES [9] was one of the inspirations for our work. AutoCES aimed to generate datasets for system call logs, but it had several shortcomings. All system call logs were collected using the Linux audit suite, which introduces significant overhead. Moreover, the events within the dataset lack the critical information needed to convert the logs to a provenance graph unlike events collected by a tool such as CamFlow. Finally, AutoCES primarily focused on an IoT scenario rather than a 5GCN.

With this work, we aim to fill the gap by providing a dataset of provenance graphs generated by a 5GCN. This should enable better training of PIDS to be utilized in 5GCN. We also pave the way for other researchers to generate further datasets in different scenarios, making training PIDS more accessible for other use cases.

IV. 5GProvGen Framework

In this section, we detail the design of our 5GProvGen framework. 5GProvGen consists of four main components, namely *Orchestrator*, *Collector*, *Activity Generator*, and *Annotator*, as shown in Figure 2. We discuss each of these components and their respective functions. We also expose our labelling (i.e., annotation) approach for the generated datasets.

A. Orchestrator

The orchestrator is the backbone of the 5GProvGen framework, managing the entire pipeline of dataset generation. It controls the configuration, initialization, and termination of provenance collection, ensuring that the necessary data is collected accurately while minimizing overhead. By handling these tasks, the orchestrator allows for a seamless transition between different stages of data collection and activity simulation, making the dataset generation process quite hands-off. It also ensures that the environment is properly set up before any activities commence, and systematically shuts down processes to ensure no data is lost or corrupted.

In addition to managing the collection process, the orchestrator schedules the sequence of necessary activities. It triggers the activity generator to simulate various services and attacks at different intervals, facilitating a dataset that includes a wide range of scenarios. This scheduling capability allows the orchestrator to create a realistic network environment that closely mimics real-world usage patterns. The orchestrator also handles the management and storage of system calls, ensuring that all collected data is organized and easily accessible for further analysis.

A critical function of the orchestrator is to keep a detailed log of all activities and their corresponding timestamps. This includes recording the start and stop times of each activity, as well as any notable events that occur during the simulation. These timestamps are essential for accurately labelling the dataset and separating provenance graphs that contain either benign or malicious behaviours. By maintaining precise records, the orchestrator facilitates the post-processing and analysis phases, allowing the annotator to label graphs and correlate them with the system's behaviour. This logging ensures the dataset's reliability and usefulness for training and evaluating ML models for intrusion detection.

We also built the orchestrator to be modular and configurable. Both the collector and activity generator can be modified separately without needing to update the orchestrator, as long as they meet their class definitions. The orchestrator is also configurable to change the duration of the dataset, the number of benign and malicious activities, their types as well as the schedule strategy for each.

B. Collector

We use CamFlow as our primary tool for whole-system provenance collection. The rich semantics of the events collected by CamFlow facilitate building provenance graphs from the system calls. Many of the existing models (e.g., Unicorn) support CamFlow-generated logs. We write the logs in the W3C format, which is supported by CamFlow out of the box.

Importantly, CamFlow is integrated at the kernel level, allowing it to distinguish activities between different containers. This is in contrast to tools such as auditD, which operate at the OS level. Furthermore, CamFlow's support for multiple hosts is crucial to extend our framework in the future, given the cloud-native nature of 5GCN.

We configure CamFlow to collect all system call types, as deeper GNN models can capture patterns and detect minor cues that could signify anomalous behaviour. For ML models that do not consider all node and edge types, we filter the data in post-processing before model evaluation.

C. Activity Generator

The activity generator is designed to be extendable by any user of the framework, allowing the addition of various types of activities to generate a dataset. We provide example classes of both benign and malicious activities that can be used as templates. As long as these activities implement the activity interface, they can be integrated into our activity generator and work seamlessly with the orchestrator. Furthermore, each activity class is designed to consume a log file where the necessary timestamps for labelling the data are written, ensuring accurate and organized dataset generation.

D. Annotator

The granularity of labelling in our dataset includes both whole graphs and temporal snapshots as small as one minute. This approach primarily relies on timestamps marking the start and duration of attacks to recognize anomalous behaviour. Time-based labelling is a standard method in the literature (e.g., AutoCES, Streamspot) due to the sheer volume of events generated by provenance collection tools, which makes it challenging to label specific events or processes directly [9].

Currently, provenance collection tools do not provide mechanisms to highlight specific events, and narrowing down malicious activities based solely on processes is impractical. A single process can generate thousands of events, with only

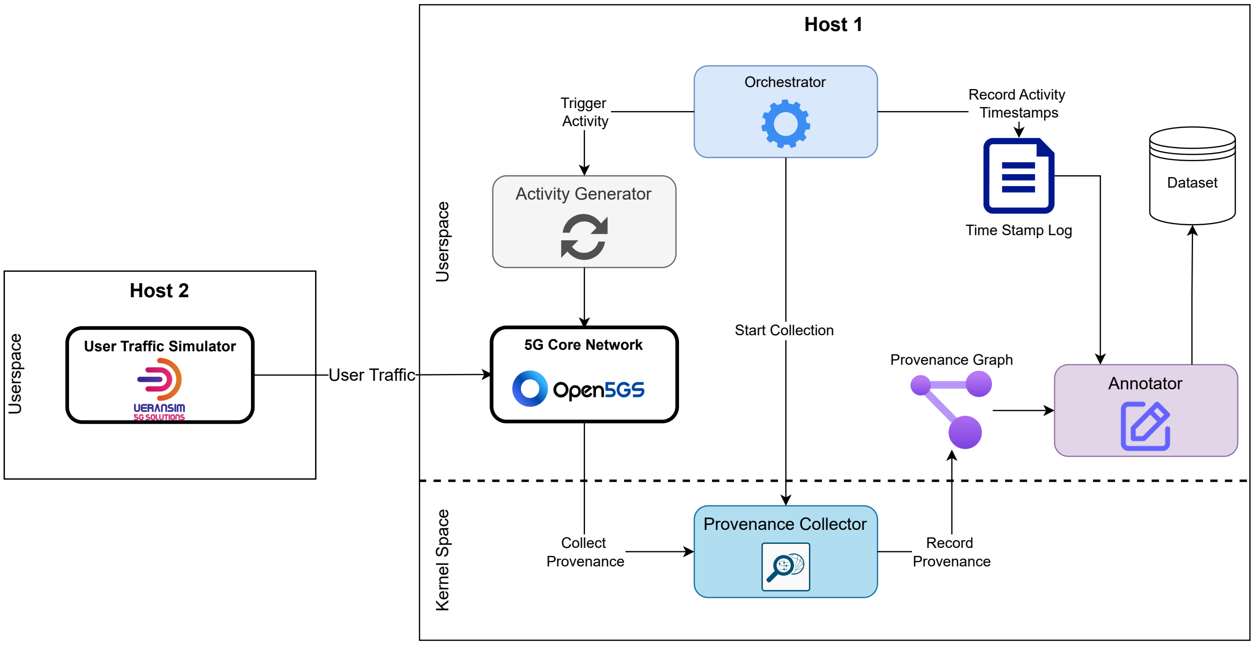

Fig. 2. An overview of 5GProvGen framework components and dataset generation setup

a minor fraction being malicious. This limitation necessitates the use of time-based labelling to manage the data effectively. The only datasets offering finer-grain labels are those provided by DARPA, which benefit from extensive resources. However, DARPA's tools are closed-source, and their level of resource allocation is not available to the broader research community.

To address these constraints, we meticulously record timestamps for the initiation and termination of each activity, whether benign or malicious. For activities involving multiple steps, we attempt to capture timestamps for each stage. These precise timestamps allow us to segment the data into discrete time windows, which can then be labelled accordingly. This method ensures that each time window reflects the activity occurring within it, providing a clear and organized dataset for training and evaluating PIDS. By maintaining detailed logs of activity timings, we enhance the dataset's usability for anomaly detection.

V. Dataset Generation

In this section, we detail our setup to generate the provenance dataset for 5GCN. We present the hardware and software leverage, benign and malicious activity implementations, and the pipeline to generate and parse the dataset.

A. Setup

We leverage a high-performance compute node, which includes a dual Intel Xeon Silver 4314 CPU @ 2.40GHz, 1TB memory, and 2x Nvidia A40 GPUs, to generate the majority of the activity (e.g. running the 5GCN) as well as conduct the provenance collection. This node is depicted as Host 1 in Figure 2. Host 2 in the figure is an Intel NUC Skull Canyon, powered by an Intel i7 CPU @ 3.50GHz and 16GB memory. Host 2 is exclusively to run the User Equipment (UE) and gNB simulation for it to not be included in the dataset.

We utilize Kubernetes to orchestrate and control our containerized cluster, ensuring a modular and flexible setup. Our 5G testbed leverages a modified version of Open5GS network core, which allows each NF to run in a separate container within the Kubernetes cluster. [29] This configuration ensures efficient management and scalability of the NFs, maintaining the testbed environment as close as possible to the real-world setting. We also use UERANSIM to simulate a gNB and an array of UEs.

In the network core, we deploy two slices with different bandwidth requirements. These slices share some NFs but have independent User Plane Function (UPF) and Session Management Function (SMF). By separating the UPF and SMF for each slice, we can observe the performance and interactions of different service types within the network core, providing a more pertinent dataset for training ML models. Finally, we attach 10 UEs to each slice. These UEs run on a different machine. This separation is crucial to prevent cross-contamination of activities, and allow for training ML models exclusively on the network core behaviour. Figure 2 depicts a high-level overview of our dataset generation setup.

B. Benign Activity

To generate benign data, for each UE, we deploy a custom random activity generator that selects different activities at random intervals. The available activities include browsing the web, downloading files, and streaming YouTube videos. The activity generator aims to mimic normal human behaviour by varying both the types of activities and their timing. This approach ensures that the generated traffic patterns are dynamic and realistic, closely resembling real-world user behaviour. The corresponding traffic flowing through the 5GCN results in the different NFs interacting and handling user requests, which are recorded.

C. Malicious Activities

To generate the non-benign portion of the dataset, we focus on attacks that target containerized systems, networks, or any specific module of the 5G network. We chose an array of attacks to validate that ML models trained on our dataset's benign activity can detect varying malicious patterns.

- *Denial of Service:* An attacker may overload the host server by consuming more computational resources than authorized. As a result of this attempt, the host may deny service to benign users, referred to as *Resource Exhaustion*. This attack proves possible in container systems where a container with limited privileges can abuse host resources, such as CPU and RAM. We execute a script from inside the container, exhausting the resources of the container's host in this malicious activity [9]. This attack relates to the endpoint denial of service technique that falls under the impact category of the MITRE taxonomy [2].
- *Container Escape:* Containers are well-known for creating idealized subsystems that share resources with the host. These resources include, but are not limited to processors, memory, and storage. Containers may need to be isolated from the host system since the container user may not be the host's administrator. The presence of certain vulnerabilities in the creation of the container allows the non-privileged user to execute commands as the host administrator. This is referred to as *Host Execution*. Primarily, the vulnerability is due to the container being over-privileged, mounting the host file system, and running commands that are not normally permitted on the host. In this malicious activity, we create a file in the host file system, leading to password bypassing for a privileged user, and escalating privileges of the attacker from container sudoer to host sudoer [9]. This attack combines elements from the privilege escalation and the credential access categories of the MITRE FiGHT taxonomy [2].
- *Network Service Scanning:* Every service on the Internet is accessed via its host IP address and service port. However, the attacker is not always aware of what services a host offers, and which ports the services are running on. In this malicious activity, in an enumerative manner, the attacker attempts to communicate with the host IP address via every possible port number and takes a record of which ports are open, filtered, or closed. This attempt is referred to as *Port Scanning*. In this malicious activity, we use the nmap software to scan ports on one of the 5G Access and Mobility Management Function (AMF), which is a service normally exposed to the Internet [2].

TABLE II
GENERATED SUB-DATASETS AND THEIR COMPOSITION

Name	Duration	# of Events	Size	Types
Benign-1	24 Hours	300M	131 Gbs	All benign
Benign-2	36 Hours	45B	1.7 Tbs	All benign
Hybrid-1	25 Hours	310M	140 Gbs	Hybrid
Hybrid-2	36 Hours	45B	1.8 Tbs	Hybrid
Malicious-1	1 Hour	6M	3.4 Gbs	All malicious
Malicious-2	4 Hour	30M	15 Gbs	All malicious

- *Network Function Service Discovery:* In a network, an attacker may be concerned with how the devices in the network are connected. In a 5G core, connections are virtualized between containers of the cluster to implement the architecture of the network. *Network Mapping* is the attacker's attempt to identify the connections between the different devices in a network. In this malicious activity, we use the nmap software to reveal the modules and IP addresses of a 5G core architecture as a network user Both this attack and the previous one map into the reconnaissance category of the MITRE taxonomy [2].
- *Fraudulent UE Registration:* The overexposure of the Unified Data Management (UDM) may lead to manual requests. Failure to filter such requests may introduce a vulnerability that would allow an attacker to send manual requests using curl, for example, to *Register Unauthorized UE*. In this malicious activity, we reproduce the attack by making a POST request to the UDM service as a fake AMF, asking to register a malicious UE. This attack falls into the impact and execute categories of the MITRE taxonomy [2].

D. Pipeline

We run multiple collections of varying durations to provide different amounts of data for ML models to learn benign behaviour. Our collection starts by deploying the core and monitoring it for a few minutes. Once the core is up and running we enable the whole-system provenance collection for CamFlow. We do not record the core deployment by default, as it is a small subset of the benign behaviour and is not the expected standard behaviour. However, we provide a configuration flag to include it in the collection.

Once the collection starts, we spin up an instance of the gNB and the UEs. We do include the registration process of each UE, as this is expected to happen frequently during the normal operation of a 5G core network. We also drop connections for a few UEs at random and re-register them. We intentionally run the UEs and the gNB on a different host to collect the activity of the 5GCN exclusively. Figure 3 provides an overview of our dataset generation pipeline.

E. Description

We have collected 6 sub-datasets, each ranging in duration from 1 hour to 36 hours. These sub-datasets contain 2 pure

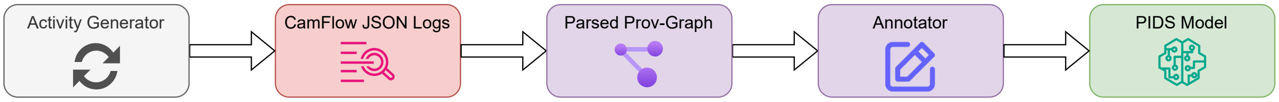

Fig. 3. An overview of the data pipeline

benign collections, 2 pure malicious, and 2 benign behaviours with attacks being triggered at random time stamps. All the datasets are available in the W3C format as the default output of CamFlow. We also provide the parsed version to match that of Unicorn [16] as it consumes much less disk space and facilitates the processing of the provenance graph with other models (e.g. Flash, KAIROS). The dataset description and the statistics about each collection are available in Table II.

VI. Dataset Evaluation

In this section, we describe how we feed our generated data to the state-of-the-art PIDS and evaluate their performance. We cover the challenges faced when parsing and processing the data as well as the different parameters we evaluate against.

A. PIDS models

A summary of the models we chose to evaluate and their respective approaches is presented in Table I. Unicorn is one of the earliest PIDS that attempts to capture the temporal and contextual dynamics of system calls through a provenance graph. It offers the most coarse granularity out of the evaluated models as it produces an anomalous temporal snapshot that could include thousands of nodes. However, it has been the standard baseline for many other models. Therefore, we chose it to be our starting point. Furthermore, Unicorn is one of the best-maintained open-source PIDS currently available.

Flash is a more recent attempt at leveraging GNNs to detect anomalies in provenance graphs. Unlike Unicorn, Flash provides an output with finer granularity, highlighting all the suspicious nodes that display anomalous behaviour. To even the ground for comparing Flash versus Unicorn, we follow the same approach as the authors of Flash, where they determine whether a snapshot or a sub-graph is suspicious based on the number of suspicious nodes it contains, i.e., they use a threshold for the number of suspicious nodes a graph can have before the whole graph is considered suspicious.

KAIROS is also a fine-grained PIDS, which highlights anomalous edges. This is the finest granularity we consider, as an edge pertains to a singular interaction between entities. Similar to Flash, we try to even the comparison with Unicorn and attempt to determine whether a graph is suspicious based on the number of suspicious edges it contains. This not only facilitates the comparison with Unicorn but also acts as a common ground as Flash and KAIROS would otherwise be difficult to compare.

There are other models (e.g., CAPTAIN [30]) that have not been included in this comparison. Primarily, we prioritized models that rely on unsupervised learning as it requires only benign behaviour for training and enables detecting zero-day attacks. This is important since all of our training datasets are attack-free. We also prioritized models with a robust open-source implementation, as we could not verify the accuracy of our implementation to mention closed-source implementations of other authors. Models like Prographer [31] report better results, however, the unavailability of their source code prevents us from confidently reproducing their results.

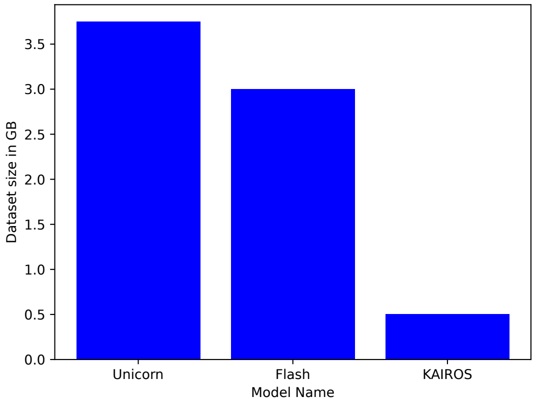

Fig. 4. Training dataset size needed to achieve the best performance

B. Parameters

During the evaluation of different models on our dataset, we noticed that all of the models perform reasonably well in detecting anomalous behaviour, demonstrating perfect recall in most cases. However, the main challenge was minimizing the false positive rate and maximizing precision. Therefore, we evaluated the PIDS with different collection durations and time window sizes, used for splitting the dataset into chunks before feeding it to the models. However, we did not evaluate with different hyper-parameter settings to stay as close as possible to the authors' implementations of their respective models.

Another goal was to evaluate the amount of data needed for model convergence, along with the overhead and runtime for each model during training and inference. While training can be done offline, it was important to look at the amount of training data, in case there is data drift and model retraining is needed. The inference time was also incredibly important, as these models will ultimately be deployed to monitor the network and infer anomalies as close to real time as possible.

C. Results

The goal of this evaluation is to demonstrate that the benign behaviour generated by our framework is learnable. This can

TABLE III
BEST PERFORMANCE OF EACH MODEL

Model	Window Size (mins)	Accuracy	Precision	Recall	F1-score	Training Time (hours)	Inference Time (mins)
Unicorn [16]	20	60.87%	12.90%	100.0%	0.2286	5	22 (6 time windows)
Flash [18]	0.05	86.67%	87.50%	83.33%	0.6757	0.5	10 (330 time windows)
KAIROS [17]	0.5	100.0%	100.0%	100.0%	1.0000	1	58 (100 time windows)

be evaluated by considering the accuracy, precision, and recall of each model. We prioritized precision and recall as the goal is to successfully detect all malicious activities while minimizing false positives. Therefore we relied on the F1-score to determine the best performance of each model as it combines both precision and recall. Table III summarizes the best performance achieved by each model.

Unicorn required the most amount of training data and the longest training time. In the original work, the authors evaluated 3 days of data collected using CamFlow. However, we trained it with both 24- and 36-hour collections. For the 24-hour collection, we evaluated Unicorn with different time windows to see how the accuracy changes with smaller time windows. We also compare run time for different time windows. In the original work, the authors split the 3 days into 125 graphs with each graph lasting around 35 minutes. In our evaluation, we achieved the best accuracy when splitting the 24-hour collection into 70 windows, each lasting about 20 minutes. As mentioned before, the main challenge was to reduce the false positives, while false negatives were non-existent across different time window sizes.

Both Flash and KAIROS also showed very high accuracy. For KAIROS, we built custom Jupyter notebooks that are heavily based on the author's original Python files. Our notebooks parse our dataset and store the parsed nodes and edges in a Postgres database as required by the author's implementation. From there, the rest of the code is the author's default implementation. Our evaluations consistently resulted in accuracy and precision above 95%. It was non-trivial to produce summary graphs, due to our dataset being labelled on a graph level rather than the edge level, similar to the datasets provided by DARPA. Furthermore, all the examples of summary graphs provided by the authors were generated on the DARPA datasets, which are labelled on the event level.

With Flash, the only caveat was that when fed the whole dataset, Flash would run for multiple days and produce very high false positive rates marking almost every benign window as malicious. This was despite our dataset being similar in format and size to the dataset provided by Unicorn, which the authors evaluated their model on. However, upon discussion with the authors, we were advised to sample the dataset and use much smaller graphs for training and inference, which resulted in a similar performance as reported by the authors.

VII. FUTURE DIRECTIONS

A. Distributed environment

One of the main advantages of CamFlow is its ability to collect provenance across multiple hosts in a distributed environment. However, for the sake of simplicity and to facilitate recreating our setup, we focus on a single-host deployment. We leave the exploration of multi-host 5G core deployment to future work, which would require the utilization of a multi-node cluster to host the environments containers and the provenance collected from the different hosts to be combined. This would allow us to study the impact of distributed provenance on model performance. It would also provide an opportunity for online threat detection in a distributed infrastructure.

B. Better models

The more recent models we evaluated performed very well, leaving little room for improvement in detection accuracy. However, there are still areas to explore to decrease the human effort required for post-detection analysis. While some models, such as KAIROS, provide reasonably succinct summary graphs, there is no reasoning provided on which entities were included in the summary graph. Being able to pinpoint dimensions in the embeddings that caused an anomaly and decoding it back to real-world features or certain events in a node's history as a cause for suspicion, could be incredibly helpful.

Furthermore, some pointers towards the type of attack represented by the summary graph, the vulnerabilities that were exploited, or the customer information in jeopardy, could provide meaningful insights for mitigating the threat. This could potentially be achieved by building a multi-modal system, where the generated summary graph is further fed to a classification model that provides such insights.

C. Minimize overhead

Although CamFlow entails much less overhead compared to auditD, it still results in substantial overhead during provenance graph collection. This can be mitigated by restricting the collected edges and nodes through a prerecorded configuration file. Nevertheless, high computing resources are needed to collect a dataset of the magnitude we achieved. Therefore, it is worth investigating other kernel tools that can collect system calls while retaining the ability to identify which system call belongs to which container.

VIII. Conclusion

In this work, we presented 5GProvGen, a framework for generating provenance datasets. We argue for the use of provenance-based IDS in the containerized 5G core. Hence, we also generate a provenance dataset for 5GCN. We evaluate the generated dataset using state-of-the-art PIDS and demonstrate its efficacy in facilitating ML models in detecting several APTs. To the best of our knowledge, this is the first modular framework to generate a dataset of provenance graphs for PIDS. We make the source code and the dataset publicly available to facilitate further research in securing 5GCN.

Acknowledgement

This work was supported in part by funding from the Innovation for Defence Excellence and Security (IDEaS) program from the Department of National Defence (DND).

References

[1] A. Yurchenko, "Chinese apt targets 5g providers around the globe," https://socprime.com/blog/chinese-apt-targets-5g-providers-around-the-globe/, March 23, 2021, july 7, 2024.

[2] The MITRE Corporation, "Fight: 5g hierarchy of threats," https://fight.mitre.org, 2023.

[3] T. Hammouchi, D. Rupprecht, and K. Kohls, "Intrusion detection system for 5g core systems," Ph.D. dissertation, Radboud University Nijmegen, 2023.

[4] S. Samarakoon, Y. Siriwardhana, P. Porambage, M. Liyanage, S.-Y. Chang, J. Kim, J. Kim, and M. Ylianttila, "5g-nidd: A comprehensive network intrusion detection dataset generated over 5g wireless network," *arXiv preprint arXiv:2212.01298*, 2022.

[5] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, "Provenance-based intrusion detection systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, dec 2022. [Online]. Available: https://doi.org/10.1145/3539605

[6] U. Ali, G. Caso, L. De Nardis, K. Kousias, M. Rajiullah, Ö. Alay, M. Neri, A. Brunstrom, and M.-G. Di Benedetto, "Large-scale dataset for the analysis of outdoor-to-indoor propagation for 5g mid-band operational networks," *Data*, vol. 7, no. 3, p. 34, 2022.

[7] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond throughput, the next generation: A 5g dataset with channel and context metrics," in *Proceedings of the 11th ACM multimedia systems conference*, 2020, pp. 303–308.

[8] K. Kousias, M. Rajiullah, G. Caso, U. Ali, O. Alay, A. Brunstrom, L. De Nardis, M. Neri, and M.-G. Di Benedetto, "A large-scale dataset of 4g, nb-iot, and 5g non-standalone network measurements," *IEEE Communications Magazine*, 2023.

[9] J. Pope, F. Raimondo, V. Kumar, R. McConville, R. Piechocki, G. Oikonomou, T. Pasquier, B. Luo, D. Howarth, I. Mavromatis, P. Carnelli, A. Sanchez-Mompo, T. Spyridopoulos, and A. Khan, "Container escape detection for edge devices," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 532–536. [Online]. Available: https://doi.org/10.1145/3485730.3494114

[10] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 405–418.

[11] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? what form? what from?" *The VLDB Journal*, vol. 26, pp. 881–906, 2017.

[12] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-fi: collecting high-fidelity whole-system provenance," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 259–268.

[13] SUSE Linux Enterprise, "The linux audit framework," http://www.suse.com/documentation/sled10/pdfdoc/audit_sp2/audit_sp2.pdf, 2008.

[14] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat detection and investigation with system-level provenance graphs: A survey," *Computers & Security*, vol. 106, p. 102282, 2021.

[15] L. Zeng, Y. Xiao, and H. Chen, "Auditing overhead, auditing adaptation, and benchmark evaluation in linux," *Security and Communication Networks*, vol. 8, no. 18, pp. 3523–3534, 2015.

[16] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," *arXiv preprint arXiv:2001.01525*, 2020.

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

[18] 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)*. IEEE Computer Society, 2024, pp. 139–139.

[19] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 012–12 038, 2021.

[20] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 793–803.

[21] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE transactions on knowledge and data engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[22] L. Yu, J. Shen, J. Li, and A. Lerer, "Scalable graph neural networks for heterogeneous graphs," *arXiv preprint arXiv:2011.09679*, 2020.

[23] C. D. Barros, M. R. Mendonça, A. B. Vieira, and A. Ziviani, "A survey on embedding dynamic graphs," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–37, 2021.

[24] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *nti, Series*, vol. 2, no. 9, pp. 12–16, 1968.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[26] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[27] G. Ouyang, Y. Huang, and C. Zhang, "Analyzing the usefulness of the darpa transparent computing e5 dataset in apt detection research," in *International Conference on Computer, Artificial Intelligence, and Control Engineering (CAICE 2022)*, vol. 12288. SPIE, 2022, pp. 400–409.

[28] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1035–1044.

[29] N. Saha, N. Shahriar, R. Boutaba, and A. Saleh, "Monarch: Network slice monitoring architecture for cloud native 5g deployments," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–7.

[30] L. Wang, X. Shen, W. Li, Z. Li, R. Sekar, H. Liu, and Y. Chen, "Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection," *arXiv preprint arXiv:2404.14720*, 2024.

[31] 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)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 4355–4372. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/yang-fan