DetGen: Deterministic Ground Truth Traffic Generation using Docker for Machine Learning

Anonymous Author(s)

ABSTRACT

ACM Reference Format:
Anonymous Author(s). 2018. DetGen: Deterministic Ground Truth Traffic Generation using Docker for Machine Learning. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The exponential growth of data availibity enabled the machine learning revolution of this decade and transformed many areas of our lives. Ironically, researchers struggle to gather qualitative network traffic data to ... Well-designed datasets are such a rarity that researchers often evaluate intrusion detection systems on datasets that are well over a decade old [**?** **?**], calling into question their effectiveness on modern traffic and attacks. The lack of quantity, variability, meaningful labels, and ground truth has so far prohibited ML-based methods from having a bigger impact in network security.

Privacy and security concerns discourage network administrators to release rich and realistic datasets for the public. Network traffic produced by individuals contains a host of sensitive, personal information, such as passwords, email addresses, or usage habits, requiring researchers to expend time anonymising the dataset [**?**]. In order to examine malicious behaviour, researchers are often forced to build artificial datasets using isolated machines in a laboratory setting to avoid damaging operational devices. Background traffic is generated either from ...

The datasets currently available are meant to be all-purpose and are *static* in their design, unable to be modified or expanded. This proves to be a serious defect as the ecosystem of intrusions is continually evolving. Furthermore, it prohibits a more detailed analysis of specific areas of network traffic. To prevent this, new datasets must be periodically built from scratch.

Additionally, ground truth

Developing a framework that allows researchers to create datasets that circumvent these issues would be extremely beneficial. We propose that this can be done using Docker [**?**]. Docker is a service for developing and monitoring containers, also known as OS-level virtual machines. Each Docker container is highly specialised in its purpose, generating traffic related to only a single application process. Therefore, by scripting a variety of Docker-based *scenarios* that simulate benign or malicious behaviours and collecting the resultant traffic, we can build a dataset with perfect ground truth. Furthermore, these scenarios could be continually enhanced and expanded, allowing for the easy creation of datasets containing modern, up-to-date traffic and attacks.

This is the primary goal of this work. Furthermore, we demonstrate the utility of this framework by performing a series of experiments: one that measures the realism of the network traffic produced by our Docker scenarios and two that would be difficult to perform using a conventional dataset.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

2 DATA FORMATS AND EXISTING DATASETS

Computers in a network mostly communicate with each other by sending *network packets* to each other, which are split into the control information, also called packet header, and the user information, called payload. The payload of a packet in general carries the information on behalf of an application and can in be encrypted, while the header contains the necessary information for the correct transmission of the packet, including the transmission protocol layer, IP addresses, etc. Packet-level methods can be divided into payload inspection, header-based, or hybrid. Packets are usually stored in the widespread *pcap* format.

The majority of packets are exchanged between two hosts within bidirectional connections. Another common format of network traffic information is based on connection summaries, also called **network flows**. RFC 3697 [**?**] defines a network flow as a sequence of packets that share the same source and destination IP address, IP protocol, and for TCP and UDP connections the same source and destination port. A network flow is usually saved containing these informations along with additional information such as the start and duration of the connection as well as the number of packets and bytes transferred in the connection.

3 PROBLEMS IN EXISTING DATASETS AND TRAFFIC GENERATORS

A problem with all of these datasets is their static design.

Furthermore, many network traffic datasets are not comprehensively labelled. Even if only a single process is initiated by the user, VMs generate additional traffic from a variety of background processes, such as software querying servers to check for updates, resulting in aggregated application flows between processes. This necessitates the use of ground truth generation tools to classify flows. However, current methods of establishing the ground truth of network traffic datasets are well-known to be fallible[**?**]. These are port-based methods, which classify traffic flows based on their port numbers, and deep-packet inspection (DPI) based methods, which classify flows based on analysis of their packet payloads. Port-based methods are unreliable due to the dynamic allocation of ports, several services sharing the same port and processes running on atypical ports. Moreover, although DPI-based methods are

capable of inspecting the entirety of a packets contents, their performance has also shown to be lacking. Bujlow et al. [**?**] have shown that many commonly used DPI-based methods for ground-truth generation fail to classify packets, with some methods failing to classify common protocols such as HTTP and FTP with less than 10% accuracy. In contrast, it is trivial to produce a fully-labelled dataset from our Docker framework.

CIC-IDS 2017 [**?**], released by the Canadian Institute for Cybersecurity, is the primary dataset that we shall compare our results to. The dataset was created by monitoring the network activity of several virtual machines running a series of scripted scenarios. The majority of these virtual machines produced exclusively benign traffic whilst others were designated as attackers, producing malicious traffic. Moreover, the exploit scenarios contained within the dataset are moderately recent, including botnets, cross-site scripting attacks and SQL injections. Furthermore, the dataset is far larger than many similar datasets, consisting of five sub-datasets captured over the course of a working week. Capturing traffic over such a lengthy period of time allows for the temporal development of attack scenarios to take place over several days, more accurately mimicking an intruder's movement through the network. CIC-IDS 2017, however, does not address the problems discussed in this section.

4 REQUIREMENTS

The primary task of this project is to develop a suite of Docker containers capable of producing traffic suitable for training machine-learning-based intrusion detection systems. The containers are arranged in different configurations corresponding to particular *capture scenarios*. Running a given capture scenario triggers the creation of several Docker containers, each with a scripted task specific to that capture scenario. A simple exemplary capture scenario may consist of a containerised client pinging a containerised server. We ensure that each Docker container involed in producing or receiving traffic will be partnered with a `tcpdump` container, allowing us to collect the resulting network traffic from each container's perspective automatically. We wish to publish this framework to a wider audience, allowing for further modification. To achieve this goal, we introduce the following key design principles:

(1) To ensure that we produce representative data for modelling, we want the traffic generated by our container suite to consist of a good number of protocols that are commonly found in real-world traffic and existing datasets. For malicious traffic, we want to ensure that the attacks are modern and varied, both in purpose and in network footprint.
(2) For each protocol, we want to establish several capture scenarios to encompass the breadth of that protocol's possible network traces. For instance, if we consider a capture scenario consisting of a client downloading various webpages over SSL, it is not enough to only generate traffic from successful connections. We must also include several scenarios where the client fails to download the aforementioned webpage because of a misspelled address or missing certificate. Whenever possible, we also want to capture WAN traffic.
(3) Traffic capture scenarios should be implemented in a modular way to allow for a straightforward addition or modification of traffic capture modules.
(4) Since ground truth a main focus of this work, we want the capture scenarios to be, on some level, deterministic. This way, we can relate individual traffic events to the computational procedure responsible for its generation. We discuss what it means for a scenario to be deterministic in section **??**.
(5) Communication between containers should be subject to the same disturbances and delays as in a real-world setting.

5 VIRTUALISATION AND DOCKER

Virtualisation refers to the act of creating isolated operating systems, known virtual machines (VMs), that share the same hardware infrastructure — known as the host machine. OS-level virtualisation, also known as *containerisation*, is a virtualisation paradigm that has become increasingly popular in recent years due to its lightweight nature and speed of deployment. Containers forego a hypervisor and the shared resources are instead kernel artifacts. Although this prevents the host environment from running different operating systems — for instance, a Linux host can only run Linux containers — containerisation incurs minimal CPU, memory, and networking overhead whilst maintaining a great deal of isolation [**?**]. The high-level of isolation between both the host OS and any running containers is of paramount importance for our framework as it allows us to run malicious software that could potentially compromise the security of the system it is running on with negligible worry [**?**].

Docker Images. In Docker's terminology, a container is a single, running instance of a Docker image. In turn, there are two major categories of image: those which have been built by modifying and extending already existing images — known as parent images — and those which have been developed from scratch inheriting no instructions from previously built images — known as base images.

Base images typically consist of Unix-like operating system whose primary purpose is to act as a environment for further software to be installed. These base images include all of the libraries and tools that are typically shipped with these operating systems.

Most images are not base images but use a parent image which serves as a template. Users alter this template when first building the image by running a series of commands which may, for instance, consist of transferring software from the host machine to the image, downloading dependencies or exposing ports. This allows for software to be shipped with all of its requirements and, therefore, the container is a standardised environment in which the software can be executed.

Dockerfile. The specifications of a Docker image are defined in a text file known as Dockerfile which allows Docker to automatically build images. Every Dockerfile consists of an initial FROM command which indicates the parent image — or FROM scratch for base images. Following this, modifications are made to the base image via a series of further instructions, such as COPY to copy files from the host system, RUN to execute command line arguments or

ENTRYPOINT to specify a command that should be executed each time the container is started.

Each instruction builds an intermediate, read-only image, known as a layer, whose filesystem is stacked upon the one below. Docker utilises a union mount filesystem which allows layers to be reused, saving disk space and improving the build speed of images. Finally, a docker *container* has a final read/write layer — known as the container layer — on top of the underlying layers. This degree of separation between layers allows the user to add, alter and delete files from the container layer whilst preserving the integrity of the Docker image.

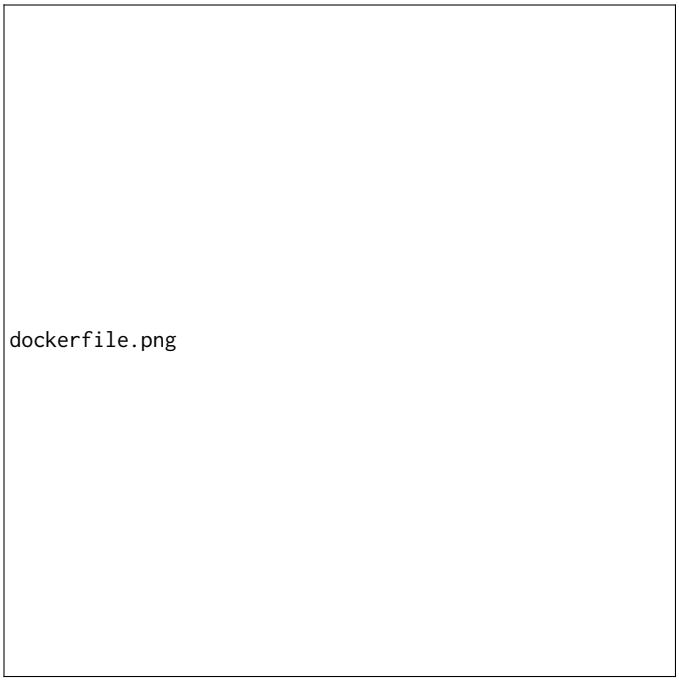

Figure 1: Basic Dockerfile for Tcpdump Container

6 DESIGN

7 CAPTURE

While traversing between to parties, a network packet can pass multiple connecting devices which direct the packet in the right direction. Any device in the immediate circuit traversed by a packet can capture and store it. In a monitoring setting, packets are usually captured by network routers and stored in the widespread *pcap* format. In case of space shortage or privacy concerns, the payload of a packet can be dropped in the saving process.

8 ACKNOWLEDGMENTS

We are grateful for our ongoing collaboration with our industry partners (blinded) on this topic area, who provided both ongoing support and guidance to this work. Discussions with them have helped reinforce the need for a better evaluation and understanding of the possibilities that new intelligent tools can provide.

Full funding sources after currently blinded.

A RESULT TABLES