Building Cyber Security Datasets and Tools for Anomaly Detection

Andi Ari

Master of Science
School of Informatics
University of Edinburgh
2023

Abstract

As new threats frequently emerge, detecting cyber attacks necessitates more sophisticated methods. One technique for detecting novel attacks is anomaly detection, which leverages machine learning. However, using real-world data in anomaly detectors presents challenges, including privacy concerns and the need to train the system to detect rare events. This highlights the critical role of synthetic datasets. Nevertheless, the quality and extensibility of available synthetic datasets still need to be improved.

This study addresses these challenges by building a technique to generate data leveraging the DetGen framework. We have extended this framework to generate synthetic Sysmon logs. We demonstrate that the framework is adaptable to create additional log sources and maintains essential quality standards, as measured by its correctness, completeness, determinism, and utility. The results from this report are essential for developing anomaly detection and can also be used to validate attack detection rules in systems like Security Information and Event Management (SIEM).

Research Ethics Approval

Or include this statement:

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Andi Ari*)

Acknowledgements

I thank Professor David Aspinall, my supervisor, for the continuous guidance and encouragement throughout this research journey. I would also extend my appreciation to Robert Flood who gives insights in every discussion.

Lastly, I thank my family, whose love, support, and sacrifices have been my constant source of strength and motivation.

Table of Contents

1 Introduction **1**

2 Background **4**

2.1 Docker Container and Its Underlying Technology 4

2.1.1 Docker Overview 4

2.1.2 Namespaces: A Core Concept in Linux 5

2.2 Sysmon 7

2.2.1 Sysmon Overview 7

2.2.2 eBPF: The Heart of Sysmon for Linux 7

2.2.3 Available EventIDs 8

2.3 CommunityID 9

2.4 Zeek 11

2.5 Elastic 11

2.6 Data Generation Technique and Criticism 12

2.7 DetGen Framework 13

3 Design and Implementation **15**

3.1 Requirments 15

3.1.1 Requirement 1 - Correctness 15

3.1.2 Requirement 2 - Completeness 16

3.1.3 Requirement 3 - Deterministic 17

3.1.4 Requirement 4 - Utility 17

3.2 Design Considerations and Choise 18

3.2.1 System Architecture 18

3.2.2 Design Choice 19

3.3 System Configuration and Tooling 20

3.4 Program Implementation 21

3.5 Scenario Selection 23

4 Experiments **25**

4.1 Experiment 1: Accuracy of Generated Sysmon Data 25

4.1.1 Objective 25

4.1.2 Methodology 25

4.2 Experiment 2: Measuring the determinism 26

4.2.1 Objective 26

4.2.2 Methodology 26

4.3 Experiment 3: Assessing the utility of the data 27

4.3.1 Objective 27

4.3.2 Methodology 27

5 Evaluation and Results **28**

5.1 Evaluation of Experiment 1 28

5.1.1 Correctness 28

5.1.2 Completeness 28

5.2 Evaluation of Experiment 2 30

5.2.1 Nginx Scenario 31

5.2.2 Insecure SQL Scenario 32

5.3 Evaluation on Experiment 3 35

6 Conclusion **37**

6.1 Discussion on the Design and Experimentation 37

6.1.1 Discussion on Design 37

6.1.2 Disucssion on Experimentation 38

6.2 Future Work 39

6.3 Conclusion 40

Bibliography **41**

Chapter 1

Introduction

Software vulnerabilities enable adversaries to execute exploitation, leading to catastrophic security incidents, which recently rose in numbers as presented in some reports [29, 10, 26]. Denning [12] argues that software is often flawed, and fixing these flaws is non-trivial, given the cost and effort. One method to prevent such exploitation of vulnerable systems is to have the ability to detect an attack and stop it.

Over the past few decades, efforts have been made to enhance detection mechanisms. While traditional signature-based detection has proven effective at identifying known attacks [25], it struggles to detect more recent threats. To address this, researchers have started using machine learning methodologies to design anomaly detection systems, an approach that offers a way to detect new types of attacks [3, 22, 49]. However, a critical challenge arising with machine learning is data quality issues, a concern particularly relevant in the field of cybersecurity [32]. While real-world data could be utilised, numerous complications arise.

The first concern in the use of real-world data is privacy. Real-world data often contains sensitive information. Processing or sharing such data without adequate sanitisation might lead to the leak of private or sensitive information [50]. Secondly, there is an imbalanced dataset. This imbalance can skew the performance of detection algorithms toward high true negatives but poor true positive rates [17]. Another issue is temporal variability. Network traffic patterns can change over time, influenced by user behaviour, software updates, or infrastructure changes. Models trained on outdated data might fail to recognise new patterns [42]. Fourthly, there is a lack of ground truth issues. Accurate labelling of real-world data is challenging. Supervised learning techniques rely on labelled datasets and can only apply effectively with clear ground truth. Finally, a challenge in consistently evolving attack techniques. Cyber attackers continually

develop new methods and strategies. Thus, real-world datasets might not capture the full range of current threats, making it hard for detection models to stay up-to-date. Colbaugh [7] argues that traditional threat detection methods, which often rely on historical data, may need to be equipped to handle new, unseen attacks, highlighting the importance of capturing the full range of potential threats in datasets.

Consequently, researchers have suggested using synthetic datasets[47, 8, 32, 7]. Nevertheless, even these datasets come with their own set of hurdles. Firstly, achieving a realistic representation of network behaviour is difficult. A synthetic dataset must capture the intricacies and nuances of real-world network traffic, including benign activities, to be effective [39]. There are also requirements for novel attacks and their resemblance to real-world traffic[18]. Secondly, ensuring that a synthetic dataset covers a broad spectrum of attack types, both known and potential, is challenging. There is a risk of overemphasising specific attacks while neglecting others [42]. Thirdly, more standardised methods for generating data for anomaly detection would be needed to enable consistent and objective benchmarks [47]. Finally, Sharafaldin et al. [39] suggest that validating the effectiveness and realism of synthetic datasets is challenging. One might need real-world datasets to validate synthetic ones, but the availability of comprehensive real-world datasets is limited.

Concerning those problems, in previous research at the University of Edinburgh, Clausen [4] further underscored that currently, available synthetic datasets lack four crucial aspects: heterogeneity, ground truth labels, large data size, and up-to-date content. To address these issues, Clausen et al. [5] introduce the DetGet framework to build data that meet these conditions.

The existing DetGen Framework investigated in [4, 5, 6, 16] focused on examining network data that pertains to the traffic that traverses a network, captured in the form of packets. Network data encompasses source and destination IP addresses, port numbers, payload content, protocol types, packet size, and any other information [38]. Specifically, the mentioned studies harness the traffic microstructure information defined as ”reoccurring patterns in the metadata and temporal ordering of packet sequences in an individual connection, such as the packet sizes of a Diffie-Hellman exchange or typical IATs of video streaming...” [4]. IATs is traffic interarrval times. Clausen et al. [6] suggest that effectively capturing and representing these microstructures is fundamental to developing robust traffic anomaly models.

On the other hand, there are log data that are records produced by various components of an information technology system, including operating systems, applications,

and other devices. They chronicle events, transactions, and operational behaviour [21, 52]. Log data is invaluable for forensic investigations, allowing cybersecurity professionals to trace back events leading up to an incident. They also help in understanding the behaviour of systems and users over time, allowing for detecting anomalies that might signify a threat [21]. To this definition, Sysmon (System Monitoring) data falls under this category. It contains detailed process creation information, network connection monitoring, and file modification tracking[1]. We argue that having good quality Sysmon data contribute to the advancement in the anomaly detection domain. Accordingly, this dissertation extends the DetGen framework to generate synthetic Sysmon data.

This dissertation aims to explore and investigate the Sysmon data generation technique to expand from the previous network dataset investigation performed in [16]. This effort falls under the ongoing Detlearsom (Detection by Learning Software Models) project, whose overarching goal is to develop a system for detecting anomalies by understanding the structure of software.

Our main contributions are as follows:

- Building a framework to generate synthetic sysmon data established on specific scenarios utilising the DetGen framework. This demonstrates the extensibility of the DetGen framework to generate richer data types. Such an extensible framework is valuable for creating a better anomaly detector.
- We then use evaluative metrics to assess the quality of the generated data based on four fundamental criteria: correctness, completeness, determinism, and utility.
- Manifesting the utility of the generated data to enable event correlation with other log sources by incorporating Community ID.

Chapter 2 provides the background, detailing Docker technology used in DetGen, an overview of sysmon, and critiques of current datasets in the domain. Chapter 3 delves into the design and implementation processes, while Chapter 4 outlines the experimentation procedures employed. Chapter 5 presents a comprehensive evaluation, and the results obtained from the study, and Chapter 6 concludes the dissertation, summarising key findings and implications.

[1] https://github.com/OTRF/OSSEM-DD/tree/main/linux/sysmon/events

Chapter 2

Background

We aim to build a data generation technique for sysmon data, which is a valuable source of information for attack detection due to its granularity. This data generation method relies heavily on Docker and its underlying technology. We harness Docker containers' isolation capability to identify granular processes inside them. These processes represent simulated attacks or benign behaviour in the DetGen framework. A comprehensive understanding of such technology is crucial to building a technique that results in good quality synthetic sysmon data.

To set the relevancy of this study, we perform a literature review of the existing data generation techniques along with their criticism. Then, we show how this technique can contribute to the advancement of anomaly detectors.

2.1 Docker Container and Its Underlying Technology

2.1.1 Docker Overview

Docker is a platform that facilitates application development, shipment, and running inside containers. Containers allow a developer to package up an application with all parts it needs, such as libraries and other dependencies, and ship it all out as one package [14]. This ensures that the application will run on any other Linux machine regardless of any customised settings that the machine might have that could differ from the machine used for writing and testing the code.

At the heart of Docker's innovation is containerisation. Unlike traditional virtualisation, where each application requires a separate operating system instance to run, containerised applications share the same OS kernel and isolate the application

processes from each other [27]. This results in a significant performance boost and reduction in size. While VMs often take up several gigabytes, containers might only be tens of megabytes in size.

Docker uses the concept of images and containers. An image is a lightweight, stand-alone, executable software package that includes everything needed to run a piece of software [28]. Once an image is created, it can be used to instantiate containers that run the application. Docker Hub is a public registry containing a vast collection of images. Organisations also often use private registries to store and manage their images.

For complex applications with multiple interdependent containers, Docker Compose is a tool that allows developers to define and manage multi-container Docker applications [13].

2.1.2 Namespaces: A Core Concept in Linux

Namespaces, integral to the Linux kernel, offer a mechanism for resource partitioning. They enable different sets of processes to perceive disparate and isolated sets of system resources, thus ensuring one set of processes remains uninfluenced by another. This capability is foundational to containerisation, which allows applications to run in a self-contained environment, abstracted from the underlying infrastructure. Docker, a prominent containerisation tool, harnesses the power of namespaces to make its containerisation capabilities a reality[34].

Namespaces are instrumental in achieving the isolation and separation fundamental to containers. When Docker initiates a container, it leverages namespaces to provide an independent instance of system resources to that container. For instance, due to namespaces, a container can have its own dedicated filesystem, networking stack, process tree, and more [28]. This ensures that processes running within one container remain unaffected by processes in another, preserving the integrity and functionality of applications and services.

The Linux kernel supports various namespace types, each targeting a specific resource or set of resources[19]:

- **PID Namespace**: allocates a distinct set of PIDs to processes, ensuring no overlap with PIDs in other namespaces. When a process emerges in a new namespace, it is assigned PID 1, with subsequent child processes receiving the following PIDs. Notably, if a child process initiates its own PID namespace, it is labelled PID 1 within its namespace yet retains its original PID in the parent's namespace.

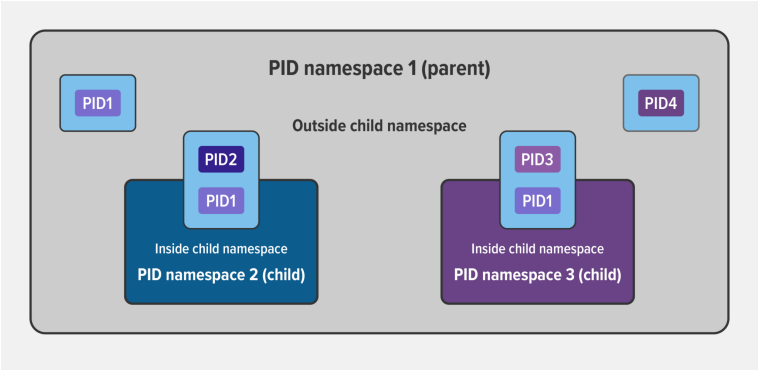


Figure 2.1: PID Namespaces Illustration, source: https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/

- **NET Namespace**: maintains a self-contained network stack. This encompasses its exclusive routing table, IP address pool, socket list, connection tracking table, and firewall, among other networking components.
- **MNT Namespace**: features a separate list of mount points, which processes within the namespace perceive. This flexibility allows for mounting and dismounting filesystems within the namespace without impacting the primary host filesystem.
- **UTS Namespace**: offers the intriguing capability where varied processes perceive the system with different host and domain names.
- **USER Namespace**: possesses a specific set of user and group IDs allocated to processes. Notably, within its respective user namespace, a process can attain root privileges without necessarily obtaining them in alternate user namespaces.

In Docker's implementation, these namespaces often operate in tandem, offering a layered and comprehensive isolation mechanism. The intricate use of multiple namespaces ensures that containers are self-contained and separated from the host and other containers.

The principles and methodologies discussed in this dissertation are deeply entrenched in the namespace paradigm. In generating synthetic sysmon data leveraging the DetGen framework, we utilise the isolative namespaces feature to identify processes running inside docker containers, where each process represents granular activities of benign or malicious scenarios. These processes are recorded in Sysmon data and can be identified by fields such as ProcessId, ProcessName, Image (a command that triggers

the process), User, network connection, and any other information. To this setting, namespaces maintain the container's namespace, including PID, NET, MNT, UTS, and USER Namespaces, so we can identify which container's processes are stored in the Sysmon data. This mechanism is the central notion of generating synthetic sysmon data that will be explained in detail in Chapter 3, Design and Implementation.

2.2 Sysmon

2.2.1 Sysmon Overview

Sysmon (System Monitor) originated as a Windows tool and has been a part of the Windows Sysinternals suite for a long time. Sysmon was designed to monitor and log system activity to the Windows event log, thus assisting in identifying security incidents and malicious activity [51].

On Windows systems, Sysmon quickly became an indispensable tool for security analysts and system administrators due to its comprehensive logging capabilities. Recognising the need for a similar tool in the expanding landscape of Linux deployments, especially with the increasing cloud adoption, Microsoft decided to bring Sysmon to Linux [40]. It is intended to bridge the monitoring gap and provide Linux users with the same in-depth insights into system activities as their Windows counterparts [40]. The move was a part of Microsoft's broader strategy to embrace and extend its tools and services to the open-source community.

Much like its Windows counterpart, Sysmon for Linux offers a variety of event IDs that correspond to different types of system activities. These include process creation, file and directory creation, network connections, and many others. Users can gain insights into potentially malicious or suspicious system activities by monitoring these event IDs.

2.2.2 eBPF: The Heart of Sysmon for Linux

At the core of Sysmon for Linux lies eBPF (Extended Berkeley Packet Filter). Shown in Figure 2.2, eBPF[1] is a technology that allows developers to run user-defined programs in the Linux kernel without altering the kernel source code or loading modules. Sysmon for Linux leverages eBPF to efficiently monitor system calls, network activities, and

[1] https://github.com/netoptimizer/prototype-kernel/blob/master/kernel/Documentation/bpf/index.rst

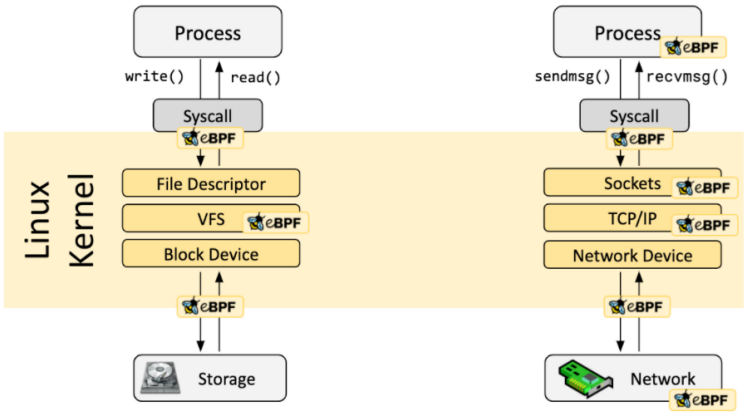


Figure 2.2: eBPF Architecture, source:https://techcommunity.microsoft.com/t5/microsoft-sentinel-blog/automating-the-deployment-of-sysmon-for-linux-and-azure-sentinel/ba-p/2847054

more, ensuring minimal performance overhead.

Originally conceived as a high-performance network packet filtering technology, eBPF has since evolved into a more general-purpose in-kernel virtual machine, making it versatile for various applications, including security monitoring, network tracing, and performance debugging.

Sysmon's integration with eBPF demonstrates the tool's adaptability. While Sysmon on Windows relied on native Windows APIs and event logging mechanisms, its transition to Linux required an understanding of the Linux kernel's inner workings. Here, eBPF came into play, allowing Sysmon to tap into the kernel and capture the necessary telemetry without being invasive[36].

2.2.3 Available EventIDs

Sysmon EventIDs categorise the system activities that Sysmon is designed to monitor. On Windows, the EventIDs span many activities, including process creation, network connections, driver loads, registry events, and many more[30]. Each EventIDs corresponds to a specific activity or event that Sysmon captures.

Regarding Sysmon for Linux, the set of available EventIDs is more limited than its Windows counterpart. This is not because of a limitation in Sysmon itself, but rather due to the intrinsic differences in the Linux and Windows operating system architectures [52]. For example, Windows-specific constructs like the registry have no direct counterpart in Linux; thus, registry-related EventIDs are absent[44]. Another reason is that sysmon for Linux is still in its relatively early stages of development

compared to its Windows counterpart. As a result, the tool may undergo iterative refinements and expansions, possibly introducing more EventIDs in future releases. The key EventIDs available in Sysmon for Linux[2] are presented in Table 2.1.

Sysmon for Linux is the central aspect of the investigation in this project. A comprehensive understanding of this technology sets the relevant scope and context to measure a good quality of synthetic sysmon data.

2.3 CommunityID

CommunityID[3]is a standardised hashing mechanism adopted primarily by Zeek, among other network analysis frameworks. Its primary objective is to generate consistent and unique identifiers for network sessions, irrespective of where or how the data is captured. This consistent identification approach becomes vital, especially in large-scale or distributed network environments, where traffic data might be sourced from multiple captures or data points. By employing CommunityID, analysts can seamlessly correlate related network sessions across diverse datasets, facilitating a holistic and integrated view of network interactions. This harmonisation not only simplifies the analysis process but also enhances the accuracy of network security investigations, ensuring that no pertinent session data is overlooked due to disparities in data representation. The Community ID is computed based on specific flow tuple details: the source and the destination IP addresses, the transport protocol, and the source and destination port. The outline of how the Community ID is computed is as follows:

- Normalisation: Regardless of which side is the client or server, the flow tuple elements are always ordered in a specific way to ensure consistency. This typically involves arranging the IP addresses and port numbers in ascending order.
- Concatenation: The normalised flow tuple details are concatenated into a single string. The precise separator used (if any) is implementation-specific, but the key is to ensure consistent ordering and representation.
- Hashing: The concatenated string is hashed, usually using a cryptographic hashing function like SHA-1.
- Base64 Encoding: The hash value is then base64 encoded to produce a compact and URL-safe representation.

[2]https://ossemproject.com/intro.html
[3]https://github.com/corelight/community-id-spec

EventID	Meaning	EventData Parameters
EventID 1	Process creation	RuleName, UtcTime, ProcessGuid, ProcessId, Image, FileVersion, Description, Product, Company, OriginalFileName, CommandLine, CurrentDirectory, IntegrityLevel, User, LogonGuid, LogonId, TerminalSessionId, ParentUser, ParentProcessGuid, ParentProcessId, ParentImage, ParentCommandLine
EventID 3	Network connection	RuleName, UtcTime, ProcessGuid, ProcessId, Image, User, Protocol, Initiated, SourceIsIpv6, SourceIP, SourceHostname, SourcePort, SourcePortName, DestinationIsIpv6, DestinationHostname, DestinationPort, DestinationPortName
EventID 4	Sysmon service state changed	UtcTime, State, Version, SchemaVersion
EventID 5	Process terminated	RuleName, UtcTime, ProcessGuid, ProcessId, Image, User
EventID 9	Raw access read	RuleName, UtcTime, ProcessGuid, ProcessId, Image, Device, User
EventID 11	File creation	RuleName, UtcTime, ProcessGuid, ProcessId, Image, TargetFilename, CreationUtcTime, User
EventID 16	Sysmon Config State Changed	UtcTime, Configuration, ConfigurationFileHash
EventID 23	File deletion	RuleName, UtcTime, ProcessGuid, ProcessId, Image, User, TargetFilename, Hashes, IsExecutable, Archived

Table 2.1: Available EventIDs in Sysmon for Linux. This table supports the explanation in Sub Section 2.2.3

- Final Format: The base64-encoded hash value is often prefixed with a version number or other metadata.

We utilise CommunityID to assess utility requirements explained in Chapter 3.

2.4 Zeek

Zeek, previously known as Bro, is an open-source network analysis tool that originated in the 1990s by Vern Paxson at Lawrence Berkeley National Laboratory [33]. Unlike traditional network intrusion detection systems, Zeek emphasises extensive logging, protocol parsing, and scriptable event-driven analysis. Its architecture can process vast traffic volumes, making it apt for high-performance networks. A distinguishing feature of Zeek is its adaptable scripting language, allowing custom analytics tailored to specific detection needs[43]. The active community behind Zeek continually enhances its capabilities, ensuring its relevance in today's dynamic network security landscape.

This project utilised Zeek to extract netflow from the pcap file generated by scenario execution.

2.5 Elastic

Elastic Platform[4], commonly associated with Elastic (or Elasticsearch), represents a suite of open-source tools designed to enable organisations to search, analyse, and visualise large datasets in real time. Rooted in its core component, Elasticsearch, a distributed search and analytics engine, the Elastic Platform also includes tools like Logstash, a server-side data processing pipeline, and Kibana, a data visualisation tool. Often leveraged in scenarios demanding log or event data analysis, Elastic Platform's scalability and speed have made it an industry favourite for operational intelligence, security analytics, and many other use cases where swift data retrieval and analysis are paramount.

We use this platform to assess utility requirements, which will be explained in Chapter 3. This platform will as SIEM and perform Community ID calculations.

[4]https://www.elastic.co

2.6 Data Generation Technique and Criticism

The origins of the static synthetic dataset for cyber security are focused on network data in several notable works, including DARPA, IDS[23], KDDCUP'99[20], NSL-KDD[46], UNSW-NB15[31], CCIDS2017[39], and NF-UQ-NIDS-v2[37]. Each of these offers a diverse array of traffic from multiple applications. Some researchers have pointed out concerns about these datasets. Among the issues raised are the absence of accurate comparisons to real-world traffic[24], potential data redundancy[45], limitations concerning period and the number of hosts used during dataset creation[5], and inaccuracies in simulating attacks[15].

Given these concerns, a novel approach is required. As highlighted by Shiravi et al. [41], there is merit in transitioning towards dynamic data generation strategies, allowing datasets to be more adaptively adjusted and recreated. Shiravi et al. [41] introduced ISCX-UNB by establishing guidelines for valid dataset generation, emphasising realistic traffic and diverse intrusion scenarios. However, it is not publicly accessible.

This aligns with the emergence of dynamic dataset generation for cyber security, which can be traced to several studies. Brauckhoff et.al [2] introduced FLAME, an attack injection tool for network flow. However, it is now discontinued [9]. INSecS-DCS [35], designed as a customisable software framework, offers on-demand dataset creation, providing raw and processed outputs. C. G. Cordero et al. [9] unveiled ID2T, addressing fundamental dataset issues and enhancing anomaly-based systems with features to evaluate traffic abnormality. Similarly, the AB-TRAP framework [11], distinguished for its reproducibility and attack currency, provides a cycle for NIDS solution design, from generating attack data to evaluating the deployed model's performance.

We identified that the data generation techniques mentioned above also have drawbacks that are well articulated by Clausen et al. [5] that suggest common problems in the modern synthetic dataset:

- Lack of diversity. The current training data mainly comes from isolated test environments, leading to a lack of diversity in protocols and resulting in uniform network flows. This simplicity in data structure makes it easier to differentiate between benign and malicious activities, but it can lead to overly optimistic results in machine learning.

- Lack of ground truth labelling. It is essential to understand the computational similarity between two connections and the nature of the traffic they represent.

However, obtaining this information is challenging, so current public NIDS do not utilise datasets containing these traffic labels.

- Static design. The current Network Intrusion Detection (NID) dataset, designed around specific vulnerabilities using fixed testbed machines, faces challenges in adapting to updated traffic structures due to the rigidity of the testbed setup.
- Limited size. Existing NIDS datasets are constrained by their test bed, usually capturing data from 5-10 hosts over 5-6 weeks. Ideally, researchers should be able to produce any amount of specific traffic types.

We realise that the dataset generation techniques discussed above are focussed on network data sets. Accordingly, harnessing the eminence of DetGen framework [4] in terms of its dynamic design and extensibility, this dissertation investigates a technique to generate synthetic sysmon data.

2.7 DetGen Framework

DetGen framework addresses challenges observed in synthetic datasets generation technique. Clausen et al. [4] highlighted various dataset prerequisites, including diversity (representing real-world protocols), ensuring repeatability, adding new attacks without altering existing data, and the capability to produce vast data volumes. The framework harnesses the isolative features of containers, differentiating various traffic types and their inherent scalability, given the ease of replicating containers. This approach offers a clear advantage compared to others, such as segregating traffic and leveraging the finer details and metadata of traffic to enhance the detection of malicious activity. DetGen operates within docker containers and has four capabilities:

- Scenarios. Specify the container interaction and are captured from each perspective to construct the base for malicious and benign traffic.
- Subscenarios. Created to specify more granular aspects of the traffic following the Scenarios. These sub-scenarios include variations of successful and unsuccessful file transfers, successful and failed login attempts and many more.
- Randomisation within sub-scenarios. Introduced within the sub-scenarios to generate a random distribution for the dataset that mimics real data. For example, this includes the distribution of file size and file names.

- Network transmission. Network parameters such as packet drop rate, bandwidth limit, and latency are modified to ensure they conform to the desired distribution.

We briefly explain one example of a DetGen scenario as follows. In SQL Injection Scenario, three docker containers are involved: MySQL container act as a database, Apache container as a web server, and Attacker container which simulate malicious python script to perform SQL injection attack on the web server. To capture the traffic on each container, associated tcpdump containers are applied where their function is to perform packet capture resulting in pcap files. The behaviour and configuration of each container are scripted in the docker-compose file. The Scenario is run through a script that executes docker-compose up command to start each container. The script is parameterised by duration and attack type that can be executed, i.e., attack0.py is a successful attack, attacking the login page, and attack1.py is an unsuccessful attack attacking the register page in the Apache container. The corresponding tcpdump container captures each container's network packet and stores them in each corresponding pcap file. This attack simulation occurs during the Duration supplied to the scenario execution script.

Chapter 3

Design and Implementation

3.1 Requirments

The main objective of this dissertation is to develop a technique to generate synthetic sysmon data. The technique leverages DetGen Framework. We defined the quality requirements that our framework should achieve to satisfy the requirement in the following Sub Sections.

3.1.1 Requirement 1 - Correctness

As suggested by Clausen et al. [5], ground truth labelling is a vital requirement for synthetic datasets. Applying labels on incorrect data will lead to incorrect results. Consequently, we will not get valid ground truth labeling. For that reason, our priority in this project is to create correct data so that labels can be accurately applied in future work. Our criteria for this requirement are defined as follows:

- We want the generated data to be correctly recorded according to the corresponding events resulting from scenario executions. For instance, if a container executes an HTTP connection, then the generated synthetic sysmon should also record that connection correctly for every EventID regarding its source IP address, source port, destination IP address, destination port, process id and any other relevant information.

- There should be no extraneous EventID generated from the intended scenario execution. For example, in the Nginx scenario, which simulates the HTTP connection from a client to a server, the typical identified EventID generated by

Sysmon is EventID 1, 3, 5, and 11. The log is correct if no extra EventID is included in the synthetic sysmon data.

- To get the right setting to prove that the EventID generated is correct, we do not add any activity noise to the container scenarios used in our experiment, which will be explained in Chapter 4.

3.1.2 Requirement 2 - Completeness

Sysmon data contains granular information that is valuable for anomaly detection [21]. Concerning this project, we want to harness the extensibility of the DetGen Framework by introducing a technique to generate Sysmon data built on top of this framework. Our technique will not give any advantage to the anomaly detection domain if the resulting data is incomplete, so the granularity is not achieved.

Our synthetic sysmon data is complete if it represents all the benign or malicious behaviour regarding scenario executions. We define completeness as every EventID corresponding to the processes in scenario executions should be recorded. The criteria for this requirement are as follows:

- Synthetic Sysmon should record all representative EventID according to the activities inside the docker container's scenarios. For instance, in the Insecure SQl scenario (which includes MySQL database container), the corresponding synthetic sysmon data should contain EventID 1, which represents process creation, EventID 3 for the network connection between MySQL and Apache, EventID 11 informing file creation, EventID 23 representing file deletion, and EventID 5 stating that process termination.

- Specific for EventID 3 network connection, we realise that they may have discrepancies compared to the pcap file in connection counts. For instance, in the Nginx scenario that utilises Siege tool to generate concurrent traffic, sysmon may not record all HTTP requests from client(s) on the server side. It will only record what connection is being processed by the Nginx server. In this project, the traffic processing capacity in the Nginx container is constrained by the value of *worker_connections* (representing how many requests the server will handle), *worker_processes* (representing the number of CPU will be utilised) and how long the server opens the connections is limited by the *keep_alive* timeout. In this case, we use 1024 **worker_connections**, auto detect *worker_processes*, and 65s

keep_alive timeout. This means that, with 4 CPUs, the approximate maximum number of simultaneous connections can be handled is 4x1024 connections, each of which takes 65s to keep open.

3.1.3 Requirement 3 - Deterministic

Clausen et al. [5] suggest that one of DetGen's objectives is to equip researchers with extensive ground truth information and produce controllable datasets. They suggest that to ensure accurate ground truth, it is vital to establish scenarios and sub scenarios that are stable and can be reliably reproduced when executed multiple times. Accordingly, this project also aims to inherit this determinism so that accurate ground truth labelling can be applied to future work.

In this project, we define our technique as deterministic if we have consistent correctness and completeness of synthetic Sysmon data across repeated scenario execution. Hence, we define the criteria as follows:

- Upon repeated execution of the scenarios, the generated data should not result in significant non-deterministic deviation. For instance, the network connection event represented by Event ID 3 should represent its corresponding network flow extracted from the PCAP file generated by the scenarios we pick.
- The correctness and completeness should also be consistent with other relevant EventID generated by picked scenarios across all iterations. For example, the Ngix scenario will generate EventID 1,3,5 and 11 for both the Nginx and Siege containers. This condition should be consistent across all iterations.

3.1.4 Requirement 4 - Utility

We aim to enrich DetGen Framework to generate sysmon data so it can also be used to correlate across different data sources to validate attacks. For example, attack information captured by NIDS can be enriched by the Sysmon data so that we can obtain more detailed information about what process triggered the connection, what type of malicious activity was realised in an attacked host, what file being accessed, and any other relevant information provided by Sysmon data. To validate the utility, we defined the criteria as follows:

- There should be matched CommunityID between network flow and synthetic sysmon data (represented by EventID 3 network connection). This match implies

that event correlation is valid since we found the corresponding traffic (benign or malicious) in network flow and sysmon data.

- The match amount depends on the processed connection recorded by Sysmon. The discrepancy is similar to what has been defined in Requirement 1. For example, in the server side of Nginx scenario, the number of CommunityID generated is constrained by the number of connections that the server can handle with respect to worker_processes, worker_connection, and keep_alive value.

3.2 Design Considerations and Choise

3.2.1 System Architecture

During the design phase, we considered three potential architectures to explore in implementing Sysmon to record events generated in docker scenarios (picked scenarios are explained in Sub Section 3.3). We explore the design alternatives as follows:

- **Design 1 - Host Sysmon View**. We deploy Sysmon for Linux [40] on the host machine. Consequently, the host Sysmon, captures every event from the host's perspective. Our objective is to generate synthetic sysmon according to the container's perspective as it represents actual containers' behaviour regarding their ProcessId, ParentProcessId, User, and any other relevant Sysmon information.

  We utilised the Namespaces feature discussed in Chapter 2 to achieve this objective. As explained in Chapter 2, every container has its own isolated PID, User, Network, Mount Point, and UTS. For instance, PID Namespace maintains the mapping between the host view and associated container PID. Upon running each picked Scenario, we generate synthetic sysmon by first querying the host Sysmon parameterised by host PID associated with their activity in each Scenario. We then modify the query result and store each record according to their container's view information.

- **Design 2 - Sysmon Container**. This method used Sysmon Container. This method may not require querying sysmon based on ProcessID like the one in Design 1. In Design 2, we expect Sysmon Container to record Sysmon data natively according to the container's perspective. In this design, we must modify each docker image to include sysmon in their docker requirement. However,

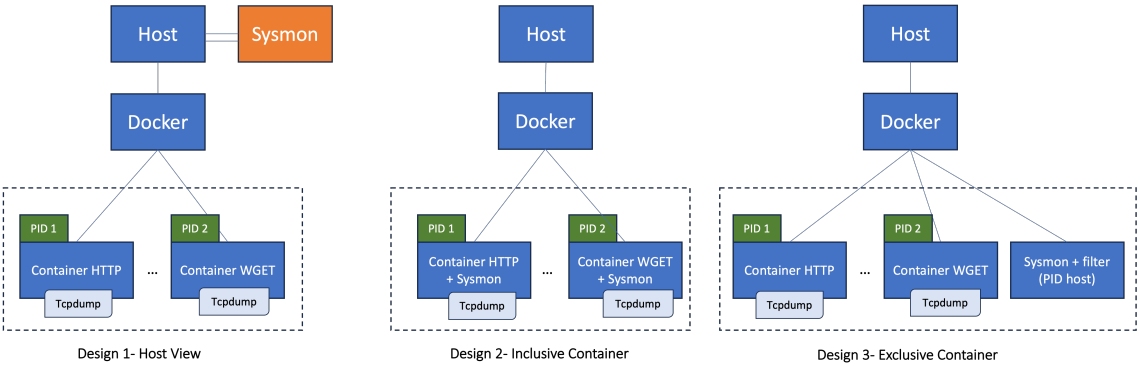


Figure 3.1: Design Options

we have found minimal references to deploying such containers. One source mentions that the events come back as null[1].

- **Design 3 - Exclusive Container**. This design differs from Design 2 in running an independent docker container so that we do not have to rebuild existing images. In this design, the query and modification mechanism utilising namespaces information mentioned in Design 1 is still required.

The design alternatives are shown in Figure 3.1

3.2.2 Design Choice

The design alternatives discussed above come with their benefits and drawbacks, which will be our justification factor and design decision.

The advantage of Design 1 is that we do not have to redesign and rebuild the container scenarios picked in Sub-Section 3.3. However, the downside of this design is the challenges in capturing a short-lived process. In Design 2, besides the limited sources to deploy the sysmon container, we have to redesign and rebuild each container scenario, resulting in debugging errors that may result from modifying docker images. This approach is quite risky considering the timeline to complete this project. Lastly, Design 3 inherit the benefit and drawback from Design 1. Additionally, it inherits Design 2's shortcomings in creating docker images, which will have potential errors, so we have to perform time-consuming debugging.

Considering the advantages and the downsides constrained by the time to investigate this project, we adopted Design 1. We explain the mechanism to tackle the drawback in Design 1 in Progam Implementation Section.

[1]https://github.com/Antonlovesdnb/LinuxVisibilityContainer

3.3 System Configuration and Tooling

We deployed the machine along with various tools to implement Design 1, elaborated as follows:

- OS Environment. To execute scenarios and generate sysmon data, we deploy a virtual machine on Google Cloud Platform (GCP) with the specification of e2-standard-4 (4 vCPUSs x86/64 architecture, 16GB Memory, and 64GB Balanced persistent disk) and Ubuntu 20.04.1 operating system.
- Sysmon for Linux. We deploy Sysmon for Linux based on the documentation provided in [40]. We configure the Sysmon to capture all the EventID[2]
- XML processor. Another tool that is required for this project is xmltartlet[3] which is used to transform Sysmon into a pure XML file so that we can normalise and sanitise the sysmon data properly.
- SIEM. We also use the open-source version of the Elastic platform[4] to act as a SIEM system. This tool is practical for simulating event correlation used in the experiment elaborated in Chapter 4. This tool is deployed in our personal computer, separated from where the synthetic sysmon data is generated.
- Data pre-processing. Additionally, for the data pre-processing phase, we use Python data science, which is very useful for transforming data and calculating statistical aspects.
- Calculating tree distance. We use ZSS python library[5] to calculate Tree Edit Distance to measure similarity between process trees across iterations in each scenario execution.
- Netflow extraction. To extract netflow data from the pcap file generated by scenario executions, we use Zeek[6] (previously Bro IDS). This NetFlow is fed to the SIEM tool for the next processing.

We deploy SIEM, Python Data Science, ZSS Library, and Zeek on our personal computer, separated from the machine where scenario executions occur.

[2]https://github.com/microsoft/MSTIC-Sysmon/blob/main/linux/configs/collect-all.xml
[3]https://xmlstar.sourceforge.net
[4]https://www.elastic.co
[5]https://pypi.org/project/zss/1.1.4/
[6]https://zeek.org

Algorithm 1. Executing Scenario

```
1.  For i From 1 to Repeat
2.      Call BringUp function
3.      Call MonitorProcessContainer function in the background
4.      Pause for Duration in Seconds
5.      Wait for all background processes to finish
6.      Call GenerateSysmon function
7.      Call Teardown function
8.  End For
```

Figure 3.2: Pseudo code for the main program to execute Scenario

3.4 Program Implementation

The program to generate synthetic Sysmon data follows the Design 1 principle. We modify the main script to execute the Scenario to accommodate sysmon data generation (Pseudocode shown in Figure 3.2). It is implemented in the following sequence:

1. **Step 1: Process identification**. The main idea of this step is to identify ProcessId representing processes running inside each executed container scenario. Subsequently, we create reference tuples for each container scenario that contain information including ProcessId, container ProcessId, ParentProcessId, container ParentProcessId, name of corresponding Scenario (for example, in Nginx scenario, this can be Server or Client), Image (process name), User from host perspective, and User from container perspective. ProcessId(s) are the keys in the created reference tuples used as a reference to perform Query in the next step.

   To get the host's ProcessId and ParentProcessId, we run the *docker top* command for each container. Subsequently, we find the corresponding container's ProcessId and ParentProcessId by leveraging the PID Namespace feature where it maintains its information inside */proc/pid/status*. Finally, to get the rest of the information required in the reference tuples, we run *ps* command parameterised identified ProcessId. The pseudocode for Step 1 is shown in Figure 3.3.

2. **Step 2: Sysmon Query**. In this step, we perform Query on the original Sysmon stored in */var/log/syslog* based on the identified ProcessId in Step 1 utilising the reference tuples. This queried Sysmon representing the log for each executed Scenario. Subsequently, we transform the original sysmon into pure XML format

Algorithm 2. Process Identification

```
1.  Function MonitorProcessContainer()
2.    Calculate end_time as current system time in seconds +
      monitor_duration
3.    While current system time in seconds < end_time:
4.      Get the list of container PIDs and store in container_pids
5.      For each pid in container_pids:
6.        Get process_name, parent_pid, container_id,
          parent_container_id, user_name, container_uname
7.        Append as unique tuple to the file named
          container_name
8.      End For
9.    Sleep for 1 second
10.   End While
```

Figure 3.3: Pseudo code for Step 1: Process Identification

Algorithm 3. Querying Sysmon

```
1.  Function sysmonQuery()
2.    Initialize Input Stream (IFS) to read each line from a file named container_name
      containing tuples
3.    For Each line in the file:
4.      Extract process_id
5.      Search the /var/log/syslog for lines containing the "ProcessId" value that
        matched the extracted process_id
        Append the matched lines to a new Sysmon file
6.    End For
7.    Call SanitisedData Function
```

Figure 3.4: Pseudo code for Step 2: Sysmon Query

so we can properly perform Normalisation and Sanitation in Steps 3 and 4, respectively. Pseudocode for this step is shown in Figure 3.4

3. **Step 3: Normalisation**. This step modifies ProcessId and User field inside the Sysmon data, resulting in Step 2 by utilising the reference tuples. We perform this operation using the xmlscarlet tool mentioned in Section 3.3. Pseudocode shown in Figure 3.5.

4. **Step 4: Sanitation**. The main objective of this step is to shape the synthetic sysmon to contain information based on the container's perspective. It employs stripping out fields: ParentProcessGuid(s), ParentImage(s), ParentCommand-Line(s), and ParentUser(s) for ProcessId(s) that has parent process from the host

Algorithm 4. Sanitising the queried sysmon

```
1.  Function SanitisedData()
2.    Transform queried Sysmon into pure xml format
3.    For Each line in the file named container_name:
4.        Extract process_id, container_process_id, parent_process_id,
          parent_container_process_id, user_name, and container_username
5.        Update Sysmon xml file with that corresponding extracted  field
6.        Update Sysmon xml file, removing and parent process of init process in every
          container
7.        Update Sysmon xml for, removing any pattern of overlay file system
8.  End For
```

Figure 3.5: Pseudo code for Step 3 and 4: Normalisation and Sanitation

perspective but has no corresponding ProcessId(s) according to the Namespaces information maintained in */proc/pid/status* in the host. In this step, we remove every field containing the directory path that still refers to the overlay file system used by containers. The Sanitation step ensures correctness, meaning that we do not generate ”noisy” information that is still associated with the host’s processes. Pseudocode shown in Figure 3.5

3.5 Scenario Selection

We depict 13 scenarios from the private DetGen Github repository in Table 3.1. The chosen scenarios ranged from simple to complex representing benign and malicious traffic. Before determining scenarios for our experiment, we conduct a run-through test to identify which container can run smoothly on our machine. Several scenarios can not be executed for several reasons. For instance, The MPD, Nginx Patator, Slowhttptest, ssh patator, and Stream can not be executed mainly because the images are no longer available in the Github repository[7].

[7]https://github.com/detlearsom/detgen-working/tree/master

Table 3.1: Implemented Scenarios

#	Scenario	Description	Attack Type
1	Nginx Nginx Wget Nginx SSL Apache Apache Wget Apache SSL	Generating http/s traffic to corresponding server (Nginx and Apache)	Benign
2	Wordpress	Generate recursive wget to wordpress server	Benign
3	Syncthing	File synchronization	Benign
4	Openssh	Establishing SSH connection	Benign
5	Nginx Bruteforce	Generating bruteforce traffic	Attack
6	Heartbleed	Simulating heartbleed attack	Attack
7	Secure SQL	Simulating unsuccessful SQL injection attack	Attack
8	Insecure SQL	Simulating successful SQL injection attack	Attack
9	Backdoor	Simulating backdoor communicating with CnC	Attack
10	Goldeneye	DDoS simulation on Nginx server	Attack
11	Mirai	Simulating Mirai botnet	Attack
12	Wget SSL	Running wget variation of user agents	Benign
13	NTP	Generating NTP traffic between client and server	Benign

Chapter 4

Experiments

Our objective is to conduct experiments to evaluate our technique and the resulting data against the requirements mentioned in Chapter 3: correctness, completeness, determinism, and utility. To achieve this, we design three experiments.

4.1 Experiment 1: Accuracy of Generated Sysmon Data

4.1.1 Objective

This experiment evaluates the degree of completeness and correctness of the generated data across the implemented scenarios. Given the program implementation explained in Chapter 3, we want that every field in each data structure between synthetic and the actual sysmon data are matched. We must also ensure that all the relevant EventId(s) are available in the generated data.

4.1.2 Methodology

To evaluate the correctness, we conduct field-by-field comparisons manually to ensure the Query, Normalisation, and Sanitisation are correctly executed. To do this, we utilised the reference tuples mentioned in Section 3.4. The generated synthetic Sysmon data should be a correctly normalised and sanitised version of the corresponding original Sysmon data.

Additionally, to measure the completeness, we check every EventID representing the activity of the scenarios that can be analysed via the docker-compose file and the process behaviour using the Linux command to catch the PIDs. As a test case, we

sample a simple scenario like one of the HTTP scenarios and a complex scenario like secure and insecure SQL.

4.2 Experiment 2: Measuring the determinism

4.2.1 Objective

Our objective in this experiment is to measure the degree of determinism, meaning that for every repeated scenario execution, the resulting data remains similar without any meaningful deviation. As this framework leverages docker containers with strong isolation characteristics, we argue that our generated data will achieve this determinism.

4.2.2 Methodology

We measure the determinism on two criteria. Firstly, we specifically evaluate the consistency of generated Event ID 3 representing network connection for selected scenarios. The metrics used are the match percentage of connections compared to network flow generated from pcap data for each Scenario. We then calculate the Mean, Max, and Min of the connections count, and finally, we calculate the Coeffisien Variance [1] to assess the relative volatility or spread across the iterations.

Secondly, we also measure the similarity of process trees for more complex scenarios that generate more activity, including process creation, file deletion, and process termination. In this method, we employ the Tree Edit Distance (TED) metrics to measure the similarity of process trees across iterations. TED is a measure used to determine the similarity between two trees, specifically by quantifying the minimum cost of transforming one tree into another using a set of elementary operations. These operations consist of node deletions, node insertions, and node relabeling. The cost is then the cumulative cost of these operations. TED provides a way to understand how structurally similar or dissimilar two trees are by capturing the effort needed to morph one into the other. The concept is analogous to the string edit distance (or Levenshtein distance), which measures the similarity between two sequences or strings [48]).

In both methods, if there is no significant deviation across iteration over the metrics used, the framework satisfies the deterministic criteria.

4.3 Experiment 3: Assessing the utility of the data

4.3.1 Objective

The generated synthetic Sysmon data should be useful for detecting attacks. This experiment aims to generate sysmon data along with corresponding Community ID[1]. The Community ID is a standardised flow hashing mechanism designed to label network traffic flows with a consistent ID. This is useful for various network analysis and security tasks, such as correlating logs from different sources. By doing this, we want our generated data to be able to correlate with corresponding network flow data in each Scenario.

4.3.2 Methodology

We will implement the Community ID on the pipeline of our implemented elastic search during ingestion before the data is indexed[2]. We use three steps of data pipeline: data collection, parsing, and analysis. A brief explanation of the environment setup is as follows:

- Data preparation. We use Zeek (previously Bro) to extract flow from the pcap for each Scenario. Then, the XML sysmon data generated in DetGen are transformed to CSV.
- Data Collection. This is part of data processing step in the elastic where we use filebeat to collect data from the directory where the prepared files reside; then we send it to the elasticsearch[3].
- Creating pipeline in elastic search. In this step, we create a pipeline script. Its primary purpose is to catch the logs from filebeat with the correct format and calculate CommunityID for each connection for sysmon and network flow data right before the ingested data is indexed.
- Calculating Match. This is the step where we find the match of Community ID in Netflow and sysmon. We calculate the match by creating a visualisation dashboard rule to query the data from both log types with the same community ID.

[1] https://github.com/corelight/community-id-spec
[2] https://www.elastic.co/guide/en/elasticsearch/reference/current/ingest.html
[3] https://www.elastic.co/guide/en/elasticsearch/reference/current/ingest.html

Chapter 5

Evaluation and Results

5.1 Evaluation of Experiment 1

5.1.1 Correctness

The event comparison mapping between the synthetic sysmon and the original sysmon stored in */var/log/syslog* indicates that the generation technique yields a correct result. It implies that our Query, normalisation, and sanitation process work properly. We sample this event comparison on Event IDs 1 and 3, where most normalisation and sanitation occur.

We use the Nginx scenario to compare each Event ID. In Event ID 1, the *ProcessId* normalisation runs correctly. It transforms from *ProcessId: 2086676* in the original sysmon to *ProcessId=1* in the synthetic one. In the original sysmon log, this *ProcessId* correspond to *ParentProcessGuid: e9d7acd6-787b-64d9-b640-860000000000*, *ParentProcessId: 2086655*, *ParentImage: /usr/bin/containerd-shim-runc-v2*, *ParentCommandLine: /usr/bin/containerd-shim-runc-v2*, *ParentUser: root* which are the host process.

For EventID 3, the normalisation is applied for fields *ProcessId* and **User**. In the original file, field **User** contains value *systemd-network* while the sanitised version contains value *nginx*. The *ProcessId* is also normalised to value 8 from 2086720. The event comparison is shown in Figures 5.1 and 5.2.

5.1.2 Completeness

We observed that the generated EventIDs are complete. It generates relevant EventIDs, including EventID 1 process creation, EventID 3 network connections, Event ID 5 file

Table 5.1: Evaluation of Event Completeness on each scenarios (Experiment 1)

Scenario	Description	Attack Type	Generated Event ID
Nginx Nginx Wget Apache Apache Wget	Generating http/s traffic to corresponding server (Nginx and Apache)	Benign	server: 1, 3,5,11 client: 1, 3, 5, 11
Nginx SSL Apache SSL	Generating http/s traffic to corresponding server (Nginx and Apache)	Benign	Server:1,3,5,11; Client:1,5,11
Wordpress	Generate recursive wget to wordpress server	Benign	database: 1,3,5,11,23; sServer: 1,3,5,11,23; Wget: 1,5,11
Syncthing	File synchronization	Benign	Syncthing1:1,3,5,11,23 Syncthing2:1,5,11 Syncthing2:1,5,11
Openssh	Establishing SSH connection	Benign	Server:1,3,5,11; Client:1,5,11
Nginx Bruteforce	Generating bruteforce traffic	Attack	Server:1,3,5,11; Client:1,3, 5,11
Hearbleed	Simulating heartbleed attack	Attack	Server:1,5,11; Client:1,5,11
Secure SQL	Simulating unsuccessful SQL injection attack	Attack	Server:1,3,5,11 Attacker: 1,3,5,11 attacker:1,3,5,11
Insecure SQL	Simulating successful SQL injection attack	Attack	Attacker: 1,3,5 MySQL: 1,3,5,11,23 apache:1,3,5,11
Backdoor	Simulating backdoor communicating with CnC	Attack	Server:1,3,5,11; CNC:1,3,11, 23
Goldeneye	DDoS simulation on Nginx server	Attack	server: 1,3,5,11 Server:1,3,5,11,23 Attacker:1,3,5,11
Mirai	Simulating Mirai botnet	Attack	CNC:1,3,5,11 bot(1,2,3):1,3,5,11 Apache and Attacker: 1,5,11
Wget SSL	Running wget variation of user agents	Benign	Server:1,3,5,11; Wget1:1,3,5,11; Wget2:1,5,11; Wget3:1,3,5,11; Wget4:1,3 5,11, 23
NTP	Generating NTP traffic between client and server	Benign	Server:1,5,11; Client:1,5,11

Synthetic Sysmon	*Original Sysmon from /var/log/syslog*
<EventData>	Event SYSMONEVENT_CREATE_PROCESS
<Data Name="RuleName">-</Data>	RuleName: -
<Data Name="UtcTime">2023-08-14 00:42:35.985</Data>	UtcTime: 2023-08-14 00:42:35.985
<Data Name="ProcessGuid">{e9d7acd6-787b-64d9-74a7-f0e63d560000}</Data>	ProcessGuid: {e9d7acd6-787b-64d9-74a7-f0e63d560000}
<Data Name="ProcessId">1</Data>	ProcessId: 2086676
<Data Name="Image">/usr/sbin/nginx</Data>	Image: /usr/sbin/nginx
<Data Name="FileVersion">-</Data>	FileVersion: -
<Data Name="Description">-</Data>	Description: -
<Data Name="Product">-</Data>	Product: -
<Data Name="Company">-</Data>	Company: -
<Data Name="OriginalFileName">-</Data>	OriginalFileName: -
<Data Name="CommandLine">nginx -g daemon off;</Data>	CommandLine: nginx -g daemon off;
<Data Name="CurrentDirectory">/</Data>	CurrentDirectory: /
<Data Name="User">root</Data>	User: root
<Data Name="LogonGuid">{e9d7acd6-0000-0000-0000-000000000000}</Data>	LogonGuid: {e9d7acd6-0000-0000-0000-000000000000}
<Data Name="LogonId">0</Data>	LogonId: 0
<Data Name="TerminalSessionId">4294967295</Data>	TerminalSessionId: 4294967295
<Data Name="IntegrityLevel">no level</Data>	IntegrityLevel: no level
<Data Name="Hashes">-</Data>	Hashes: -
<Data Name="ParentProcessGuid">-</Data>	ParentProcessGuid: {e9d7acd6-787b-64d9-b640-860000000000}
<Data Name="ParentProcessId">-</Data>	ParentProcessId: 2086655
<Data Name="ParentImage">-</Data>	ParentImage: /usr/bin/containerd-shim-runc-v2
<Data Name="ParentCommandLine">-</Data>	ParentCommandLine: /usr/bin/containerd-shim-runc-v2
<Data Name="ParentUser">-</Data>	ParentUser: root
</EventData>	
</Event>	

Figure 5.1: Event ID 1 comparison. The highlighted rows show the event comparison. The left column is the synthetic Sysmon data that has been normalised and sanitised, while the right column is the original Sysmon data.

Synthetic Sysmon	*Original Sysmon from /var/log/syslog*
<EventData>	Event SYSMONEVENT_NETWORK_CONNECT
<Data Name="RuleName">-</Data>	RuleName: -
<Data Name="UtcTime">2023-08-14 00:42:37.658</Data>	UtcTime: 2023-08-14 00:42:37.658
<Data Name="ProcessGuid">{e9d7acd6-787c-64d9-74a7-f0e63d560000}</Data>	ProcessGuid: {e9d7acd6-787c-64d9-74a7-f0e63d560000}
<Data Name="ProcessId">8</Data>	ProcessId: 2086720
<Data Name="Image">/usr/sbin/nginx</Data>	Image: /usr/sbin/nginx
<Data Name="User">nginx</Data>	User: systemd-network
<Data Name="Protocol">tcp</Data>	Protocol: tcp
<Data Name="Initiated">false</Data>	Initiated: false
<Data Name="SourceIsIpv6">false</Data>	SourceIsIpv6: false
<Data Name="SourceIp">172.29.0.3</Data>	SourceIp: 172.29.0.3
<Data Name="SourceHostname">-</Data>	SourceHostname: -
<Data Name="SourcePort">36412</Data>	SourcePort: 36412
<Data Name="SourcePortName">-</Data>	SourcePortName: -
<Data Name="DestinationIsIpv6">false</Data>	DestinationIsIpv6: false
<Data Name="DestinationIp">172.29.0.2</Data>	DestinationIp: 172.29.0.2
<Data Name="DestinationHostname">-</Data>	DestinationHostname: -
<Data Name="DestinationPort">80</Data>	DestinationPort: 80
<Data Name="DestinationPortName">-</Data>	DestinationPortName: -
</EventData>	

Figure 5.2: Event ID 3 comparison. The highlighted rows show the event comparison. The left column is the synthetic Sysmon data that has been normalised and sanitised, while the right column is the original Sysmon data.

creation, and Event ID 23 file deletion as presented in Table 5.1. On the other hand, Event ID 4 sysmon started or stopped, EventID 9 raw data access, and EventID 16 sysmon configuration updated do not appear because the corresponding scenarios do not generate such processes. As we defined in Requirement 1 in Chapter 3, no noise process is added to each Scenario.

5.2 Evaluation of Experiment 2

We sample the Nginx and Insecure SQL representing the simple and the complex Scenario, respectively.

5.2.1 Nginx Scenario

For the Nginx scenario, we used 100 threads, 100 requests per thread within 30 seconds, while the repetition was realised at 5, 10, 30, and 100. With this setting, the Siege container should make 100(threads)x100(request per thread)=10,000 simultaneous connections.

We observed that tcpdump does not record all the TCP conversations (6,020 out of 10,000 generated HTTP connections). Since we use only 100 request in this experiment, this finding is aligned with what explained in the DetGen GitHub Repository[1] that Siege http request happening so fast so there may not enough time for tcpdump to capture the traffic. We can see this phenomenon in Figure 5.3. In contrast, at the server side / Nginx container, we observed that the associated tcpdump container records 100% (10,000 out of 10,000 generated HTTP connection) of TCP conversation. We can see the TCP conversation count in Figure 5.4.

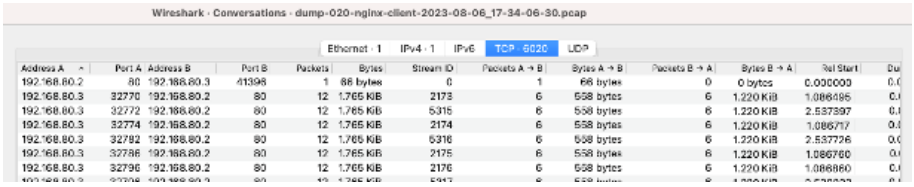


Figure 5.3: TCP conversation in captured by tcpdump for Siege container. The TCP conversation count is 6,020

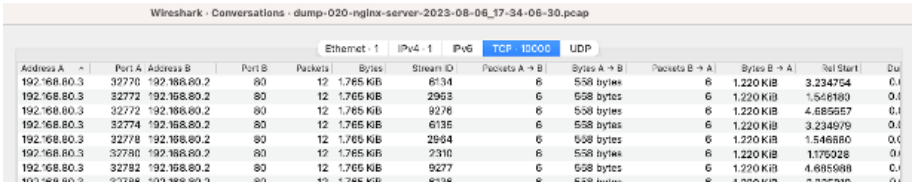


Figure 5.4: TCP conversation in captured by tcpdump for Nginx container. The TCP conversation count is 10,000

We then compare the match connection between network flow (extracted from the pcap file discussed above) and the associated network connection recorded in synthetic sysmon represented by EventID 3. On the client side / Siege container, all network flow connections are included in Sysmon connection records. This is represented by flow_match_average on the left-hand side of Figure 5.5. Furthermore, there are Sysmon

[1] https://github.com/detlearsom/detgen-working/blob/master/captures/capture-020-nginx/README.md

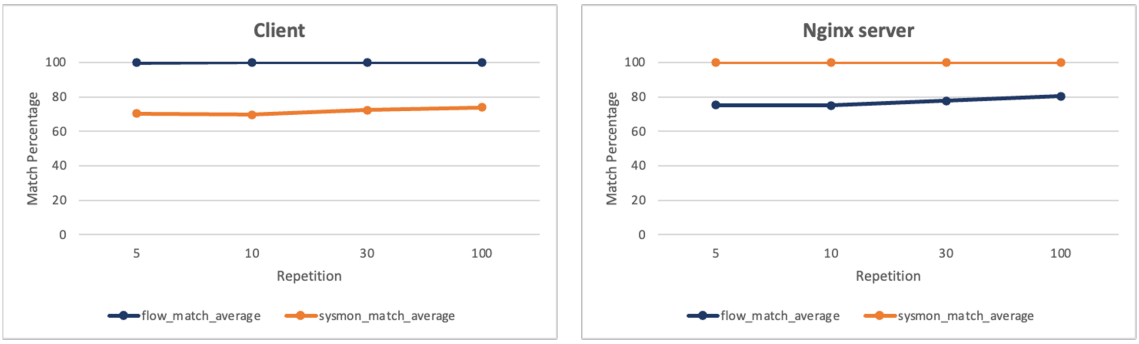


Figure 5.5: Match percentage comparing network flow and sysmon network connection for Nginx scenario both for client and server side

connections that has no corresponding traffic in network flow. This is represented by sysmon_match_average on the left-hand side in Figure 5.5. On the server side / Nginx container, we observe that not all network flow connection is recorded in synthetic sysmon, represented in flow_match_average on the right-hand side of Figure 5.5. In contrast, all connections recorded in synthetic sysmon match with network flow connection. It is represented by sysmon_match_average on the right-hand side of figure 5.5.

The pattern in Figure 5.5 matches what is shown in Figure 5.3. and 5.4 where at the client side, tcpdump does not record all the connections due to the condition explained above. Then on the server side, tcpdump records all the TCP conversations.

Mean, Max, Min, and Coefficient Variant are calculated here. The results is presented in Figure 5.6. We can observe that the Mean connection count is consistent across iterations at approximately 10,000 at client side / Siege container. The result implies that our technique generates consistent logs across iterations. On the server side, synthetic sysmon does not fully record the connection. This is the same pattern as the connection comparison explained above. This phenomenon, will be discussed in Chapter 6. However, the Mean of connection counts across iterations is also consistent with 2% Coefficient Variant. This also implies that our technique is consistent across repeatable execution.

5.2.2 Insecure SQL Scenario

This Scenario simulates SQL injection attacks on the server side through the execution of a malicious script on the client side, along with taking injection parameters from a word list.

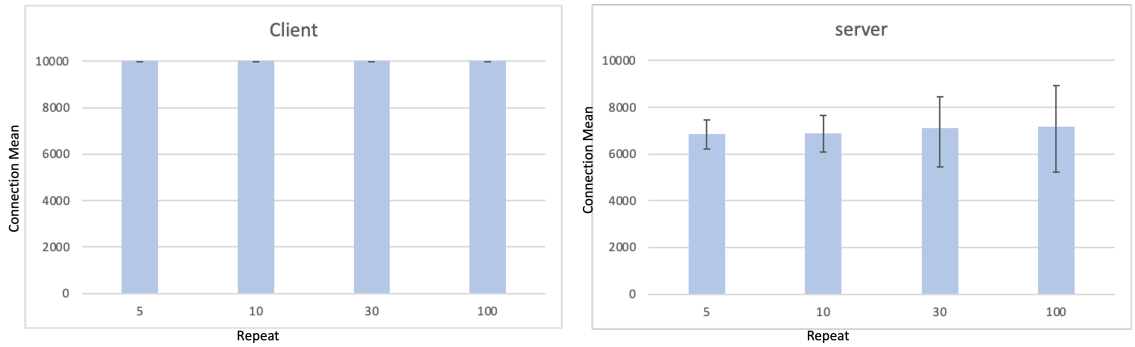


Figure 5.6: The variation of the number of network count in the sysmon (represented in EventID 3 both for client and server-side)

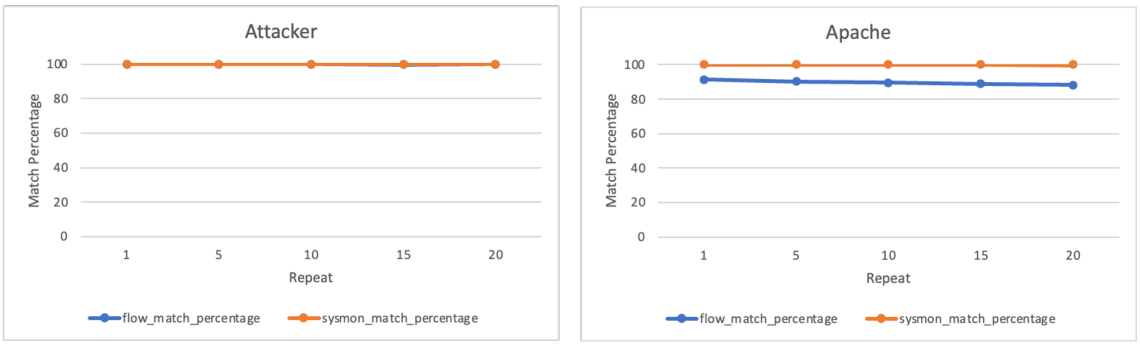


Figure 5.7: The variation of the number of network count for Insecure SQL scenario in the sysmon (represented in EventID 3 both for client and server-side)

We observed the identical result pattern with the Nginx scenario. Using 60 seconds of execution time across 1, 5, 10, 15 and 20 repetitions, on the attacker side, the percentage of match between flow and sysmon and vice versa is 100% (left-hand side of Figure 5.7). The same phenomenon is found in the server side of Nginx (right-hand side of Figure 5.8). We also see no significant difference in the number of connections across the entire repetition in the experiments with 6.48% Coefficient Variant.

To measure the repeatability, we also consider the process tree for InsecureSQL as a representation of complex scenarios. We presented the sample of the process tree from the Sysmon generated by the Insecure SQL scenario for 20 repetitions on the MySQL side. The process tree represents all relevant MySQL processes.

- Process ID: 1 (Image Name: /usr/sbin/mysqld): This is the root process with Process ID (PID) 1. It is running the MySQL daemon, which is the main service for the MySQL database system. All the processes listed under it (indented with the |– prefix) are its child processes.

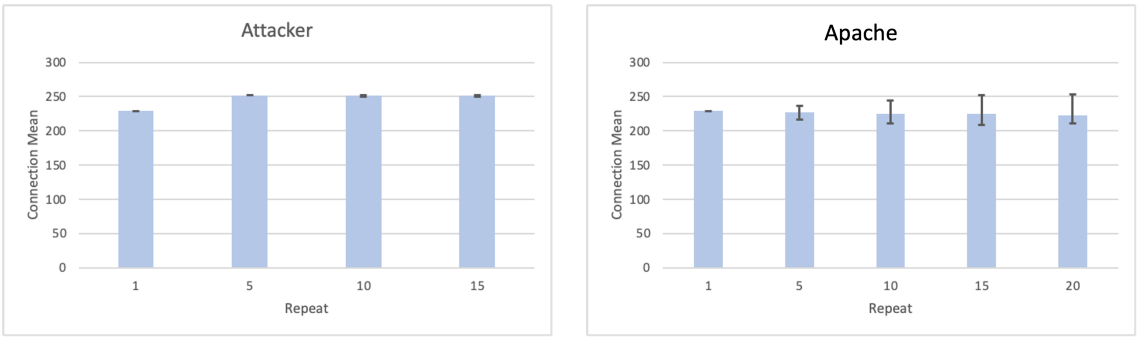
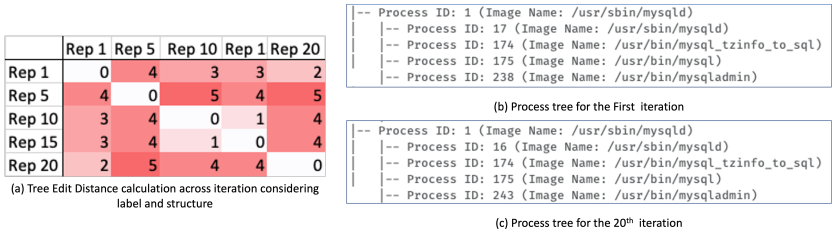


Figure 5.8: The variation of the number of network count in the sysmon (represented in EventID 3 both for client and server-side)

- Process ID: 16 (Image Name: /usr/sbin/mysqld): This is a child process of PID 1. It is also running the MySQL daemon, which suggests that the main MySQL daemon (PID 1) has started another instance or thread of itself.
- Process ID: 174 (Image Name: /usr/bin/mysql_tzinfo _to _sql): This is another child process of PID 1. It's running a utility related to MySQL, specifically one that deals with time zone information.
- Process ID: 175 (Image Name: /usr/bin/mysql): This is another child process of PID 1. This process is running a MySQL client, which is a command-line tool to interact with the MySQL database.
- Process ID: 243 (Image Name: /usr/bin/mysqladmin): Another child process of PID 1. This one is running the mysqladmin utility, a command-line tool that provides administrative operations for the MySQL server.

Then, we calculate TED for both structural and label and structural only. As we predict, the TED scores vary for the first one as the ProcessID that represents the label is always newly generated by the container across repetition (Figure 5.9). On the other hand, the result of the structural-only TED calculation is also as we expect (Figure 5.10). It consistently scores zero, meaning the process tree generated is identical across repetition. Except for iteration 5, we identify that this Image /usr/sbin/mysqld is generated twice inside the tree. Regarding this finding, we observed that ProcessId 106, as shown in Figure 5.10 c, runs command *mysqld –user=mysql –daemonise –skip-networking –socket=/var/lib/mysql/mysql.sock –default-time-zone=+00:00* which initialised MySQL daemon. When the MySQL Docker container is first initialised (when it detects an empty data directory), it undergoes an initialisation process. This process might involve



	Rep 1	Rep 5	Rep 10	Rep 1	Rep 20
Rep 1	0	4	3	3	2
Rep 5	4	0	5	4	5
Rep 10	3	4	0	1	4
Rep 15	3	4	1	0	4
Rep 20	2	5	4	4	0

(a) Tree Edit Distance calculation across iteration considering label and structure

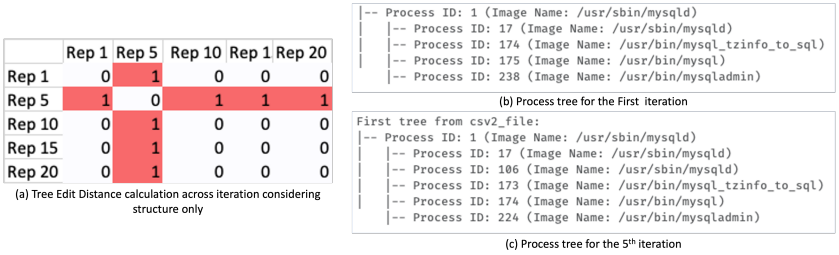
```
|-- Process ID: 1 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 17 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 174 (Image Name: /usr/bin/mysql_tzinfo_to_sql)
|   |-- Process ID: 175 (Image Name: /usr/bin/mysql)
    |-- Process ID: 238 (Image Name: /usr/bin/mysqladmin)
```

(b) Process tree for the First iteration

```
|-- Process ID: 1 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 16 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 174 (Image Name: /usr/bin/mysql_tzinfo_to_sql)
|   |-- Process ID: 175 (Image Name: /usr/bin/mysql)
    |-- Process ID: 243 (Image Name: /usr/bin/mysqladmin)
```

(c) Process tree for the 20th iteration

Figure 5.9: TED calculation (label and structure)



	Rep 1	Rep 5	Rep 10	Rep 1	Rep 20
Rep 1	0	1	0	0	0
Rep 5	1	0	1	1	1
Rep 10	0	1	0	0	0
Rep 15	0	1	0	0	0
Rep 20	0	1	0	0	0

(a) Tree Edit Distance calculation across iteration considering structure only

```
|-- Process ID: 1 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 17 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 174 (Image Name: /usr/bin/mysql_tzinfo_to_sql)
|   |-- Process ID: 175 (Image Name: /usr/bin/mysql)
    |-- Process ID: 238 (Image Name: /usr/bin/mysqladmin)
```

(b) Process tree for the First iteration

```
First tree from csv2_file:
|-- Process ID: 1 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 17 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 106 (Image Name: /usr/sbin/mysqld)
|   |-- Process ID: 173 (Image Name: /usr/bin/mysql_tzinfo_to_sql)
|   |-- Process ID: 174 (Image Name: /usr/bin/mysql)
    |-- Process ID: 224 (Image Name: /usr/bin/mysqladmin)
```

(c) Process tree for the 5th iteration

Figure 5.10: TED calculation (structure only)

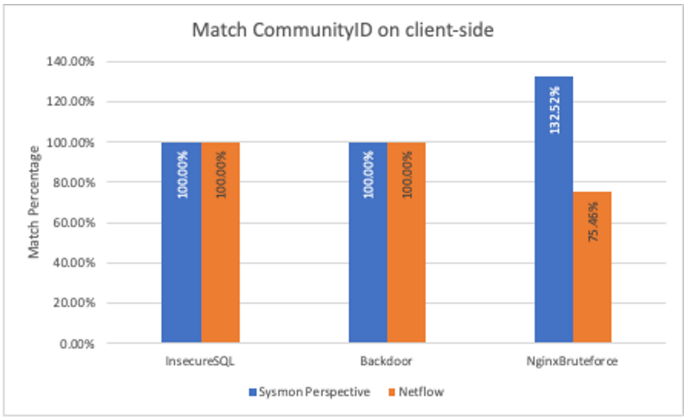
executing specific MySQL commands to set up the initial databases and configurations. Once initialised, subsequent starts of the same container may not show the same initialisation steps because the data directory is no longer empty.

5.3 Evaluation on Experiment 3

We conduct this experiment on attack scenarios including InsecureSQL, Backdoor and NginxBruteforce within 60s. In this experiment we calculate Community ID for sysmon connection and NetFlow data. We then calculate the match percentage from each perspective. From the sysmon point of view, the match percentage is a fraction of sysmon unique Community ID over netflow unique Community ID that shares the same values, and vice versa for the network flow point of view.

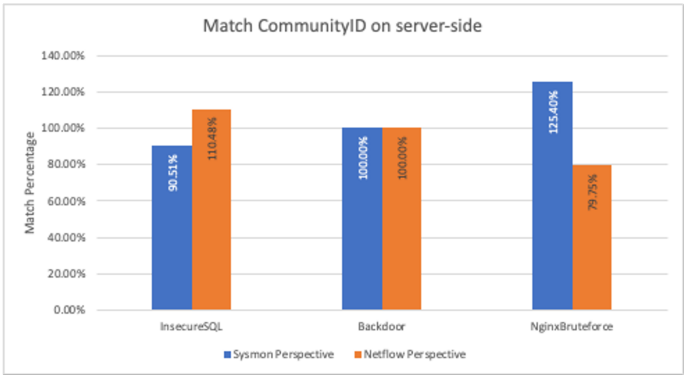
Figure 5.11 represent matches CommunityID on client-side from Sysmon and Netflow perspectives. We got 100% match for InsecureSQL and Backdoor scenario. However, in NginxBruteForce scenario, sysmon capture more connections than the netflow. We find the same pattern as in the previous Nginx scenario, where client-side sysmon generates more connection data than the tcpdump.

However, in NginxBruteForce scenario we find a different pattern for the server side (Figure 5.12). Server-side Sysmon also generate more connection data than netflow. We suspect that this is because the bruteforce attack behaviour constantly generates server traffic.



(b) Matching Community ID on client side

Figure 5.11: Matching Community ID, both from sysmon log and NetFlow log point of view



(a) Matching Community ID on server-side

Figure 5.12: Matching Community ID, both from sysmon log and NetFlow log point of view

Community ID Value	Netflow	Sysmon
1:+8Fb9IAKRuJ34zvTamvZYP57ka0=	1	1
1:+sPLUyuH0PMlYkE2ueXEyslFV5s=	1	1
1:+tZ/bqhLf9azoKlibjhPtAB2pE=	1	1
1:+yCzdKbFKTkGnSpepkfWW0dbj1g=	1	1
1:/6kTCCdXuhRQzAFtSMXGPmUH/rY=	1	1
1:/MYwDrRDiGSNgNhL51kt0+Jo/ZU=	1	1
1:/SoxdOa3hdLhWXFabOYqoiNZgE8=	1	1
....	....	....

Figure 5.13: Sampled of match CommunityID

Chapter 6

Conclusion

6.1 Discussion on the Design and Experimentation

6.1.1 Discussion on Design

We have designed a framework to generate synthetic sysmon data from selected scenarios in the DetGen framework. Several challenges and critique that arise are identified as follows:

- Challange in capturing very short-lived processes. For this kind of Scenario, we employ a workaround which uses process names directly to identify ProcessId, specifically on InsecureSQL scenarios; this solution solved the issues.

  A drawback in our technique is that we do not identify processes in container scenarios that employ *watch* command. The scenarios using this mechanism are NginxSSL, ApacheSSL, and WordPress. For instance, in the NginxSSL scenario, the HTTP request is performed by executing *command: watch -n 5 wget "http://172.16.236.15" -P /data* as stated in the corresponding Docker-Compose file. For every Scenario that employs this approach, we cannot identify every process that runs after *watch* command, in this example *wget*. Given this condition, we are able to generate corresponding synthetic sysmon for scenarios using this approach up to EventID 1,5,11 or 23.

- During the experimentation, we also found containers that do not work correctly in our machine; it can be observed from the generated pcap file. These scenarios include Heartbleed, where no proper pcap was generated and also no Event ID 3 was generated. We suspect this is due to the payload loaded by the attacker

container containing a running msf console that is not allowed to run in GCP, but this requires further investigation. However, Event ID 1,5,11 are correctly generated.

- Another scenario that is not run as we expected is Mirai botnet. We experience an intermittent blocking problem when the attacker's script runs telnet command; this issues persist also for the pcap file generated by the tcpdump containers. Considering the time constraint in this dissertation, this problem needs to be investigated further. We suspect that this is due to machine configuration. However, there are complete EventIDs for bot1, bot2, bot3, and cnc (Event ID 1,3,5, and 23), while attacker and apache generate Event ID 1,5 and 11.

6.1.2 Disucssion on Experimentation

The experiments are arranged to evaluate the correctness, completeness, determinism, and utility. Given these criteria, we have conducted three experiments where the experiment design and results are presented in Chapter 3 and 4. In this section we discuss the challenge and the critique to our approached used in this project.

- **Experiment 1**. The aim of this experiment is to evaluate the correctness of the generated data. Firstly, we confirm that the the synthetic sysmon is correctly generated. It can be seen from the result of field comparison experiment between the queried, normalized, and sanitized sysmon and the original sysmon stored in the /var/log/sysmon. Secondly, for the containers that run properly and also employ suitable command with our data generation method, we verify that the data is complete, as EventIDs are generated for each corresponding scenarios.

  In the field-to-field experiment, we employ sampled manual field comparison to show that the generated synthetic sysmon data is correct. Although we utilised information from reference tuples (per program implementation) that contain accurate information on PID Namespace mapping, the experiment could be automated and expanded to more scenarios.

- **Experiment 2**. This experiment is design to measure the determinism. As our experiment on simple and complex scenario, we can see that across repetition of running the scenarios, we find that our data generation are repeatable. The result show consistent pattern in terms Event ID generated. Specifically we delve on Event ID 3 by comparing network connection pattern with corresponding network

flow extract from pcap files per scenarios. We observed that connection counts across iteration is consistent. In the TED calculation on the process tree across iteration, we observe the consistent result.

For the Scenario that generates HTTP traffic. On the server side, there are different connection counts between synthetic sysmon and NetFlow. We suspect several conditions. Firstly is that Sysmon's EventID 3 logs network connection events. If software reuses a connection (like persistent HTTP connections in the keep-alive state), it might not necessarily create a new EventID 3 event for every data exchange on that connection. Secondly, a connection might be observed in the pcap in the initial stages (SYN, SYN-ACK) but might only be established partially due to various reasons (like RST being sent). Sysmon might log only fully established connections in some configurations. This presumption needs further investigation.

- **Experiment 3**. The main objective of this experiment is to demonstrate the utility of the generated data. Given the idea of data fusion, we can detect and confirm attack from different log sources to minimise false positive. Our experiment show that the synthetic Sysmon can be fully utilised for data correlation on the solution like SIEM. We demonstrate that we can compute the correct Community ID for both synthetic and netflow data, so that the event correlation can be conducted.

  We perform semi-manual data pipelining in the experiment from data generation, prep-rocessing, and processing. This process should be conducted in a more automated manner so that it is scalable to cover more scenarios. The automated process also minimises error.

Common aspect that could be improved is to include more scenario in each experiment, so that we can draw a more generalised conclusion.

6.2 Future Work

Considering the design challenges in the previous section, investigating Designs 2 and 3 could be beneficial to compare the practical approach in generating synthetic sysmon data in terms of completeness to tackle the drawback mentioned in the discussion section above. Secondly, generating more scenarios suitable for detecting endpoint attacks is also favourable. For example, scenarios that generate complex process trees

in some malware variants can be considered. This Scenario can also be employed to build a pipeline for testing detection rules before deploying into production to minimise false positives on generated alerts.

6.3 Conclusion

Data plays a critical role in anomaly detection. Using real-world quality data faces challenges, such as privacy issues, rare events, and the requirement of standardised data for objective comparison. This is where synthetic data generation techniques become crucial. However, there are still concerning matter for this method in terms of its quality, variability, and extensibility.

This dissertation aims to build a framework to generate synthetic sysmon data leveraging existing DetGen framework scenarios that satisfy the above criteria. We measure the quality of the data by its correctness, completeness, determinism, and usefulness. Our experiment shows that for every Scenario with the suitable command that runs correctly on our machine, we can generate synthetic sysmon data that fulfil such quality defined in our requirements. We generated correct and complete synthetic sysmon data according to our chosen scenarios. However, we only test some of the data due to the time constrain of this dissertation.

Regarding variability and extensibility, our work offers a valuable approach to generating sysmon data where more variation attacks and benign scenarios can be built.

Bibliography

[1] Arthur G Bedeian and Kevin W Mossholder. On the use of the coefficient of variation as a measure of diversity. *Organizational Research Methods*, 3(3):285–297, 2000.

[2] Daniela Brauckhoff, Arno Wagner, and Martin May. FLAME: A flow-level anomaly modeling engine. In Terry Benzel, editor, *Workshop on Cyber Security and Test, CSET'08, San Jose, CA, USA, July 28, 2008, Proceedings*. USENIX Association, 2008.

[3] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutorials*, 18(2):1153–1176, 2016.

[4] Henry Clausen. *Traffic microstructures and network anomaly detection*. PhD thesis, School of Informatics, The University of Edinburgh, 9 2021.

[5] Henry Clausen, Robert Flood, and David Aspinall. Traffic generation using containerization for machine learning. *CoRR*, abs/2011.06350, 2020.

[6] Henry Clausen, Robert Flood, and David Aspinall. Controlling network traffic microstructures for machine-learning model probing. In Joaquín García-Alfaro, Shujun Li, Radha Poovendran, Hervé Debar, and Moti Yung, editors, *Security and Privacy in Communication Networks - 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6-9, 2021, Proceedings, Part I*, volume 398 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 456–475. Springer, 2021.

[7] Richard Colbaugh and Kristin Glass. Proactive defense for evolving cyber threats. In *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, pages 125–130, 2011.

[8] Carlos Garcia Cordero, Emmanouil Vasilomanolakis, Aidmar Wainakh, Max Mühlhäuser, and Simin Nadjm-Tehrani. On generating network traffic datasets with synthetic attacks for intrusion detection. *ACM Trans. Priv. Secur.*, 24(2):8:1–8:39, 2021.

[9] Carlos Garcia Cordero, Emmanouil Vasilomanolakis, Aidmar Wainakh, Max Mühlhäuser, and Simin Nadjm-Tehrani. On generating network traffic datasets with synthetic attacks for intrusion detection. *ACM Trans. Priv. Secur.*, 24(2):8:1–8:39, 2021.

[10] Crowdstrike. 2022 crowdstrike global threat report. https://www.crowdstrike.com/resources/reports/global-threat-report/. Last accessed 6 January 2023.

[11] Gustavo de Carvalho Bertoli, Lourenço Alves Pereira Júnior, Osamu Saotome, Aldri L. dos Santos, Filipe Alves Neto Verri, Cesar Augusto Cavalheiro Marcondes, Sidnei Barbieri, Moises S. Rodrigues, and José M. Parente de Oliveira. An end-to-end framework for machine learning-based network intrusion detection system. *IEEE Access*, 9:106790–106805, 2021.

[12] Dorothy E. Denning. An intrusion-detection model. *IEEE Trans. Software Eng.*, 13(2):222–232, 1987.

[13] Docker. Docker compose overview. https://docs.docker.com/compose/. Last accessed 8 August 2023.

[14] Docker. Use containers to build, share and run your applications. https://www.docker.com/resources/what-container/. Last accessed 8 August 2023.

[15] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the CICIDS2017 case study. In *IEEE Security and Privacy Workshops, SP Workshops 2021, San Francisco, CA, USA, May 27, 2021*, pages 7–12. IEEE, 2021.

[16] Robert Flood. A data-driven toolset using containers to generate datasets for network intrusion detection. Master's thesis, School of Informatics, University of Edinburgh, 2019.

[17] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. ADASYN: adaptive synthetic sampling approach for imbalanced learning. In *Proceedings of the*

International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008, pages 1322–1328. IEEE, 2008.

[18] Hanan Hindy, David Brosset, Ethan Bayne, Amar Kumar Seeam, Christos Tachtatzis, Robert C. Atkinson, and Xavier J. A. Bellekens. A taxonomy of network threats and the effect of current datasets on intrusion detection systems. *IEEE Access*, 8:104650–104675, 2020.

[19] Scott v Kalken. What are namespaces and cgroups, and how do they work? https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/. Last accessed 8 August 2023.

[20] kddcup99 mld. kddcup99 dataset. Last accessed 6 January 2023.

[21] Karen Kent and Murugiah Souppaya. Guide to computer security log management, 2006.

[22] Ilhan Firat Kilincer, Fatih Ertam, and Abdulkadir Sengür. Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Comput. Networks*, 188:107840, 2021.

[23] LINCOLN LABORATORY. 1998 darpa intrusion detection evaluation dataset. Last accessed 6 January 2023.

[24] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Comput. Networks*, 34(4):579–595, 2000.

[25] Hongyu Liu and Bo Lang. Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences (Switzerland)*, 9, 10 2019.

[26] Mandiant. M-trends 2022 insights into today's top cyber security trends and attacks. https://www.mandiant.com/m-trends. Last accessed 6 January 2023.

[27] Karl Matthias and Sean P Kane. *Docker: Up & Running: Shipping Reliable Containers in Production*. " O'Reilly Media, Inc.", 2015.

[28] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[29] Microsoft. Microsoft digital defense report 2022. https://www.microsoft.com/en-us/security/business/microsoft-digital-defense-report-2022. Last accessed 6 January 2023.

[30] Microsoft. Sysmon v15.0. https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon. Last accessed 8 August 2023.

[31] Nour Moustafa and Jill Slay. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 Military Communications and Information Systems Conference, MilCIS 2015, Canberra, Australia, November 10-12, 2015*, pages 1–6. IEEE, 2015.

[32] Sergey I. Nikolenko. Synthetic data for deep learning. *CoRR*, abs/1909.11512, 2019.

[33] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[34] Nigel Poulton. *Docker Deep Dive: Zero to Docker in a single book.* NIGEL POULTON LTD, 2020.

[35] Nadun Rajasinghe, Jagath Samarabandu, and Xianbin Wang. INSECS-DCS: A highly customizable network intrusion dataset creation framework. In *2018 IEEE Canadian Conference on Electrical & Computer Engineering, CCECE 2018, Quebec, QC, Canada, May 13-16, 2018*, pages 1–4. IEEE, 2018.

[36] Roberto . Rodriguez. Automating the deployment of sysmon for linux and azure sentinel in a lab environment. https://techcommunity.microsoft.com/t5/microsoft-sentinel-blog/automating-the-deployment-of-sysmon-for-linux-and-azure-sentinel/ba-p/2847054. Last accessed 8 August 2023.

[37] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards a standard feature set for network intrusion detection system datasets. *Mob. Networks Appl.*, 27(1):357–370, 2022.

[38] Karen A. Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps), 2007.

[39] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Paolo

Mori, Steven Furnell, and Olivier Camp, editors, *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, pages 108–116. SciTePress, 2018.

[40] Kevin Sheldrake. Sysmon for linux. https://github.com/Sysinternals/SysmonForLinux. Last accessed 8 August 2023.

[41] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput. Secur.*, 31(3):357–374, 2012.

[42] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 305–316. IEEE Computer Society, 2010.

[43] Joel Sommers, Vinod Yegneswaran, and Paul Barford. A framework for malicious workload generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 82–87, 2004.

[44] Andrew S Tanenbaum and Albert S Woodhull. *Operating systems: design and implementation*, volume 68. Prentice Hall Englewood Cliffs, 1997.

[45] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009, Ottawa, Canada, July 8-10, 2009*, pages 1–6. IEEE, 2009.

[46] UNB. Nsl-kdd dataset. Last accessed 6 January 2023.

[47] Emmanouil Vasilomanolakis, Carlos Garcia Cordero, Nikolay Milanov, and Max Mühlhäuser. Towards the creation of synthetic, yet realistic, intrusion detection datasets. In Sema Oktug, Mehmet Ulema, Cicek Cavdar, Lisandro Zambenedetti Granville, and Carlos Raniery Paula dos Santos, editors, *2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016, Istanbul, Turkey, April 25-29, 2016*, pages 1209–1214. IEEE, 2016.

[48] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[49] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yan-Miao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018.

[50] Dingbang Xu and Peng Ning. Privacy-preserving alert correlation: A concept hierarchy based approach. In *21st Annual Computer Security Applications Conference (ACSAC 2005), 5-9 December 2005, Tucson, AZ, USA*, pages 537–546. IEEE Computer Society, 2005.

[51] Pavel Yosifovich, David A Solomon, and Alex Ionescu. *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*. Microsoft Press, 2017.

[52] Lei Zeng, Yang Xiao, Hui Chen, Bo Sun, and Wenlin Han. Computer operating system logging and security issues: a survey. *Security and Communication Networks*, 9(17):4804–4821, 2016.