A Container Driven Toolkit to Replicate Network Intrusion Datasets

Euan Wallace

4th Year Project Report
Artificial Intelligence
School of Informatics
University of Edinburgh

2023

Abstract

To enable the use of the ever-evolving internet without placing the clients or servers at risk is a complex task which has been developed since the existence of wide area networks. To disable attackers from obtaining sensitive data we must place safeguards on network access. Nonetheless, with the immense increase in network traffic over the past few decades not only through entire networks but overall traffic points we need to develop methods to protect the users of the network. To better understand intrusion detection researchers often seek out network traffic data to generate learning models in order to correlate and classify reasons for intrusions. Research to improve the safety of a network has proven to be a more difficult task than meets the eye as a plethora of data protection laws and public safety stops the ability to create datasets on real users' traffic data. Researchers will often try to anonymise human data to protect their privacy but this is not efficient and is often time-consuming. In recent research, most opt to develop artificially generated data to base their research on but this can come with many flaws with crowded noisy data which doesn't accurately emulate normal day-to-day network traffic. This report will scrutinise the standard of one of the gold standard research data sets "CICIDS2017" analysing some of its downfalls and limitations to possibly generate a more up-to-date traffic alternative to inject or replace the attack scenarios in the dataset. We will generate this traffic using scripted scenarios created from Docker containers. Using the already created framework "DetGen" we can expand the library of available scenarios that can generate both benign and attack network traffic to perform a series of experiments to explore and compare the quality of the network traffic.

Research Ethics Approval

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.

(*Euan Wallace*)

Acknowledgements

Thank you to my parents and brother for providing their invaluable support over the course of this project. Thank you to my supervisor David Aspinall and Robert Flood, both of whom provided tremendous help and constructive criticism throughout the research, implementation and presentation of this project.

Table of Contents

Chapter 1

Introduction

In recent research, the number one issue of experimentation and understanding of anomaly detection is the difficulty of generating a valuable dataset [6]. Lack of availability to users' real-world traffic due to privacy issues makes the accumulation of valuable network data scarce. While some datasets do exist, they often lack detail and enough traffic volume to provide quality experimentation. CICIDS2017 (Canadian Institute for Cybersecurity Intrusion Detection System) [39] is a popular industry standard dataset used to train machine learning models and experiments to better understand intrusion detection systems(IDS). The research findings showed a distinct lack of publicly available datasets with most research running experiments on CICIDS2017, CICIDS2018 [39], HIKARI-2021 [12] or internal variants thereof. Rather than accept these as the gold standard, this research intends to improve some of the fundamental flaws of these datasets using DetGen [5] to provide a more complete dataset with a variety of attacks while maintaining similarity to the CICIDS2017 dataset. While most modern datasets have a large variety of data, it is static, limiting researchers to the variability of the given attack traffic. Introducing tooling to artificially generate network traffic allows us to add variability to artificial datasets while simulating real-world traffic data characteristics.

Generating artificial network traffic is not a new research development. However, generating high-quality datasets that correctly emulate natural human network traffic proves to be a significant challenge. One of the main challenges is ensuring that the generated traffic accurately reflects real-world patterns and behaviours. This includes simulating the types and frequencies of different traffic flows, as well as the various protocols that are used in real network communications. Another difficulty is handling the large amounts of data that are generated during the process, as this requires sufficient storage and processing resources. Additionally, it can be challenging to accurately model the various factors that can affect network traffic, such as network congestion, delays, and packet loss, which can impact the accuracy of classification in the generated data. This data must then be trained on machine learning models to create an IDS (Intrusion Detection System), which then can be evaluated using a simple test and train data split. While these datasets produce high accuracy scores, the standard for intrusion detection systems is very high, as any malicious intrusions leaking into the network can have

catastrophic consequences.

The Docker [8] container framework will be utilized to create these datasets. This approach enables the separation of the traffic data from the host environment. Using Docker produces a more lightweight solution for creating virtual environments inside a virtual network without the presence of noisy traffic generated from a typical PC by separating the application from the host environment.

The objective of this research is to demonstrate the effectiveness of using containers to generate deterministic traffic of malicious network traffic to improve the quality and variety of the CICIDS2017 dataset. Additionally, the study will investigate the limitations and obstacles involved in using containers for generating network traffic. This includes the likes of enforcing traffic latency on individual containers, in order to artificially vary packet statistics of any given traffic scenario. This study will also explore other potential opportunities to enhance the realism and variability of the datasets.

Chapter 2 provides a review of the relevant background information. In Chapter 3, the design and implementation of traffic-generating containers using Docker scenarios are discussed, along with the requirements placed on traffic generation. The experimentation of the toolkit is detailed in Chapter 4, highlighting the motivation, design, and methodology of each conducted experiment. In Chapter 5, the experiment results are explained, and the overall quality of scenarios and solutions are evaluated, while also identifying the key difficulties and limitations in the work. Finally, in Chapter 6, the work is discussed and critiqued, along with potential future work.

Chapter 2

Background Research

2.1 VMs and Containers

Containers eliminate the problems when creating code in one environment and not being able to replicate it in another, by bundling together application code with related libraries, configuration files and dependencies required to run them. When trying to capture network traffic between two devices, filtering out irrelevant transmissions can become difficult due to the sheer volume of network traffic that a computer uses to communicate across a network. Using containers isolates applications from their host network, wrapping them in an executable and isolated software package. This encapsulated all the network traffic into the isolated network produced when creating a container. This tooling enables the isolation of network traffic events and captures network interactions that are solely affected by the application's interaction with other devices or hosts on the network.

Taking advantage of this, the DetGen [5] project produces containers using Docker to generate deterministic traffic data. DetGen relies on the use of containers, a more lightweight and efficient form of virtualisation. Virtual environments are lightweight isolated packages that simulate hardware and software; this allows servers to simulate multiple operating systems in a single environment. In a virtual environment, each virtual system is given its own section of memory which is separate from the host OS (Operating System) unless specifically specified. Running a virtual environment necessitates the use of a hypervisor, a program used to run and manage one or more virtual machines running on a host machine. This program uses significant memory which is why, recently, a switch to using a more lightweight system in the form of containers is becoming ever more popular. Containers make the use of a hypervisor redundant, as the shared resources are represented in kernel artefacts.

2.1.1 Docker

Docker containers are created from Docker images, which are similar to a snapshot of a particular application or environment. Once a container is created, an application or environment can be run inside it. Containers isolate the application or environment from

the host's machine to make the running of the container independent of the setup of the host environment. This self-contained nature of containers allows these applications to be packaged and run on any host without the host environment affecting the application at runtime.

Due to the nature of the traffic generated to train an IDS, it is important to isolate the network traffic if it contains malicious flows (i.e. security attacks). Docker's container environments allow us to produce malicious traffic data from its suite of attack tools without compromising a user's host machine. The isolated nature of applications within a container insures no malicious traffic can leak into the host network, allowing users to be able to vigorously test malicious traffic tools without compromising their host machine. Docker containers are made up of multiple parts which are used to build, manage and run a packaged application.

- **Docker objects**: When utilizing Docker [8], images, containers, networks, volumes, and plugins are created and used. Docker then packages all of these into a single assembly using its image-container architecture to make lightweight applications.
- **Containers**: A container is an instance of a Docker image which packages the code and dependencies defined in the Dockerfile from the created image. This instance can then run the defined application, regardless of the host's OS or other changes in the environment. A Docker container can then be run virtually anywhere as it virtualises CPU, memory, storage, and network resources at an OS level. When a container has been removed, any changes to its state that are not stored in persistent storage are deleted from memory.
- **Images**: A Docker image is a snapshot of a filesystem used to execute code in a Docker container. Docker images can be viewed as a template to build a container. The Docker image contains the application code along with all the required libraries, dependencies, and tools required to run the application code. A Docker image can be used to create one or many instances of the container depending on the requirements. A Docker image is made up of multiple layers, which all contribute to the build environment.
- **Dockerfile**: A Dockerfile is a set of build instructions required to build an image. These instructions specify what base image to build from, what packages/libraries to install and what commands are required to set up the container environment.
- **Docker-Compose**: Docker-Compose is a tool used for defining and running multi-container Docker applications. It allows users to describe their application's infrastructure in a declarative way using a YAML file and then spin up the entire application stack with a single command. With Docker Compose, users can easily create and configure multiple containers that work together as a single application. They can specify the dependencies between the containers, such as which containers need to be started first, and how they communicate with each other.

```
#Dockerfile
FROM python:3
RUN apt-get -y update
RUN apt-get install -y git
RUN git clone https://github.com/gkbrk/slowloris.git
WORKDIR slowloris
RUN ls -al
ENTRYPOINT ["python3","slowloris.py"]
CMD ["https://www.google.com"]
```

Figure 2.1: Example of DoS SlowLoris Dockerfile

2.2 Intrusion Detection

Intrusion detection is the antithesis of network attacks. Network attacks are when a malicious user performs actions on a network to alter, steal, destroy or deny access to often private data; intrusion detection systems are used to flag attempts on this behaviour. There are two main types of IDS: Host-based IDS and Network-based IDs.

- **Host-Based IDS**: A host-based IDS is deployed on a particular endpoint of a network (server, user's laptop, etc) and designed to protect it from malicious internal and external threats. An IDS has the ability to monitor incoming and outgoing network traffic, observe running processes on the host machine and inspect system logs to flag and act upon malicious behaviour. A host-based IDSes traffic visibility is limited to the local machine so lacks the ability to gain context of the traffic for decision-making but has a deeper visibility into the effects traffic causes onto the computer's internals.
- **Network-Based IDS**: A network-based IDS is used to monitor an entire network. It has visibility of all traffic on its own network domain and makes determinations on malicious traffic based on packet metadata and packet content. This wider viewpoint gives us more context to network traffic flows, which gives us the ability to examine widespread threats but lacks visibility into how the attack affects the internals they protect.

After data collection either at either the host level or network level, the IDS is designed to observe network traffic and match traffic patterns to known attacks; this is often called pattern correlation. IDSes commonly use two key detection methods: Signature-based detection and Anomaly-based detection.

- **Signature-based Detection**: Signature-based detection uses fingerprints of known threats to identify them. Once malware or malicious content has been identified, it is given a signature and added to a list that the IDS uses to test incoming traffic. This allows us to achieve a very low false-positive rate as the IDS will only flag known-malicious content. While a low false positive rate is beneficial, the design of this method limits us in detecting unknown threats or zero-day attacks (attacks that vendors are unaware of or have never seen).
- **Anomaly-based Detection**: Anomaly-based detection is based on a model built

on the "normal behaviour" of the system. This model is built on machine learning methods and any anomalies are flagged and labelled as threats. While this system detects unknown or day-zero threats, due to the fundamental difficulty of building a detection model using machine learning, the system has to balance the false-positive and false-negative rates.

Most common day IDS systems use "Hybrid Detection" which is an implemented combination of both signature-based and anomaly-based detection to produce a system which detects more attacks with a lower error rate than using either in isolation.

2.2.1 Machine learning methods

Machine learning algorithms can help Intrusion Detection Systems (IDS) to detect anomalies and patterns in network traffic, system logs, and user activity. When a deviation from normal behaviour is detected, security teams can investigate and respond to the incident. Machine learning can also detect sophisticated attacks that traditional methods may miss and can respond to incidents in real-time. However, finding a reliable dataset to distinguish between benign and malicious traffic is a challenge for IDSes [21]. Some common machine learning methods used for intrusion detection are as follows:

- **K-means Clustering**: Groups similar data points into K clusters by iteratively assigning data to the closest cluster based on weighted features of the dataset and updating the centroids of the clusters until convergence. Muda et al [29] highlights using this method as an effective way to highlight anomalies within network traffic.
- **Bayes Network**: Probabilistic graphic model. Takes advantage of the Bayes rule to use the condition dependence by plotting the dependencies on the edge of a directed graph. Makes the assumption nodes are conditionally independent (inferred by the Bayes rule). Muda et al [29] also perform classification using this method with successful results, but a combination of Bayes networking with K-means clustering allowed models to best classify anomaly data.
- **Random Forest Classifier**: A popular method for training IDS which combines many individual decision trees together into an ensemble. Each individual decision tree makes a classification; the decision with the majority classified in the ensemble is then defined as the final classification. Farnaaz et al [11] explores the quality of this training model to classify anomaly network traffic with accuracy exceeding 99% across multiple training sets.

2.3 Problems with modern datasets

Due to privacy regulations and ethical issues, it is almost impossible to share malicious or benign traffic from real-world traffic captures. Sharing such data without proper consent or anonymization can be a violation of privacy. Moreover, if the data is not properly secured, it could be exploited by malicious actors for criminal activities such as identity theft or fraud. In most prevalent academic datasets, researchers rely on the production of synthetic network traffic data to evaluate the quality of an intrusion

detection system. Researchers have discussed the disadvantages of this at length comparing real-time traffic data with that of synthetic generation [44].

2.3.1 Dataset Quality

There are a number of significant challenges that modern researchers face when producing a useful dataset for IDS [21, 40]:

- **Data Labelling**: Supervised machine learning requires the dataset to be labelled in order to be trained on. Lack of labelling or inaccurate/incorrect labelling.
- **Network Variety**: Creating an artificial dataset that emulates a variety of IP addresses allows us to better emulate a real network situation of supplying multiple clients and servers which can connect and interact with each other. A distinct lack of address or a lack of connections between IPs would not emulate natural human network traffic.
- **Dataset Size**: In order to perform any kind of machine learning, we need a sufficient volume of data. While the quality of data is paramount to performing valuable analysis, without a sufficient volume of data our analysis cannot be solidified.
- **Variety of Attacks**: Creating a dataset that can cover network flows for a plethora of attacks allows users of the data to create learning models for a wider range of attacks and better understand how an IDS should react to these attacks.
- **Benign traffic**: For a synthetic dataset to appear authentic it must emulate other traffic outside of network attacks. Providing traffic that would occur from legitimate users would improve the dataset as it will perform more like real-world network traffic.

2.3.2 CICIDS2017

CICIDS2017 (Canadian Institute for Cybersecurity Intrusion Detection Dataset) [39] is one of the publicly available datasets for IDS quality comparison. This dataset claims to contain data that resembles true real-world pcap data. Pcap data is a format for storing network traffic data. It is commonly used by network administrators, security professionals, and researchers to capture and analyze network traffic. Pcap data files contain a record of every packet that has been transmitted or received on a network, including packet headers and payloads. The producers of the dataset also offer a tool (CICFlowMeter [16]) that produces network analysis results from pcap data. CICIDS2017 generates traffic that emulates realistic background traffic; emulating traffic over 5 days from a 09:00 to 17:00 time-frame with Monday containing all benign traffic and Tuesday through Friday containing both benign and malicious traffic.

Panigrah et al [33] and Gints et al. [26] highlight clear shortcomings in the CICIDS2017 dataset which should be addressed to improve the dataset quality:

- **Lack of variation**: Synthetic datasets are generated to emulate network traffic, but the scripting used to create the traffic can be repetitive and lacking in variation.

In the CICIDS2017 dataset, a majority of the FTP transfers consist of a client node downloading the same Wikipedia page multiple times, which is not representative of real-world FTP traffic. While clearer structures in the data can lead to better accuracy for IDS machine learning models, an over-optimized model based on such data may struggle to differentiate between malicious and benign traffic in a real network setup due to the lack of variation in the synthetic data.

- **Limited size**: Machine learning models perform better with larger datasets as they can identify more complex patterns and relationships. However, the amount of data available for network traffic is limited to small virtualized networks, leading to models struggling to generalize with traffic that has small occurrences within the dataset. For example, the CICIDS2017 dataset has limited occurrences of certain types of traffic, causing ML models to treat them as noise in the data.
- **Static design**: NIDS datasets used for machine learning practices are created in a static environment where a test network of virtual machines is set up to emulate the traffic of a limited network. This approach limits the ability of the datasets to represent the ever-evolving nature of network traffic. For example, the CICIDS2017 dataset has a limited suite of vulnerabilities and only represents traffic that was prevalent at the time of its creation. The dataset's static nature makes it challenging to accurately represent real-world attacks as there is limited availability to update the traffic structures.
- **Dispersed data:** The CICIDS2017 dataset is scattered across 5 files to represent traffic for each working day. Individual file processing of pcap for training is tedious due to the lack of sufficient data labelling and information supplied in dataset documentation. The dataset would be more serviceable if sufficient information on the setup and configuration of the traffic was supplied.
- **Missing/Mislabelled values:** Dataset contains 288602 instances with missing class labels and 203 instances with absent information, removing these incomplete instances allows for a unique and complete dataset. While missing labels account for a very small amount of the dataset, we are also presented with the flow data produced for statistical analysis producing incorrect flow labels across both benign and malicious traffic.
- **Class imbalance:** 83.34% of the data is labelled as benign traffic. This presents a high-class imbalance between the majority class and the minority class traffic. A high prevalence rate of 1 class can lead to misaligned detection systems, in this case, a high rate of favouritism towards benign traffic classification. This can lead to failed detections of low prevalence classes such as Heartbleed even in models designed to extrapolate from low occurrence data as it only contains 11 instances compared to the 2359087 instances of multiple operations of benign traffic.

By merging the data across the 5 files, the combined dataset of the CICIDS2017 traffic can be represented to analyze the distribution of traffic. The classwise instances of the total original dataset can be seen in Table 2.1. As mentioned previously, there is a high skew of data volume towards benign traffic. While many of the DoS (Denial of Service) attacks have enough traffic instances to be easily recognisable, many of the significant

attacks used occur a trivial amount compared to the benign data.

Table 2.1: Traffic instances of the CICIDS2017 dataset [33]

Benign/Attack Label	Number of instances
Benign	2359087
Botnet	1966
DDoS	41835
DoS GoldenEye	10293
DoS Hulk	231072
DoS Slow-HTTPtest	5499
DoS slowloris	5796
FTP_Patator	7938
SSH_Patator	5897
Heartbleed	11
Infiltration	36
PortScan	158930
Web Attack - Brute Force	1507
Web Attack - XSS	652
Web Attack - SQL Injection	21

2.4 Packet Filtering

Wireshark is a robust tool that enables users to capture, scrutinize, and visualize network traffic in real-time. However, collected data often includes excessive noise and irrelevant information, which can make it challenging to identify specific network instances or problems. This is especially problematic when dealing with mislabelled pcap data, as it can result in large misclassifications across traffic classes when attempting to analyse and classify types of abnormal traffic patterns.

To tackle this issue, Wireshark offers a range of filtering techniques that allow users to filter captured traffic. The filter expression syntax is based on the Berkeley Packet Filter (BPF) syntax, enabling users to specify the specific protocols they want to view in the capture data.

To manually filter pcap data using Wireshark, users can employ the built-in filter expression language that allows for intricate filtering based on specific protocol fields and attributes. For instance, users may filter for all HTTP traffic by entering the filter expression “HTTP” in the display filter box. This will display only those packets that contain the HTTP protocol, eliminating all other protocols and simplifying the analysis of HTTP traffic in isolation.

Manually filtering pcap data is particularly beneficial when working with mislabelled pcap files, as it can aid in identifying the protocols that are genuinely present in the captured data. For instance, if a pcap file is labelled as containing only HTTP traffic but actually contains a combination of HTTP and SMTP traffic, manual filtering can help isolate and identify the SMTP traffic, leading to a more accurate analysis of the captured data.

2.5 Related Work

IDS datasets have been frequently studied in academic literature. A comprehensive study covering popular IDS datasets presented over the last 2 decades is presented by Ring et al. [37]. Despite the review's good coverage of prevalent issues between IDS datasets, it fails to dive into detail about the specifics of issues within the individual datasets. Instead, Ring et al. highlight issues at a high level such as data duplication and labelling errors. Gints et al. [26] conduct a detailed case study on the specifics of the errors across both CICIDS 2017/2018 and raised important issues beyond basic data labelling and size and highlight issues with the dataset construction in terms of overfitting upon the use of ML techniques and specific remarks on the errors made within payloads across different attack scenarios. Lanvin et al. [23] and Panigrahi et al. [33] highlight imbalances in traffic volume and key issues with much of the attack traffic data in CICIDS2017. This includes traffic containing empty payloads, thus are obsolete contributions to ML training for the identification of dangerous payloads.

Since 2017 many studies have deployed CICIDS2017 for research and ML studies on intrusion detection as well as the updated CICIDS2018 versions [25, 22]. The studies highlight multiple machine learning methods to evaluate the quality of detection performance of anomaly-based intrusion detection. Using these studies, Iman Sharafaldin et al. [40] analyse the issues within CICIDS2017 specifically to produce an updated version to eliminate errors within the original dataset while using the same CICFlowMeter produced by CIC. This paper also highlights the best-detected attacks within their dataset and performs machine learning algorithms to highlight the dataset quality using CICFlowMeter [24] to extract network statistics.

Other prevalent artificially generated datasets include KDDCUP99 [32] ad UNSW-NB15 [28]. These datasets both produce simulations of both benign and malicious traffic across multiple days to produce raw pcap files of network traffic interactions. Furthermore, many network traffic datasets have been produced via a variety of network traffic generation processes.

We previously highlighted that the CICIDS2017 dataset [39] was produced from a network of virtual machines. Although it is the primary dataset for comparison in this project, many other studies explore the possibilities of artificial traffic generation. Moustafa et al. [28] use a hybrid of real modern normal traffic data mixed with synthetic attack traffic to produce the UNSW-NB15 dataset to evaluate the performance of NIDS systems. Artificial data in this dataset is a mix of both benign and malicious traffic produced from the IXIA traffic generator [20]. While the produced data is unique, data is produced from a network of virtual servers similar to the CICIDS2017 dataset.

Unlike CICIDS2017 and UNSW-NB15, the KDDCUP99 dataset is one of the earliest implementations of artificial network traffic (released in 1999) and is not produced from a network of virtual machines. Instead, traffic generation in KDDCUPP99 is produced through a simulation within an isolated LAN (Local Area Network) across the period of 9 weeks. While the attack landscape for malicious attacks has massively evolved over the last 2 decades, KDDCUP99 is still used in many NIDS evaluations in modern academic literature [45, 4].

Chapter 3

Design and Implementation of Traffic Generating Containers

This project is an analysis of the quality of the CICIDS2017 dataset and to develop a suite of container environments capable of replicating the attack traffic produced in the dataset in a deterministic setup. The traffic generated from this containerized environment should be suitable for training data to an ML-based IDS system. This project aims to produce a deterministic suite of attack scenarios aligned with the CICIDS2017 dataset with the ability to alter variables such as the victim network and attack payload.

3.1 Replicating CICIDS2017

Our produced dataset should produce pcap data and network flows that emulate those of CICIDS2017. In the scope of this research, these flows will be limited to emulating the flows produced within the attacks. The CICIDS2017 represents many labelled attacks from varying sources and attack types. To replicate the dataset accurately, a deeper understanding of the tooling is required to replicate the traffic flows produces by the virtual network. Detail of the main attacks used within the dataset can be seen below:

- **DoS HULK** [15]: HULK("HTTP Unbearable Load King") is designed to overwhelm web servers by continuously requesting single or multiple URL's from a machine. What differentiates HULK from other attacks is by generating traffic volumes that are unique and obfuscated by varying the types of encrypted traffic sent from variable IP addresses also varying between HTTP, HTTPS and DNS. HULK bypasses caching engines to directly hit the server's resource pool, making it harder to detect by an IDS system.
- **DoS GoldenEye** [38]: GoldenEye is designed to overwhelm a web server by continuously requesting single or multiple URL's from a machine, similar to DoS "HULK". GoldenEye generates requests dynamically by randomizing its user agents, referrers and other traffic parameters. GoldenEye's benefit is that it uses completely legitimate HTTP traffic to flood a machine's web server. It makes an

honest full TCP connection and then floods the network by performing requests at large time intervals to keep connections open. This technique is beneficial as the tool doesn't need to generate a lot of traffic to flood the target network

- **DoS Slowloris** [49]: Slowloris is an application layer attack which operates on partial HTTP requests to open many connections between the attacker and target and keep them open as long as possible. The target server will only have a finite amount of available threads to handle concurrent connections, each thread will attempt to stay alive while waiting for a request response, which never occurs. When the server's maximum number of connections has been exceeded, all genuine additional connections will be dropped, causing a denial of service.
- **DoS SlowHTTPtest** [42]: SlowHTTPtest exploits the HTTP "GET" request to exceed the number of available open connections allowed on the target web server. This prevents genuine users from accessing and allows the attacker to open multiple HTTP connections on the same server
- **DoS LOIC** [30]: Low Orbit Ion Cannon is a client-server application with the client being the host and the server being installed to the victims' server. LOIC allows the generation of both TCP and HTTP flooding with options to specify packet interval/sizes and is sent to the victims' server. What separates LOIC from other DDoS applications is that its setup is simplified to make it accessible to users with limited technical experience
- **FTP Patator** [18]: FTP Patator performs a brute force attack to guess the FTP login password. A brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing correctly.
- **SSH Patator** [18]; SSH Patator performs a brute force attack to guess the ssh login password. A brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing correctly.
- **Port Scan** [14]: Port Scan is a method attacks use to "scope" out their attack environment by sending packets to specific ports on the host and analysing the responses to find vulnerabilities and gain information on what services and their versions are active on the host machine
- **Botnet ARES** [10]: Botnets are a group of internet-connected devices that are controlled as a group without the device users knowledge. These devices can all be controlled from a single device that the attacker controls. Botnets in this situation were activated by a Trojan which is placed on all the machines that were hijacked. A Trojan is a piece of malicious code that when activated allows the attacker to gain access to the operations of the device. The ARES botnet has a central command (usually the host machine) and a control (C2) server which issues commands to infected computers. This CC server can be located anywhere in the world that the host machine can access. The botnet usually infects victim computers using exploits or spam emails to inject malware onto the system to allow shell control on the infected machines.
- **Web attack- Brute-force** [48]: A (usually) automated tool that sends requests to a

login page of a website using a database of username and password combinations. Brute forcing tools can vary in complexity depending on the website security, perhaps including a "Captcha" or attempts limit to individual IP addresses and 2-factor authentication.

- **Web attack- XSS** [48]: Cross-site scripting injects malicious code into a website in order to steal sensitive information. There are two main types of XSS attacks: reflective XSS and persistent XSS. In a reflective XSS attack, the attack injects a malicious script into a website, which is then executed in the user's browser anytime they visit the site. In a persistent XSS, the attacker injects malicious code into a website's database which is stored and executed when any user visits the webpage. XSS attack tools vary in complexity depending on website security such as input validation and sanitation techniques embedded into the webpage.
- **Web attack- SQL injection** [48]: SQL injections insert malicious code into a website's SQL database to gain unauthorised access to sensitive information. To perform an injection, a user will insert malicious code into the website's input fields such as a search box or login form, in an attempt to execute inputted code as part of a legit SQL command to the database to gain access to specific information via given queries. Tools vary on the complexity of the website, such as the use of prepared statements and parameterized queries.
- **Heartbleed** [46]: The Heartbleed vulnerability allows an attacker to exploit a flaw in the implementation of the Transport Layer Security (TLS) heartbeat extension to read sensitive information from the memory of a vulnerable server. The information that could be obtained through a Heartbleed exfiltration attack includes passwords, private keys, session cookies, and other sensitive data that could be used to compromise the security of the server or the users whose information was stolen.

3.1.1 Determinism and Variability

Creating a suite of containers capable of hosting and executing the variety of attack options available to deterministically produce traffic data to replicate that of the original dataset. The design of the tooling also allows us to define environment variables to not only replicate but also provide alterations and improvements to the suite of attacks. For example, Gints et al. [26] highlights the misinterpretations of the implementation of the DoS HULK attack, specifically its use of a deprecated version which successfully executes the DoS attack due to forced setup configurations. Producing a tool that can alter run time variables gives us the ability to correct errors in traffic production within the dataset as well as alter the total traffic volume produced.

3.2 Traffic Requirements

For each container, the application should be capable of producing reputable results as well as correctly executing the required attacks contained within the application. Each container should be able to produce a series of "capture scenarios". These scenarios

initialise the creation of multiple containers to simulate the execution of some task over a network; producing either benign or malicious traffic. All capture scenarios will be paired with a *tcpdump* container to collect the resulting network traffic of a specific scenario. In order to establish traffic scenarios that replicate real-world connections, the traffic created by the containers is constrained by specific requirements:

1. The traffic produced should provide enough volume to provide an ML model to train on and generalise well
2. The overall traffic generated should contain a varied amount of traffic while maintaining a balance between benign and attack traffic patterns to prevent a training model from learning biases to certain traffic flows
3. Traffic production should reflect real-world traffic patterns while covering up-to-date attack types in order for the IDS training data to be relevant to current anomaly detection systems.
4. The traffic produced should focus on the anomaly component of the traffic, overfitting to certain attacks will prevent the ML model from being able to identify abnormal behaviour and will instead be generalized to specific attacks
5. The capture scenarios should be deterministic as feasibly possible.
6. A user should be able to initialise a capture scenario and collect the resulting traffic without having to interact with the system further

3.3 Meeting Design Requirements

3.3.1 Data Volume

In order to effectively replicate the CICIDS2017 dataset, we need to produce containers that can replicate the traffic volume of the current dataset. To increase our total possible traffic volume using the DetGen toolkit [5], we expand both the volume of the traffic produced and the variability of the scenarios it covers. Due to the containerized design of the network capture methods, combining the current implementation of attack patterns and DetGen traffic patterns, it is possible to produce high-volume traffic datasets.

3.3.2 Traffic Variability/Balance

When creating traffic datasets, there are often prevalent imbalances in data volume between benign and malicious traffic data. While normal traffic patterns should represent a majority of benign traffic data, an increased volume of attack traffic is required to produce variability in traffic protocols to allow an ML model to find anomalies. In essence, the dataset should not be overpopulated with malicious traffic so that a trained model would not perceive it as abnormal but should contain a high enough volume of varied traffic such that an IDS can recognise the attack pattern as well as recognise it as an anomaly. For example, In CICIDS2017 the majority of total traffic is represented in the category “benign” which cumulatively contains 83.34% of total network traffic flow. Whereas, a malicious traffic event “Heartbleed” contains only 0.00039% of total traffic

instances [33]. While this amount of traffic may be considered an anomaly as it happens so infrequently, this can also lead to an ML-based model not having enough prevalent data on the malicious action to fit the model to. As well as this, in the CICIDS2017 dataset, all executions of Heartbleed are successful. Although, we would hope that an IDS would be capable of detecting both successful and unsuccessful intrusions. To mitigate this, we need to supply variations of a given traffic pattern to represent both successful and failed traffic flows.

3.3.3 Traffic Choice / Malicious Traffic

To construct a dataset that is valuable to modern-day IDS detection, the choice of traffic has to be relevant to what standard network traffic looks like at the current time. This ensures that the dataset is up-to-date and representative of the current threat landscape. In addition, as cyber threats are constantly evolving, using a variety of attack traffic allows us to learn based on what an anomaly in traffic patterns resembles. A system trained on varied network traffic is more likely to detect day 0 attacks instead of only learning to detect the specific characteristic of already discovered attacks. While we want a dataset to be varied to cover all possible kinds of attacks, different industries will suffer from different prevalent malicious attack patterns. Creating container environments that allow for variability in traffic choice gives users of the tools the ability to bias the learning model to reflect the needs of their IDS system.

3.3.4 Determinism

Creating an exactly deterministic environment using this tooling is not guaranteed due to limitations on hardware, variability in network conditions and randomization in individual protocols. However, while we cannot guarantee each repeated scenario will elicit the exact same results, we can infer that any data being produced can be recreated with a small margin of difference between datasets. We can describe the use of containers to produce network traffic deterministic, up to limitations on hardware and network configurations. This means when running scenarios we expect the majority of the pcap data to be similar but not equal in all cases; such as the key values in a Diffie-Hellman key exchange the determined binary keys are exposed to cases of randomisation. The tooling created to execute the container scenarios is, therefore, able to produce reputable outputs such as successful brute force attempts (as long as given file systems to containers remain the same) but hardware limitations may not produce repeatable run-times for each iteration of a scenario.

3.3.5 Scenario Capture

To produce a successful tool to produce attack traffic, we want to automate all network interactions between containers during run-time without user interaction. To automate these processes, we can invoke the use of bash scripting and Docker-Compose to set up environment variables and required processes to be executed through a script which assigns the needed variables at run-time. The combination of script use with setup variables within Docker-Compose allows us to impose strict regulations on our container

setup, making the capture of scenarios to be reputable and deterministic.

3.4 Improving the dataset

The overall objective of creating these deterministic scenarios is to evaluate ways to improve the current CICIDS2017 dataset while expanding the capabilities of our implemented tooling. While current academic datasets contain many properties that are still relevant to today's anomaly detection, producing tooling that can generate new traffic upon request allows easy expansion of an ML dataset. Ansam et al. [21] made evaluations over multiple popular academic IDS datasets to evaluate their properties; while CICIDS2017 covers all the general requirements for quality ML learning, from its publishing date it has not been updated to represent the new suite of malicious traffic that occurs in current day network traffic. Advancing the DetGen tool allows us to continue to improve and evaluate new malicious traffic patterns to provide a more complete IDS dataset. Using these Docker frameworks combined with scripting allows us to produce a simple tool to both produce data but also support the variability of scenario setups.

3.4.1 Improving DoS attacks

Providing datasets with varied traffic generation scenarios allows for a better learning model. This can include the likes of a variety of thread use, including both successful or unsuccessful attacks and using up-to-date versions of common attack tools. CICIDS2017 does feature a host of different DoS attacks, but many are either deprecated or implemented in a very simple fashion. In order to improve datasets variability, we implement containers that allow the users to produce traffic while changing environment variables within the tool at runtime. For example, we can produce DoS traffic using the DoS Goldeneye tool and a hosted webserver. Through the use of Docker-Compose, we can set up this simulation environment in a contained setup and produce traffic while being able to change the number of sockets/workers as well as choose which method is used to keep connections alive. As CICIDS2017 does not provide configurations or the setup to any of the traffic produced, producing tooling with the ability to produce a variety of traffic flows allows us to generate desired traffic patterns as closely as possible if pcap/flow analysis does not produce any answers to attack configuration.

3.5 Constructing Containers

3.5.1 Scenarios

We can define a scenario as a series of Docker containers interacting across a closed network, where all network traffic can be captured over the kernel-contained Docker0 network. Running a given capture scenario should trigger the launch of one-to-many Docker containers to create a simulation of a specific interaction over a pre-defined network setup. We combine the network setup with a set of *tcpdump* [27] containers to ensure traffic between containers is complete and covers total network capture. These

tcpdump containers are injected into the same Docker namespace as the active container. This allows us to capture bidirectional traffic to all actively communicating containers in the scenario to ensure we get a full traffic overview. While a full traffic overview is beneficial, the traffic data from each perspective is mounted to a separate directory. Using these mounted traffic files, we can perform analysis on both individual containers and total network traffic data.

An example traffic capture scenario could be an online form login paired with an SQL database or a brute force attack on a ssh server for a privileged connection. Each scenario has been designed such that it can be initialised and run through a setup script. These setup scripts will allow users to make alterations to implemented attack traffic or alter variables such as files used to brute force a key to provide different traffic data. Giving users the ability to alter the inherent traffic flow setup allows us to produce sub-scenarios across individual traffic categories. This is vital for producing a quality dataset, as we enforce variability in our traffic. For example, having a Patator brute force being successful in all script configurations would not produce a variety of traffic to allow for quality training data, we combat this by allowing users to vary setup to both attack methodology and server to produce traffic that results in both unsuccessful and successful entries.

3.5.2 Capture Methods

To capture traffic over the Docker network, we will run *Tcpdump* [27] concurrently in the network. *tcpdump* is a powerful command-line packet sniffer tool used for capturing, logging, and analysing network traffic. It is capable of intercepting, logging, and analysing packets for the purpose of diagnosing network-related issues. *Tcpdump* supports various network protocols such as TCP, UDP, ARP, ICMP, and many others. This allows us to capture traffic from the perspective of every container, such that we would have a complete view of the network traffic. This allows us to collate flows from all container perspectives for analysis.

3.5.3 Implementation Steps

The process to design and implement a scenario broadly follows a set number of steps:

1. Identify containers that are required to execute desired network traffic, as well as identify the primary container required to provide that service. Build a Dockerfile containing all necessary setup requirements to execute that application.
2. Identify environment variables required to provide alterations to the service. For example, altering the number of sockets used for a DoS attack
3. Design and implement Dockerfiles to provide necessary behaviour to interact with the primary container. For example, providing an SSH server for SSH-Patator to interact with
4. Add *tcpdump* [27] to every container on the network to provide traffic flow of a full network view

5. Create Docker-Compose files to launch Dockerfiles simultaneously. These Docker-Compose files will collate the containers together to launch and run concurrently on the Docker network.
6. Write runtime scripts to launch the containers. These scripts pass variables required by the user for runtime and launch the required containers through the Docker-Compose files.

To keep the setup as close to a clean Docker setup as possible, we try to limit the frameworks of Docker-Compose files to execute a single service for a specific purpose as well as keep the Dockerfiles within the Docker-Compose setup limited to as few required dependencies as possible. This skeleton setup allows us to easily update the code for network interaction, such as to keep in line with the updated version of application software.

3.5.4 A simple scenario – FTP Patator

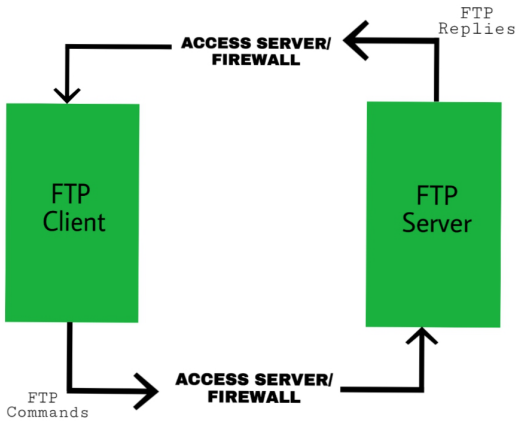

Figure 3.1: FTP Patator

In Figure 3.1 we can see a basic diagram of a standard interaction between a client and server using FTP across an access control firewall. The Patator tool aims to exploit this to gain access to the server. The interaction can be initiated by a single DetGen script, which allows the user to specify the username and password files they wish to use to try to brute force the login of the access firewall/server protection of the FTP server while being able to bypass error messages and set retry limits to test possible time-based login methods. The script sets up the required username and password values to launch the server, as well as pass the necessary runtime commands needed to execute the Patator tool. Subsequently, the containers are started, which first launches the VSFTP server and then the Patator tool. These two containers will both run *tcpdump* to produce pcap files of the traffic interactions from both perspectives. Once the Patator tool executes, regardless of success rate, the pcap files will be saved into a shared volume. An example

snippet of some of the network traffic from this scenario can be seen below in Figure 3.2.

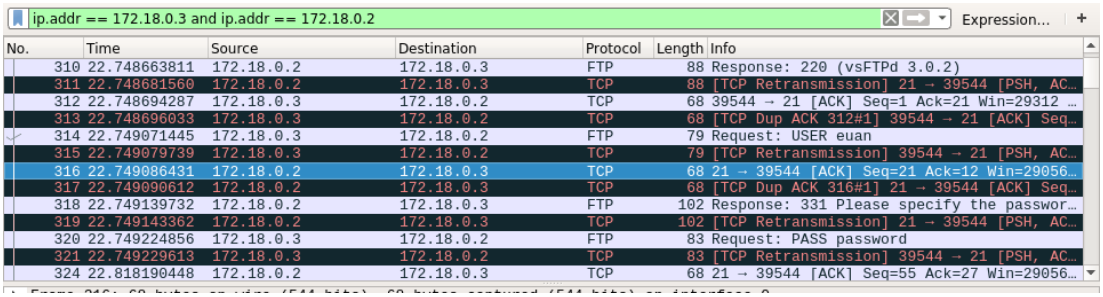

Figure 3.2: FTP Patator Wireshark traffic

3.5.5 Complex example scenario- DDoS LOIC

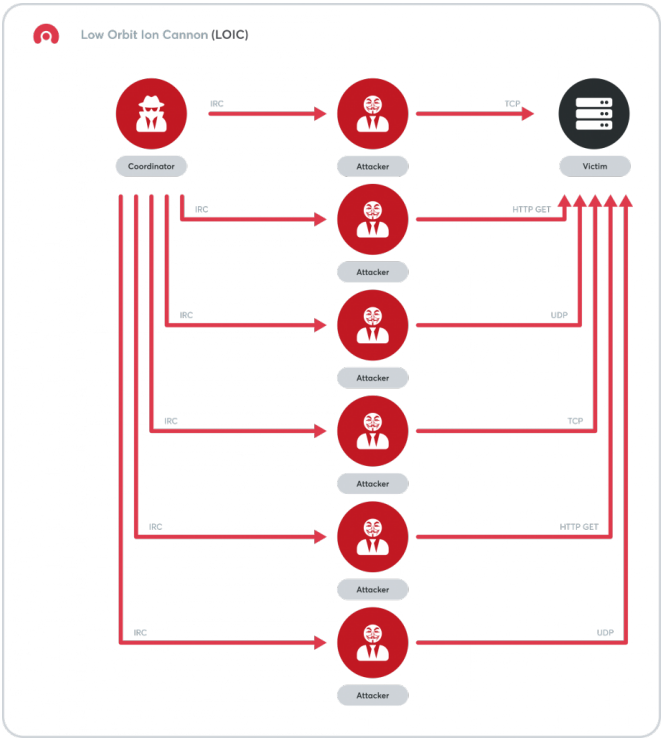

Figure 3.3: DoS LOIC configuration [1]

We outline the implementation of a user-friendly DoS attack interface. We can see an example DoS LOIC configuration using multiple users to attack a single victim in Figure 3.3. This tool allows users to flood a target system with TCP, UDP or HTTP GET requests. This tool also provides a user-friendly interface to a DoS tool making it accessible to novice users, with the code wrapped deeply into the GUI making setup and execution a simple task.

As the LOIC tool is heavily GUI-reliant, we have to set up our Docker container to support MonoDevelop in order to run the script interface correctly. We join a web service

with the DDoS LOIC tool using a compose file in order to provide a containerized web service in an isolated environment. We then use the LOIC tooling within the container to provide a suite to generate DoS traffic. Every time we start the container, the web service will launch, and then we can access the GUI to execute the DoS attack. While this implementation requires user interaction outside the command line, it allows access to the full suite of LOIC's capabilities.

In order to successfully set up a traffic scenario, a network scenario must be fully automated from container start-up. In order to create a scripted scenario, we need to recreate LOIC's tooling to allow for arguments to be parsed at the command line as runtime arguments. To create a command line tool, we extract the relevant codebase to perform the tool execution and create a new Dockerfile to configure a containerised version of this new tooling. In our case, this tool was implemented in C, bypassing the requirements of a user to interact with a GUI to execute the tool. An example snippet of network traffic of a successful DoS LOIC attack can be seen in Figure 3.4.

No.	Time	Source	Destination	Protocol	Length	Info
28	2023-02-17 10:30:23.380408	10.8.0.5	10.8.0.6	TCP	68	[TCP ACKed unseen
29	2023-02-17 10:30:23.380444	10.8.0.5	10.8.0.6	TCP	68	[TCP ACKed unseen
30	2023-02-17 10:30:23.381202	10.8.0.6	10.8.0.5	TCP	69	[TCP Previous segm
31	2023-02-17 10:30:23.381441	10.8.0.6	10.8.0.5	TCP	69	42746 → 80 [PSH, A
32	2023-02-17 10:30:23.381740	10.8.0.5	10.8.0.6	TCP	68	80 → 42746 [ACK] S
33	2023-02-17 10:30:23.381801	10.8.0.5	10.8.0.6	TCP	68	[TCP ACKed unseen
34	2023-02-17 10:30:23.381866	10.8.0.6	10.8.0.5	TCP	69	42740 → 80 [PSH, A
35	2023-02-17 10:30:23.382119	10.8.0.6	10.8.0.5	TCP	69	42746 → 80 [PSH, A
36	2023-02-17 10:30:23.382372	10.8.0.6	10.8.0.5	TCP	69	42738 → 80 [PSH, A
37	2023-02-17 10:30:23.382442	10.8.0.5	10.8.0.6	TCP	68	80 → 42746 [ACK] S
38	2023-02-17 10:30:23.382518	10.8.0.5	10.8.0.6	TCP	68	80 → 42740 [ACK] S
39	2023-02-17 10:30:23.382485	10.8.0.5	10.8.0.6	TCP	68	80 → 42738 [ACK] S
40	2023-02-17 10:30:23.382522	10.8.0.6	10.8.0.5	TCP	69	42736 → 80 [PSH, A
41	2023-02-17 10:30:23.382568	10.8.0.6	10.8.0.5	TCP	69	42744 → 80 [PSH, A

Figure 3.4: DoS LOIC Wireshark traffic

3.5.6 Dataset Creation

Using our current traffic generation framework, traffic is generated on a single interaction basis, but it is possible to merge the traffic generated to create a more extensive dataset, albeit with some issues. This data generation methodology is due to constraints on the use of personal hardware as well as network constraints produced through the use of the Docker network. The accumulation of these constraints leads to the running of many containers simultaneously infeasible. Therefore, data must be accumulated over multiple periods to produce an overall dataset. If traffic is generated randomly, this leads to a lack of repeatability, in turn, making our datasets less authentic.

As discussed by Shiravi et al [43] merging distinct and unique network data in an overlapping manner can produce issues and inconsistencies in the data. To avoid inconsistencies, we collate our larger datasets by collecting data in consecutive chunks over a fixed time period. Within each chunk, we can activate traffic scenarios via scripted interactions of both benign and malicious nature. The generated pcap files from these system interactions can be merged together into a single file using *Mergecap* [36] and *Editcap* [41]. These tools allow us to edit timing values in our pcap files such that our fixed chunks occur in succession while maintaining the consistency of the generated chunk. These consistent timings lead to our overall pcap data being more cohesive and resembling that of a more "authentic" traffic interaction between two virtual hosts.

3.5.7 Implemented Scenarios

Here, we outline the scope of the scenarios we intend to implement within the DetGen tool framework. The outline of the traffic type, the software required to implement a traffic scenario and the number of containers required to emulate a traffic scenario in Docker are described in Table 3.1 and Table 3.2 below.

Table 3.1: Newly implemented attack scenarios in Docker framework

Name	Description	Tools/ Software used	Containers required	Attack Type
DoS GoldenEye	HTTP DoS Test Tool	Goldeneye, Apache	3	DoS
DoS Hulk	DoS attack on webserver	HULK-V3, Apache	3	DoS
DoS SlowHTTPtest	HTTP DoS test tool	slowhttptest, Apache	3	DoS
DoS slowloris	DoS attack on webserver	slowloris, Apache	3	DoS
FTP-Patator	Brute forcing password over FTP	VSFTP, Patator	3	Brute Force
SSH-Patator	Brute forcing password over SSH	Open_SSH, Patator	3	Brute Force
Heartbleed	Heartbleed exploit	msfconsole,Apache,OpenSSL	4	Exfiltration
PortScan	Scans available ports on target	nmap	2	Benign
Web Attack - SQL Injection	SQL injection attack	DVWA, DVWAexploits	3	Web Attack

Table 3.2: Previously implemented attack scenarios in Docker framework

Name	Description	Tools/Software Used	Containers Required	Attack Type
Miria	Miria botnet DDoS	Miria bots, Miria CnC, Telnet	12	Botnet/DDoS
Ares	Backdoor Server	Ares CnC, Apache	3	Exfiltration
Basic Bruteforce	Bruteforcing Basic Authentication	Nginx	4	Bruteforce
URL fuzzing	Bruteforcing a URL	Nginx, Patator	4	Bruteforce
XXE	XML vulnerability exploit	PHP, Apache	4	Web Attack

Using scenarios designed from this project as well as previously implemented scenarios in the DetGen framework, we can produce deterministic traffic flows for 14 unique malicious network attacks. Furthermore, the combined scenario set allows us to produce traffic for 11/18 attacks represented in the CICIDS2017 dataset. While using a combined toolkit allows us to experiment with a larger subset of the original dataset, the research in this paper will focus only on testing and evaluating the similarity, compatibility, and quality of the newly implemented scenarios using the Docker framework outlined by the DetGen toolkit.

Chapter 4

Experimentation

4.1 Experiment Design

To evaluate the scripted scenarios' effectiveness within the DetGen framework tool, we construct a series of experiments. Firstly, we want to demonstrate that the traffic generated from our Docker scenarios produces meaningful traffic data for training and testing intrusion detection systems, as well as being representative of real-world traffic. Secondly, we want to evaluate the produced traffic data against the original performance of the CICIDS2017 dataset compared to our updated traffic data. To do this, we conduct multiple experiments:

1. Experiment 1 explores using packet extraction tools to analyse our container toolkits' ability to produce traffic aligned with the CICIDS2017 dataset
2. Experiment 2 uses packet extraction tools to compare the protocol hierarchy between our Docker-generated pcap data and that of the CICIDS2017 pcap data
3. Experiment 3 aims to explore some of the issues with replicating network traffic by exploring how varying latency affects the IATs (Inter-Packet Arrival Times), a commonly used metric for traffic classification of traffic data, across specific network interactions
4. Experiment 4 aims to highlight our toolkits' ability to generalise traffic flows across datasets and unseen data, eliciting its ability to perform anomaly detection on real-world traffic

4.2 Experiment 1 - Traffic Similarity

4.2.1 Motivation

We devote considerable time to producing traffic data deterministically to match that of the CICIDS2017 dataset in order to confirm its similarity to the dataset. To properly classify and compare internet traffic into discreet flows, tooling must be used to produce deterministic and comparable metrics. Of these metrics, inter-packet arrival times

(IATs) are shown to be particularly relevant to the detection of malicious traffic [31, 50]. Therefore, to properly compare 2 instances of similar traffic in regard to intrusion detection, we want to generate metrics that allow us to perform comparisons on traffic flows and their statistical properties.

4.2.2 Traffic Choice

The original producers of the CICIDS2017 dataset [39] offer a flow extraction tool which they use to produce their own statistical analysis on the dataset but the development of this tool is flawed and does not produce complete flows for pcap data [17, 24]. Gints [9] offers an improved version of the flow meter which fixes the shortcomings of the original flow meter. While there are tools purposefully made to extract statistics related to CICIDS2017, in order to provide a complete post-mortem analysis of traffic flows we chose to use NFStream [2]. NFStream is a flexible network data analysis framework allowing us to not only retrieve network flows without improper flow drop but also provide a full statistical analysis of each traffic flow allowing for more in-depth comparison.

We attempt to compare the flows between the 2 classes of traffic generation; one which is representative of "real-world" traffic and one which wishes to emulate "real-world" traffic generated by our Docker scenarios. For simplicity of our comparisons, we separate all malicious traffic into their individual classes to perform similarity measures in each traffic class individually.

For simplicity, we separate traffic created from our scenarios into 2 classes of network traffic: benign traffic and malicious traffic. Within these classes, individual flows are given individual labels in reference to which traffic interaction they are relevant to. For example, TCP traffic can be labelled as both benign FTP traffic as well as malicious FTP Patator traffic.

To generate our deterministic dataset, we set up containerised versions of the CICIDS2017 attack traffic locally. We then run a series of scripted scenarios to simulate attack scenarios between attackers and victims on an artificial network. These interactions consisted of multiple formats of DoS/DDoS attacks, brute force, web attacks and data exfiltration. We collect all data from both the attackers' and victims' perspectives, but specifically for this experiment, the victim traffic (server-side) will be used for traffic comparison. These artificially generated pcap files make up our instances of "malicious traffic".

4.2.3 Methodology

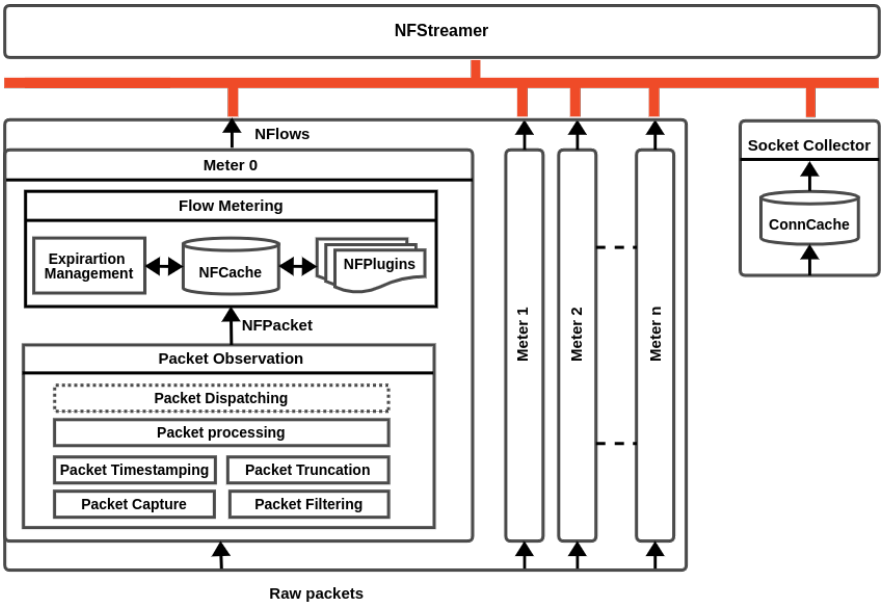

Figure 4.1: NFStream architecture to produce network flows from raw packets [2]

To measure the similarity between the Docker traffic production framework and the CICIDS2017 dataset, we perform the following steps.

1. Manually filter raw packet data from the CICIDS2017 Dataset and our Docker-generated scenarios into reduced pcap files using Wireshark. Each pcap file should only contain traffic relevant to a single attack class. Filtering of the CICIDS2017 pcap data is aided, as timestamp ranges and IP addresses are provided for each attack over the working hours period.

2. To produce statistical data, we use NFStream to extract features in the pcap data from both the original dataset and our generated malicious traffic. NFStream allows us to supply raw pcap data post-mortem to produce network analysis of the traffic interactions. For this experiment, we choose to extract the IAT (inter-packet arrival times) of each traffic flow to produce our metrics, which, as previously mentioned, are relevant to the detection of malicious traffic.

3. Using our 2 IAT vectors from both datasets, we calculate the normalised mutual information between our two vectors to perform our similarity measure using random samples at 10% of the total statistical traffic data for each specific traffic label. We use mutual information to quantify the "amount of information" obtained about one random variable by observing the other random variable. This allows us to compare two pcap vectors without ordering specifically affecting the representation of similarity. This means that due to the random nature of network traffic, taking the normalised mutual information between the CICIDS2017 traffic and any iteration of the same traffic scenario will produce the same similarity metric on every instance (provided scripted variables remain constant).

4. Using this similarity metric, fine-tune our attack parameters to further align our deterministic traffic generation with the baseline model.

4.3 Experiment 2 – Coverage and Comparability

4.3.1 Motivation

In order to evaluate the reputability of the traffic generated between the deterministic containers and the original CICIDS2017 dataset, we want to ensure we produce similar coverage in data volumes, data proportions and data types. To accurately replicate the dataset in a deterministic setup, mirroring these features as closely as possible will allow closer to a 1:1 dataset mirror while demonstrating the power of container environments to alter data volume and data properties. Specifically, we wish to mirror subsets of the total traffic given in the CICIDS2017 dataset. To do this, we can use Wireshark [47] to perform traffic analysis. Wireshark provides a wide range of features that are useful for analysing and comparing pcap files, such as packet filtering, packet decoding, and packet inspection. By using these features, we will be able to gain a better understanding of the differences between the two pcap files and draw meaningful conclusions about the effectiveness of network monitoring tools and techniques. Through the use of Docker environments, we can produce pcap traffic to which we can then perform comparisons on the original dataset as well as highlight the malleability of the Docker tools.

4.3.2 Data Preprocessing

Our goal is to measure the similarity between specific traffic instances across the CICIDS2017 dataset. We choose again to separate the original dataset into its specific malicious attack labels. For this experiment, we use our Docker containers to generate traffic for 9 specific malicious traffic labels: SSH-Patator, FTP-Patator, SQL-injection, Heartbleed, DoS Hulk, DoS Slowloris, DoS Slowhttptest, DoS LOIC and DoS Goldeneye. We can then perform individual comparisons between the CICIDS2017 dataset traffic and the Docker-generated traffic. We then extract the relevant traffic from the original traffic into individual pcap files to perform the comparison.

4.3.3 Methodology

To explore the protocol hierarchy between the CICIDS2017 dataset and our Docker-generated scenarios, we perform the following steps.

1. Use Wireshark's filtering capabilities to export the pcap data only relevant to the IP addresses and protocols of the specified attack labels. This ensures the original dataset traffic capture subsets only contain that of the relevant traffic.
2. Generate protocol statistics from each pcap file using Wireshark and provide a comparison between each capture.
3. Extract traffic protocol ratios from our Docker scenarios and the CICIDS2017 dataset for each individual attack class. We choose to measure protocol ratio for

traffic comparison as we cannot exactly emulate the exact same traffic volume as the CICIDS2017 dataset.

4.4 Experiment 3 - Varying Latency and Effects on IATs

4.4.1 Motivation

In sections 5.2.1 and 5.2.2, we present metrics that illustrate the similarity between traffic generated by Docker and the baseline CICIDS2017 dataset. To enhance the container model's ability to accurately replicate the precise configuration of the CICIDS2017 attacks, we investigate the impact of varying container processing speeds on inter-arrival times (IATs). In this context, we draw on the work of Paxson et al [34] and Bolot et al [3], who have demonstrated how packet statistics can be directly affected by network topology and processing speeds on end-to-end network dynamics. By making use of the ability to directly change packet flow in a container environment, we aim to modify our current traffic generation approach and improve its capability to emulate the CICIDS2017 dataset.

4.4.2 Methodology

To best simulate the original attack environments produced by the CICIDS2017 dataset, we wish to better emulate the similarity between the packet data produced by our Docker environment and the baseline dataset. To improve the similarity of the traffic data between datasets, we must have direct control over processing power and network latency as far as possible. To explore the possibilities of increasing the determinism of our traffic generation, we use forced latency over Docker networks. By supplying our Docker-Compose files with 2 separate networks to alias our containers with, we can set up a virtual communication "network" and enforce latency within the transmission of packets to and from the network. Using this tooling, we can then choose to enforce latency on packets within any unit of time. Specifically for this experiment, we choose to enforce latency on the SSH-Patator capture setup. This choice was made as not only has SSH-Patator previously demonstrated subpar similarity metrics but also, unlike other scenarios with subpar similarity metrics (such as FTP-patator), it contains traffic of only 1 protocol, in this case, SSH.

```
networks:
    network:
        driver: bridge
        ipam:
            config:
                - subnet: 10.5.0.0/16
                  gateway: 10.5.0.1

    Npatator:
        driver: bridge
        ipam:
            driver: default
            config:
                - subnet: 10.9.0.0/16
                  gateway: 10.9.0.1
        driver_opts:
            com.docker.network.tc_latecy: "10ns"
```

Figure 4.2: Docker networking setup for SSH-Patator

To explore the effect of forcing determinism on IAT's we perform the following steps:

1. Launch a DigitalOcean [7] droplet to spin up a virtual machine to perform latency testing on. DigitalOcean allows us to set up virtual machines with specific setup configurations to vary the number of CPUs. we launched a droplet with 2 CPU allowing for a reliable 4 GiB Memory, 80 GiB Storage and 4,000 GiB Bandwidth. This setup configuration ensures that we can successfully simulate the processing speeds similar to the CICIDS2017 dataset when generating SSH-Patator network traffic (Kali Linux/Ubuntu web server) using our container capture.

2. Within our capture setup, we keep all variables fixed except the invoked *tc-latency* to track the variability of IAT's across a range of latency variables. Using *tc-latency* within our Docker networking allows us to leverage the linux kernels' "traffic control" capabilities [13] to force specific latency on incoming and outgoing packets. we test latency periods linearly interpolated from $1 \cdot 10^{-3} ns$ to $1 \cdot 10^{6} ns$ to test a wide variety of possible congestion times.

3. Manually use Wireshark filtering capabilities to ensure the generated traffic from the containers is only relevant to the Patator attack tool. This subsequently removes any unwanted packets such as ARP or UDP.

4. Extract the "bidirectional mean packet inter-arrival time" from the baseline CICIDS2017 dataset and our deterministic container environments for SSH-Patator flows

5. Measure the cosine similarity between the vectors generated by NFStream for each enforced latency capture period on the baseline SSH-Patator data.

4.5 Experiment 4 – Flow Classification

4.5.1 Motivation

Having investigated our Docker container scenario's ability to generate and replicate data from the CICIDS2017 dataset, we explore the quality comparison of classifying flows using the container generation and baseline dataset. The CICIDS2017 dataset offers a variety of malicious traffic from a range of attack types, we aim to take a subset of these attacks which we can emulate in a container scenario and perform flow classification on post-mortem packet data to analyse the quality and ability to classify attack traffic based on packet statistics. Using NFStream, we can extract meaningful packet statistics from pcap data to perform packet inspection and ultimately extract statistics for IAT-based ML classification. However, while IATs can be used to classify traffic, it may not be effective in all scenarios because it relies heavily on the analysis of specific network characteristics. In addition, the accuracy of the classification algorithm will be affected by factors such as network topology, traffic volume and latency.

Quality traffic classification is paramount for an IDS, as it handles the detection of traffic categories when traffic patterns vary unexpectedly or uses atypical ports for the network process. This variation of typical patterns is particularly important to allow for the detection of malicious traffic, in which network traffic will purposely be manipulated into unusual patterns or timing intervals to avoid detection. With this in mind IATs have been shown to be an efficient measure for classifying traffic flows. Jaber et al. [19] highlights the use of IATs for classification purposes, with precision exceeding 90% for most protocols. While these results are promising, Jabel et al. did not specifically cover the detection of malicious traffic and only performed an evaluation of their classifier on the dataset it was trained on, giving little evidence on how IAT-based classification would perform on an unseen dataset with variations in the network traffic. Pizkac et al. [35] also explores the issues surrounding using IAT's for flow classification, with the work mainly highlighting the effects of natural network jitter on classification quality.

We attempt to use our Docker container environment and the baseline CICIDS2017 dataset to perform classification on a range of malicious traffic using supervised learning. For this classification experiment, we will supply traffic labels by hand on our deterministic traffic scenarios, as well as hand label the given pcap data from the CICIDS2017 dataset. Moreover, we will explore the models' ability to generalise across datasets by classifying a combination of baseline and container-generated data.

4.5.2 Methodology

We wish to elicit a trained classification model with the ability to classify application flows and generalise across datasets. We outline the experiment methodology through the following steps:

1. Create a training dataset using the traffic generated from our container environments. We limit our network traffic to 9 malicious traffic categories (FTP-Patator, SSH-Patator, DoS Hulk, DoS Goldeneye, DoS Slowhttptest, DoS LOIC, DoS

Slowloris, Heartbleed and SQL injection)

2. Produce multiple iterations of each traffic generation scenario with a forced network latency following a random distribution to emulate real traffic conditions outside of a kernel environment using Docker-Networking's *tc-latency* tool
3. Pre-process the generated pcap data using *Editcap* [41] to ensure that only relevant traffic generation protocols are used to train the classifier
4. Perform manual filtering on the CICIDS2017 pcap data and produce all flows using NFStream to avoid the prevalent traffic mislabelling issues when using the CICflowmeter as highlighted by Gints et al [26]
5. Two classification datasets of equal length and class hierarchy were created with the aim of ensuring that the resulting models were comparably balanced. Dataset 1 contains data only from the CICIDS2017 dataset and Dataset 2 contains data from only our Docker generate scenarios. Dataset features are also limited to packet and IAT variables (e.g. packet size, bidirectional-IAT mean, bidirectional-IAT standard deviation).
6. Train a random forest classifier with a max depth of 20 to classify traffic flows. In Dataset 1 we wish to clarify a baseline classification quality by training and testing only the CICIDS2017 dataset. In Dataset 2 we wish to test our Docker generated traffics' ability to generalise across datasets by training on our Docker dataset and testing on unseen CICIDS2017 test data.
7. Produce classification reports to compare the classification quality between Dataset 1 and Dataset 2 when training random states are fixed and the datasets are as rigid as possible

Chapter 5

Evaluation and Results

The primary objective of this dissertation was to produce deterministic traffic scenarios capable of replicating the malicious traffic produced in the CICIDS2017 dataset using a variety of Docker containers. Moreover, we wished to produce this traffic with comparable qualities aligned with the CICIDS2017 dataset for intrusion detection purposes.

5.1 Docker scenarios

Over the course of this project we generated scripted scenarios for 9/18 malicious traffic instances presented in the CICIDS2017 dataset, we evaluate our container scenarios on the traffic requirements discussed in section 3.3 in Table 5.1.

Table 5.1: Docker scenario quality

Name	Variations	Complete Capture	Total Interaction	Labelled
FTP-Patator	1	Yes	Yes	Yes
DoS GoldenEye	N	Yes	Yes	Yes
Heartbleed	1	Yes	Yes	Yes
DoS HULK	N	Yes	Yes	Yes
DoS LOIC	N	Yes	Yes	Yes
DoS SlowHTTPtest	N	Yes	Yes	Yes
DoS Slowloris	N	Yes	Yes	Yes
SQL-Injection	1	Yes	Yes	Yes
SSH-Patator	N	Yes	Yes	Yes

"N" represents an infinite number of variations on scenarios as runtime variables allow for infinite configurations

Data Volume Within our scenario generation of traffic data, we can produce an unbounded amount of traffic volume upon launch and completion of Docker scenarios. While the amount of traffic volume can be produced on a large scale, the container setup also allows for runtime variables to be set to limit the total volume of traffic produced.

For example, Patator tools will only run as long as the user/password list is provided to the container at runtime.

Traffic Variability Throughout the tooling generation of our scripted scenarios we can produce runtime variability to our scripted scenarios to allow for differences in outputted traffic such as packet latency, traffic timing intervals and the number of threads. Developing various runtime options allows all our scenarios to have variability where possible to still elicit natural traffic flow. Between the production of varied traffic through the Docker tooling and the use of *Editcap*, we can artificially manipulate the traffic balance within pcap data to better represent a range of traffic categories across a dataset.

Traffic choice Our implemented malicious capture scenarios cover a range of potential attacks. While all these attacks exist in the original CICIDS2017 dataset, we create capture scripts that allow users to invoke a variety of setup options to change the traffic production in further iterations. Our produced scenarios span a variety of attack classes including DoS, DDoS, brute-forcing, web attacks and data exfiltration. In total, we developed Docker capture scenarios to produce traffic data for 9 attack classes.

Determinism To highlight our scenario configuration's ability to produce deterministic traffic; we construct a small experiment on packet data. We generate 20 pcap files of the FTP-Patator scenario applying a fixed level of network delay with network and processing conditions fixed to minimise variations in traffic due to processes outside the Docker virtual network. We then calculate the mean IATs and packet size for packets 1-12 across all 20 pcap files to get a consistent measure to compare traffic statistics. The results of this experiment can be seen in Figure 5.1/5.2. As with the nature of the FTP-Patator traffic, we will naturally see greater periods of variation in packet size due to the combination of FTP and TCP protocols prevalent in the tooling. As well as we can see that a significant amount of our packets only vary minimally in timing. Due to these properties, we verify that our scenarios are deterministic up to differences in processing speed and network configurations.

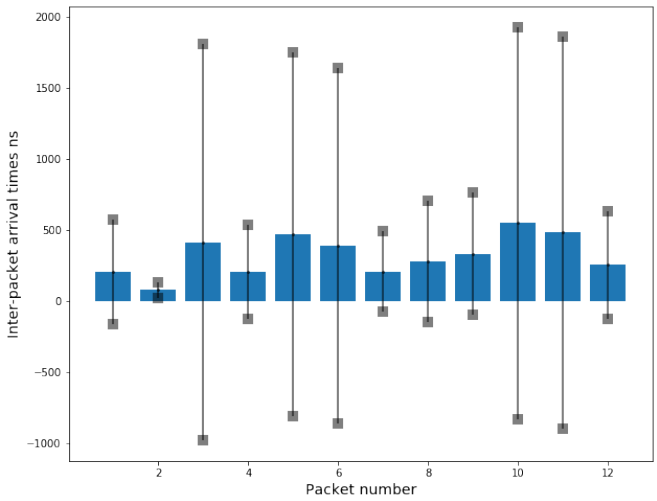

Figure 5.1: Means of IATs across first 12 packets

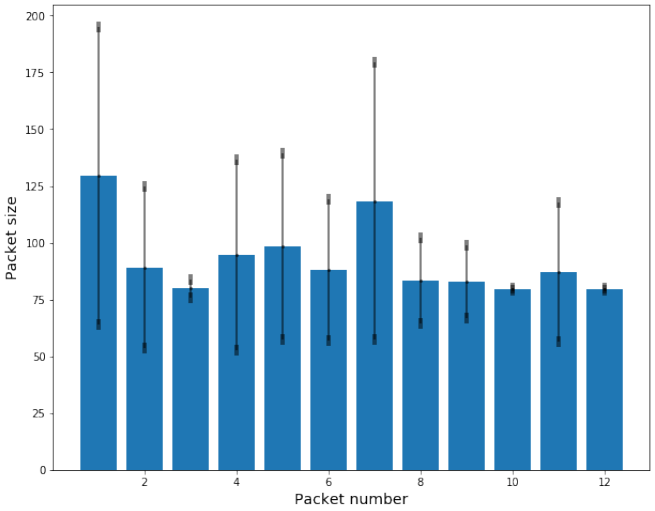

Figure 5.2: Means of packet size across first 12 packets

Scenario capture Upon running the relevant scripts to start our capture scenarios, all traffic capture is completed automatically. All scenarios start up relevant containers in specifically invoked orderings and capture all traffic until completion. Due to the configuration of the injected *tcpdump* containers within our scenarios, users can interact with the capture via the terminal during runtime to kill capture processes at any time while maintaining all previous packet capture.

5.2 Experiment Results

5.2.1 Results of Experiment 1

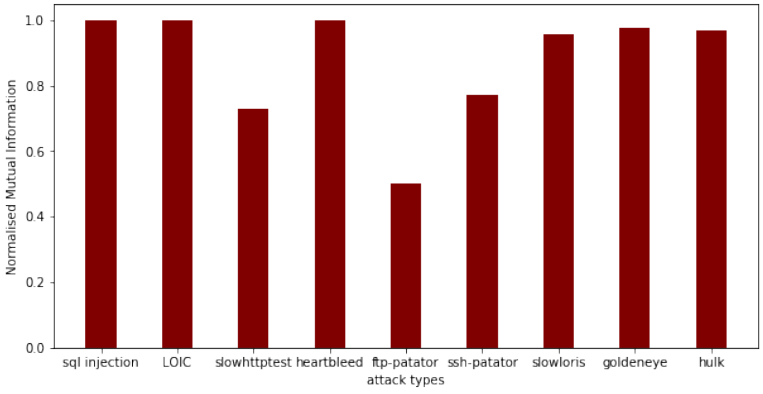

Figure 5.3: Normalised mutual information between original CICIDS2017 dataset and container generated attack traffic

As seen from the graph in Figure 5.4, a wide range of similarity quality was produced across the containerized attacks. In this case, comparing inter-packet arrival times was used as the feature vector, which highlights strong similarity performance metrics

overall and some minor outliers, highlighting a few issues in regenerating the original dataset.

While all protocols match in the traffic execution, IATs are primarily affected by both network topology and processing speed. It is key to note here that the malicious traffic tools that allowed users to supply specific artificial delays produce much better traffic similarity than such attacks which were executed at the maximum processing speed. For example, DoS Slowloris allows users to define intervals between packet floods, meaning that more variables are accessible to the user to generate similar traffic. Whereas, the execution of the Patator tool is only limited by the host processing speed and network topology, without means to directly emulate the tool on the original host machine defined by the dataset it is increasingly difficult to emulate the packet flow speeds in tool execution regardless of protocols exactly matching. This can be seen in the FTP-Patator similarity measure, where we see a low similarity in IATs. We can attribute this low similarity to the way we enforce latency in our scenarios. As we enforce latency to improve the similarity in favour of the scenarios' FTP traffic, it negatively skews the similarity of the TCP traffic in the scenario.

5.2.2 Results of Experiment 2

Table 5.2: Docker vs CICIDS2017 dataset Protocol Hierarchy

Dataset	Docker					CICIDS2017				
Attack Type	HTTP	TCP	SSH	TLS	FTP	HTTP	TCP	SSH	TLS	FTP
DoS Goldeneye	0%	100%	0%	0%	0%	0%	100%	0%	0%	0%
DoS LOIC	0%	100%	0%	0%	0%	0%	100%	0%	0%	0%
DoS HULK	0%	73%	24%	0%	3%	0%	76%	18%	0%	6%
DoS Slowloris	6%	94%	0%	0%	0%	7%	93%	0%	0%	0%
DoS Slowhttptest	1%	99%	0%	0%	0%	1%	99%	0%	0%	0%
SSH-Patator	0%	0%	100%	0%	0%	0%	0%	100%	0%	0%
FTP-Patator	0%	48%	0%	0%	52%	0%	40%	0%	0%	60%
Heartbleed	0%	71%	0%	0%	29%	0%	75%	0%	25%	0%
SQL-Injection	0%	100%	0%	0%	0%	0%	100%	0%	0%	0%

From the use of Wireshark filtering, we were able to extract the relevant packets from the pcap files to perform analysis on. We show protocol coverage in Table 5.2 to present the differences between total traffic coverage. We see that while in the previous experiment, our IATs do not always produce similar statistical vectors, the Docker-generated traffic uses the correct tooling in order to produce the correct protocol coverage. We see some protocol discrepancies in the malicious traffic of DoS Hulk, Heartbleed, FTP-Patator and DoS Slowloris. These small differences can be attributed to minor setup differences and server setup differences. As we cannot accurately determine the full victim server setup for these malicious attacks, our Docker containers aim to match the server as closely as possible. While our results have a high quality of success, without full server configurations, we cannot expect exact coverage across all attack protocols.

5.2.3 Results of Experiment 3

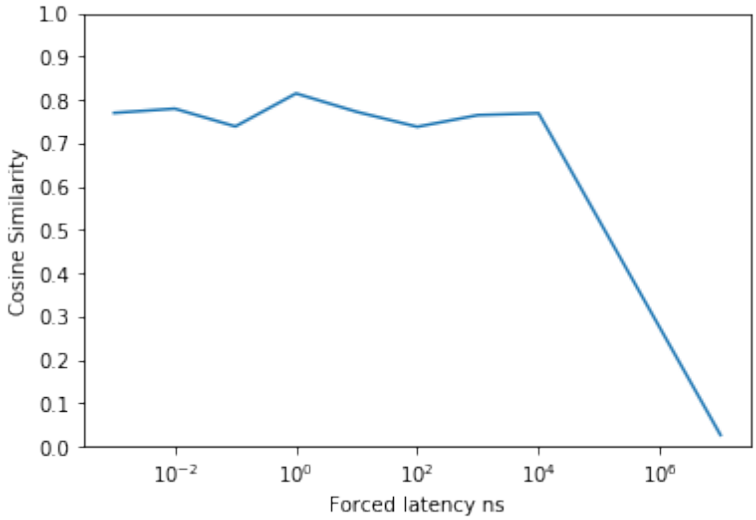

Figure 5.4: Docker networking setup for SSH-Patator

After completing a scenario with a unique forced latency variable, we compare the similarity of the relevant CICIDS2017 traffic to our generated traffic. From Figure 5.5, we can see some variability in similarity quality with extremely low enforced latency. In this test case, the similarity to the CICIDS2017 dataset peaks at 1ns. While we see a minor peak in similarity performance at 1ns, latency variations show minor effects on cosine similarity across low latency periods from $1 \cdot 10^{-3} ns$ to $1 \cdot 10^{4} ns$. This similarity then tapers off rapidly as we move closer to 1s of latency per packet. While the IATs of tools will vary on the baseline dataset as attack tools were performed across similar machines, it is important to note the rapid decline in IATs similarity as we increase/decrease network latency to align with the original dataset. We can see that the similarity measure of the pcap traffic can be directly correlated to the variability of the network latency.

5.2.4 Results of Experiment 4

Table 5.3: Baseline CICIDS2017 dataset classification performance (Dataset 1)

Attack	Precision	Recall	F1-score
FTP-Patator	0.61	0.72	0.66
DoS GoldenEye	0.67	0.47	0.55
Heartbleed	0.54	0.18	0.27
DoS HULK	0.42	0.53	0.47
DoS LOIC	1.0	0.50	0.67
DoS SlowHTTPtest	0.97	0.97	0.97
DoS Slowloris	1.0	0.99	0.99
SQL-Injection	1.0	1.0	1.0
SSH-Patator	0.58	0.48	0.52
Weighted-Average	0.67	0.67	0.67

Table 5.4: Docker dataset classification performance (Dataset 2)

Attack	Precision	Recall	F1-score
FTP-Patator	0.80	0.85	0.82
DoS GoldenEye	0.82	0.82	0.82
Heartbleed	0.90	0.79	0.84
DoS HULK	0.93	0.83	0.90
DoS LOIC	1.0	0.75	0.86
DoS SlowHTTPtest	0.99	0.98	0.99
DoS Slowloris	1.0	1.0	1.0
SQL-Injection	1.0	0.5	0.67
SSH-Patator	0.81	0.75	0.78
Weighted-Average	0.84	0.84	0.84

In Table 5.3/5.4 we can see a total breakdown for the classification of each individual traffic label across both datasets covering precision, recall and their combined metric; F1-score. In the Docker dataset, we see an overall F1-score performance of 0.84 which is a 17% improvement to the baseline dataset eliciting a significant improvement in classifying unseen CICIDS2017 data. We can see that the Docker dataset outperforms the baseline dataset on classification performance on 8/9 traffic labels with SQL injection showing a much higher classification rate when trained on only the baseline data where the Docker-trained classifier produces a 50% lower recall rate. We also see some similar performance metrics across the 2 classifiers, with DoS SlowHTTPtest and DoS Slowloris producing almost identical performance upon classifications. In contrast, the 2 classification models produce drastically varying performance when classifying “Heartbleed” traffic flows where the Docker-generated classifier outperforms the baseline classifier by 57%.

5.3 Discussion of Experiments

Experiment 1 This experiment analysed the similarity between the CICIDS2017 malicious traffic and traffic created by containerised environments. By extracting the necessary traffic using traffic filtering in *Wireshark* we extracted the relevant data for cosine similarity comparison between the 2 traffic sets. By analysing the similarity results and delving into the tool setup of each malicious traffic tool, we can highlight that we can emulate malicious traffic tools much more accurately when we can provide artificial delays in the execution, allowing us to bypass setup issues surrounding network topology and processing speed. While the containers are lightweight and execute efficiently, the packet timings are not feasible to emulate if we are limited by processing and network usage. Without being able to produce containers in a full Docker-Compose network that imitates both the hardware and network usage, tools that are not timing-limited will prove difficult to accurately emulate their IAT's.

Experiment 2 This experiment analysed the protocol coverage and similarity between the Docker environment-generated malicious traffic with that of the original CICIDS2017 dataset. By extracting the specific traffic from the original dataset using Wireshark filtering, we analysed the protocol hierarchy statistics between the two datasets. By analysing the protocol coverage, we can elicit that we accurately represent similar protocol coverage when emulating the malicious tool used from the original dataset. It is important to note that while the protocol hierarchy of the traffic produced by our Docker tooling is similar, we are not able to exactly match the protocol distribution across all traffic labels. We highlight in section 5.2.1 that without emulating the processing speeds and network topology of the original dataset we cannot truly exactly replicate the traffic communications given by the original dataset.

Experiment 3 This experiment analysed the effect of forced network latency on IATs within our scripted Docker scenarios. By enforcing strict packet delays on inbound and outbound traffic, we can affect specific container networking conditions within a scenario. After applying Wireshark filtering, we analysed how IATs varied with network latency. Furthermore, by analysing the similarity results across the latency periods, we can elicit that variation in network latency does have a direct effect on IATs of the output attack traffic. This means that in order to correctly emulate all malicious traffic in the baseline dataset, we need to perform empirical analysis to find the optimal latency depending on the host machine setup before producing traffic. Figure 5.4 also highlights that latency periods remain very similar when increasing across a very small scale but show a rapid decline in quality as packet latency trends towards 1s of latency across packet transmission. This rapid decline in similarity highlights some key issues in generating traffic within the Docker network. Without enforcing latency correctly alongside the processing speed supplied by the capture scenario, the traffic capture similarity can vary wildly.

Experiment 4 The two tables present the classification performance of various cyberattacks in two datasets: the scripted scenario Docker dataset and the baseline CICIDS2017 dataset. The intention of designing these datasets to be as similar as possible was to isolate the impact of specific modifications made to the combined Docker dataset and

to ensure a fair comparison between the two datasets.

Although the two datasets share a similar structure and evaluation methodology, there are some differences in the attack distribution and performance measures between them. For instance, the Docker dataset demonstrates better classification performance across 8/9 attack labels compared to the baseline CICIDS2017 dataset. However, there are some exceptions to this, such as the "SQL injection" attack, where the baseline dataset achieves a significantly higher F1-score score. While this is an outlier, our Docker dataset performs significantly better at classifying unseen flows than the CICIDS2017 dataset. Due to this improved performance, we can ascertain that our Docker-generated data has the ability to generalise classifications across datasets.

It is important to note that these differences may affect the accuracy and generalizability of any models trained on either of these datasets. However, given that the datasets were designed to be as similar as possible, any observed differences could be attributed to the specific modifications made to the Docker dataset. As such, both datasets could still be valuable in evaluating the performance of different models in detecting cyber-attacks, provided that the differences between the datasets are properly accounted for in the analysis. In order to specifically model our containers to match other datasets with similar attack traffic, we can use our designed latency modifications to elicit new metrics for our packet variables used to train our classifiers.

5.4 Difficulties and Limitations

A large amount of development time is required to address the complexity of developing scenarios using the Docker framework. While Docker containers offer a very lightweight package, they are also very restrictive in setup. In order to get each container to a working state that can resemble real-world traffic while maintaining the possibility of invoking runtime variables to change the output traffic, a considerable amount of scripting and configuration is necessary. While many services that are commonly used across Docker frameworks are maintained by official developers, due to the malicious traffic component of this study, many of the obscure applications required to run a malicious traffic simulation are not maintained and are usually deprecated. Furthermore, due to the complex nature of attack tooling and the desire to avoid ease of access to users without a depth of knowledge, many attack tools are void of any in-depth documentation. These setup problems are further compounded by the atypical nature in which we choose to set up communication between containers to allow for enforced latency.

For example, many issues arose when developing the DoS LOIC scenario when developing an image for the LOIC attack tool. The original tool was developed by Praetox's "LOIC project" which was designed to be as "script kiddie" friendly as possible and is heavily wrapped into a GUI setup. Due to the nature of our container use in a scripted scenario, it is not possible to use a GUI format. To create a functioning container, we decided to relax the design criteria of our image choice to allow the use of images not linked with the original project code. we choose to use undocumented source code from GitHub in which we manually check the authenticity of by cross-referencing the C

code with the original project and comparing the quality of its traffic production. This ensures we use tooling as similar as possible to the original Praetox project.

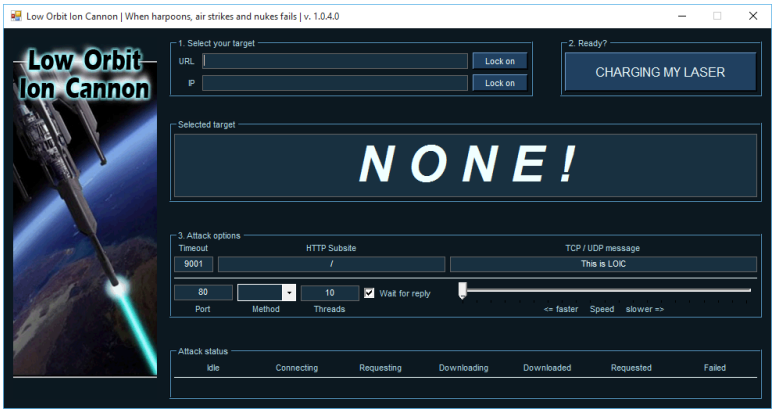

Figure 5.5: DoS LOIC GUI

In addition to image choice problems, setting up enforced latency was surrounded by a variety of issues. In previous implementations of scenarios within the Detgen project [5] latency was scripted using *tc-netem* which raised some minor issues with delay in the start-up of containers and delays in enforcing packet delays. While it was possible to continue to use this setup, we would have to continuously drop many of the early occurring packets when an artificial delay was invoked as no packet latency would be applied to them as such. To solve this issue, in-depth research into the configuration of Docker networking was required. The final solution required all containers to be set up on their own individual network. The start-up order was then based on tool dependencies upon other containers within the network. Using this networking setup, we could individually apply Docker networking's *tc-latency* tool to each container to allow for detailed latency configurations across a scenario (see Figure 4.2).

Finally, our main limitation when attempting to emulate the CICIDS2017 dataset is the lack of detail supplied by the authors on setup and configuration. Without more detailed information on the attack tool configuration and setup of the network to such fine details, we have to perform our emulation at an "informed" effort in an attempt to produce a dataset as similar as possible. For example, we cannot accurately determine any of the runtime variables supplied to any DoS attack tool from detailed packet inspection, so producing pcap and flow data to match the similarity of the baseline dataset is done purely through time-consuming experimentation. This also brings up the issue of processing power to run the containers. To accurately emulate the dataset, our container environments also need to match both the network and processing speed of the traffic communication when executed in the baseline dataset. As we cannot ascertain these configurations from the scientific literature on the dataset or pcap inspection, we choose to launch DigitalOcean droplets [7] with a redundant amount of processing power and apply latency to each scenario on a case-by-case basis to best match the packet data between our container scenarios and the original data from CICIDS2017.

Chapter 6

Conclusions

In order to protect both networks and users from malicious traffic, the development of IDSes is paramount to battling the rapid expansion of vulnerabilities that grow with the expansion of the devices available across world networks. However, we presently lack a variety of expandable datasets with accurate ground truths to train an IDS based on machine-learning anomaly detection. The primary goal of this project was to create a Docker framework capable of generating network traffic that was representative of the CICIDS2017 dataset while maintaining the capability to generate worthwhile network traffic. To do this, we designed and implemented 9 scenario cases with multiple scriptable variations to produce sub-scenarios, all of which are capable of producing malicious network traces in line with the baseline CICIDS2017 dataset. While we achieve the primary goal of this project, in doing so, we also create Docker scenarios for the DetGen project to extend the tool's ability to generate traffic for a set of prominent malicious attack patterns in the current network threat landscape.

We also demonstrated it is possible to implement and introduce artificial delays to individual containers within the Docker framework such that it can resemble a realistic wide-area network, such as the artificially generated network in the CICIDS2017 dataset. Following this, we performed machine learning on both the baseline dataset and the Docker dataset using our scripted scenarios to demonstrate that using data generated from our Docker containers can elicit significantly improved performance and characteristics compared to the CICIDS2017 dataset. Our work to extend the DetGen framework expands the possibilities to produce artificial traffic but also represents the capabilities of the generated traffic to produce quality NIDS datasets. Furthermore, our ability to replicate previously generated traffic and generate completely unique pcap data on individual scenarios shows the wide range of traffic variability we can produce.

6.1 Criticisms

Our Container scenarios are capable of producing data to produce malicious traffic for a large network intrusion dataset, but we do not perform any testing or experimentation outside small subsets of this dataset. Due to processing limitations, using this large dataset and containers to produce large amounts of pcap data was not possible within

the timeframe of this project. Instead, our ability to finitely control traffic and produce quality classification serves as an example of what our Docker scenarios are capable of. In addition to traffic volume limitations, the framework would benefit from increased variability. In our experimentation, we implement 9 scenarios into the Docker traffic generation framework to represent our traffic in traffic classification. While we produce highly successful classification results, our models would benefit from both an increased traffic variety and traffic volume.

To further improve the granularity of our Container framework, implementing solutions to limit scenario capture based on both time or number of packet captures would enable us to have further control of our capture containers. Implementing these optional limitations would allow us to better evaluate and experiment the possibilities of exactly matching traffic generation on every iteration of a scenario. This implementation would increase our ability to claim exact determinism across packet captures in our synthetic traffic.

6.2 Future Work

As our design requirements dictate our Docker framework to be expandable, we leave many avenues to explore in future work. A main priority would be to complete coverage of all 18 possible malicious scenarios within the CICIDS2017 dataset to allow for a complete comparison of generating traffic in a containerised environment compared to the network of virtual machines implemented by the Canadian Institute of Cybersecurity. Alternatively, the scenarios could be expanded to use more prevalent attack scenarios that have arisen since the production of the dataset in 2017. This would allow the dataset to better cover anomaly detection for prevalent attacks and tooling used currently by malicious users. Continuously updating the scenario capture codebase ensures the framework stays up to date with current data, which ensures a wider range of both benign scenarios and malicious scenarios to be implemented within the framework. Ultimately, supplying our models better “real world” data to train traffic classification models will improve our frameworks’ performance with more complex and diverse network data.

Bibliography

[1] Acunetix. Protecting against low orbit ion cannon. *Acunetix Blog — Web Security Zone*, 2010.

[2] Zied Aouini and Adrian Pekar. Nfstream: A flexible network data analysis framework. *Computer Networks*, 204:108719, 2022.

[3] Jean-Chrysotome Bolot. End-to-end packet delay and loss behavior in the internet. *SIGCOMM Comput. Commun. Rev.*, 23(4):289–298, oct 1993.

[4] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos. Feature selection and classification in multiple class datasets: An application to kdd cup 99 dataset. *Expert Systems with Applications*, 38(5):5947–5957, 2011.

[5] Henry Clausen, Robert Flood, and David Aspinall. Traffic generation using containerization for machine learning. *Proceedings of the 2019 Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security*, 2019.

[6] Gideon Creech and Jiankun Hu. Generation of a new ids test dataset: Time to retire the kdd collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4487–4492, 2013.

[7] DigitalOcean. The cloud for builders. https://www.digitalocean.com/, 2023. Last accessed 10/3/23.

[8] Docker. Docker documentation. https://docs.docker.com/, 2022. Last accessed 21/12/22.

[9] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 7–12. IEEE, 2021.

[10] Joshua Erickson. Ares. https://github.com/sweetsoftware/Ares, 2021.

[11] Nabila Farnaaz and Jabbar Akhil. Random forest modeling for network intrusion detection system. *Procedia Computer Science*, 89:213–217, 12 2016.

[12] Andrey Ferriyan, Achmad Husni Thamrin, Keiji Takeda, and Jun Murai. Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. *Applied Sciences*, 11(17), 2021.

[13] The Linux Foundation. Linux traffic control (tc). https://man7.org/linux/man-pages/man8/tc.8.html. Last accessed 12/3/23.

[14] Fyodor et al. Nmap. https://github.com/nmap/nmap, 2022.

[15] Dmitry Grafov. hulk. https://github.com/grafov/hulk, 2021.

[16] Arash Habibi Lashkari. Cicflowmeter-v4.0 (formerly known as iscxflowmeter) is a network traffic bi-flow generator and analyser for anomaly detection. https://github.com/iscx/cicflowmeter. 08 2018.

[17] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. Characterization of tor traffic using time based features. pages 253–262, 01 2017.

[18] Patrick Hulce. patator. https://github.com/lanjelot/patator, 2022.

[19] Mohamad Jaber, Roberto G. Cascella, and Chadi Barakat. Can we trust the inter-packet time for traffic classification? In *2011 IEEE International Conference on Communications (ICC)*, pages 1–5, 2011.

[20] Keysight Technologies. PerfectStorm. https://www.keysight.com/us/en/products/network-test/network-test-hardware/perfectstorm.html, Accessed: 2023-04-09.

[21] Vamplew Peter Kamruzzaman Joarder Khraisat Ansam, Gondal Iqbal. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):20, Jul 2019.

[22] Kurniabudi, Deris Stiawan, Darmawijoyo, Mohd Yazid Bin Idris, Alwi M. Bamhdi, and Rahmat Budiarto. Cicids-2017 dataset feature analysis with information gain for anomaly detection. *IEEE Access*, 8:132911–132921, 2020.

[23] Maxime Lanvin, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Eric Totel. Errors in the CICIDS2017 dataset and the significant differences in detection performances it makes, 2023.

[24] Arash Habibi Lashkari. Cicflowmeter. https://github.com/ahlashkari/CICFlowMeter, 2017. Last accessed 28/12/22.

[25] Laetitia Leichtnam, Eric Totel, Nicolas Prigent, and Ludovic Mé. Sec2graph: Network attack detection based on novelty detection on graph structured data. In Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 238–258, Cham, 2020. Springer International Publishing.

[26] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018. In *2022 IEEE Conference on Communications and Network Security (CNS)*, pages 254–262, 2022.

[27] Luis MartinGarcia. Tcpdump documentation. https://www.tcpdump.org/index.html, 2010. Last accessed 20/3/23.

[28] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.

[29] Z. Muda, W. Yassin, M. N. Sulaiman, and N. I. Udzir. Intrusion detection based on k-means clustering and naïve bayes classification. In *2011 7th International Conference on Information Technology in Asia*, pages 1–6, 2011.

[30] NewEraCracker. Loic. https://github.com/NewEraCracker/LOIC, 2019.

[31] Thuy Nguyen and Grenville Armitage. Grenville, a.: A survey of techniques for internet traffic classification using machine learning. ieee communications surveys tutorials 10(4), 56-76. *Communications Surveys Tutorials, IEEE*, 10:56 – 76, 12 2008.

[32] The Third International Conference on Knowledge Discovery and Data Mining (KDD Cup 1999). Kdd cup 1999 data. https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[33] Ranjit Panigrahi and Samarjeet Borah. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, 7(3.24):479–482, 2018.

[34] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[35] Pavel Piskac and Jiri Novotny. Using of time characteristics in data flow for traffic classification. pages 173–176, 06 2011.

[36] Scott Renfo. Mergecap manual page. *mergecap(1)*. Last accessed 21/1/23.

[37] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers Security*, 86:147–167, 2019.

[38] Jakob Seidl. Goldeneye. https://github.com/jseidl/GoldenEye, 2016.

[39] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. pages 108–116, 01 2018.

[40] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. A detailed analysis of the cicids2017 data set. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Information Systems Security and Privacy*, pages 172–188, Cham, 2019. Springer International Publishing.

[41] Richard Sharpe. Editcap(1) manual page. https://www.wireshark.org/docs/man-pages/editcap.html, 2023. Last accessed 21/12/23.

[42] Sergey Shekyan. slowhttptest. https://github.com/shekyan/slowhttptest, 2019.

[43] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers Security*, 31:357–374, 05 2012.

[44] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[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*, pages 1–6, 2009.

[46] Filippo Valsorda. Heartbleed. https://github.com/FiloSottile/Heartbleed, 2014.

[47] Wireshark. Wireshark manual page. https://www.wireshark.org/docs/, 2023. Last accessed 26/3/22.

[48] Robin Wood. Damn vulnerable web application (dvwa). https://github.com/digininja/DVWA, 2021.

[49] Gokberk Yaltirakli. Slowloris. *github.com*, 2015.

[50] Sebastian Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. pages 250– 257, 12 2005.

Appendix A

First appendix

A.1 LOIC Docker-Compose.yml

```
version: '3.9'
services:
  apache:
    image: httpd:latest
    container_name: my-apache-app
    ports:
    - '8080:80'
    volumes:
    - ./website:/usr/local/apache2/htdocs
    networks:
      network:
        ipv4_address: 10.8.0.5


  loic:
    image: perlloic:latest
    container_name: LOIC_working
    command: $HOST -$TYPE
    volumes:
      - $PWD:/mnt
    networks:
      network:
        ipv4_address: 10.8.0.6


  tcpdump1:
    image: kaazing/tcpdump
    network_mode: "service:apache"
    volumes:
      - ./tcpdump1:/tcpdump
```

```
  tcpdump2:
    image: kaazing/tcpdump
    network_mode: "service:loic"
    volumes:
      - ./tcpdump1:/tcpdump

networks:
    network:
      driver: bridge
      ipam:
        config:
          - subnet: 10.8.0.0/16
            gateway: 10.8.0.1
      driver_opts:
        com.docker.network.tc_latency: "100ms"
```

A.2 Basic Scenario Setup (LOIC)

```
#!/bin/bash

export HOST="$1"
export TYPE="$2"

# host == ip
# type == tcp or udp or http
# delay is in secs
docker compose -f loit.yaml up
```

A.3 Granular Scenario Setup (SlowHTTPTest)

```
#!/bin/bash

export URL="$1"
export CONNECTIONS="$2"
export LENGTH="$3"
export INTERVAL="$4"

docker compose -f slowhttptest.yaml up
```

A.4 Nomenclature

- **ML** Machine Learning
- **IP** Intern Protocol

- **IATs** Inter-Packet Arrival Times
- **UDP** User Datagram Protocol
- **SSH** Secure Shell
- **HTTP** Hypertext Transfer Protocol
- **TLS** Transport Layer Security
- **TCP** Transmission Control Protocol
- **FTP** File Transfer Protocol
- **ARP** Address Resolution Protocol
- **ICMP** Internet Control Message Protocol
- **SMPT** Simple Mail Transfer Protocol
- **CICIDS2017** Canadian Institute for Cybersecurity Intrusion Detection System 2017
- **IDS** Intrusion Detection System
- **NIDS** Network Intrusion Detection System
- **DoS** Denial of Service
- **DDoS** Distributed Denial of Service
- **XSS** Cross-Site Scripting
- **SQL** Structured Query Language
- **CPU** Central Processing Unit
- **GUI** Graphical User Interface
- **OS** Operating System