DetGen: Data generation to bridge the "semantic gap" in network intrusion detection

ACM Reference Format:
. 2020. DetGen: Data generation to bridge the "semantic gap" in network intrusion detection. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this work, we introduce a new design paradigm for traffic generation testbeds that addresses the *semantic gap* in network intrusion detection by closely controlling different factors that influence generated network traffic and providing cross-linkage information between captured traffic and these factors. Our design relies on a composition of containers to enable capturing traffic directly from programs that run in an isolated and reproducible manner. Rather than simulating the large-scale behaviour of users in a realistic way, we aim to generate small-scale traffic scenarios that contain true interactions between software components in a realistic way to enable researchers a better understanding of particular traffic events.

Data-driven traffic analysis and attack detection is a centrepiece of network intrusion detection research, and the idea of training systems on large amounts of network traffic to develop a generalised notion of bad and benign behaviour appears like the solution to cyber-threats and has received *tremendous* attention in the academic literature. However, operational deployment is dominated by systems relying on more restrictive attack signatures. Already in 2010 Paxson and Sommer [21] have identified a number of issues that are summarised as an overall lack of connection between the nature of intrusion detection data and the applied data-driven detection systems, something the authors call the 'semantic gap'. These findings have since then been confirmed by other authors such as Harang [7] in 2014 or by Liu et al. in 2019 [13].

Among others, these issues include (1) fundamental difficulties for conducting sound evaluation of detection models and a (2) lacking perspective of a network operator that handles alerts, that result in a (3) semantic gap between the development of detection models and the structural and operational nature of network traffic and intrusion detection.

Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Data-centric breakthroughs in other fields have not been achieved solely by more complex and computationally more powerful ML-methods, but have been equally reliant on a precise understanding of the data and corresponding datasets that provide researchers with richer information and enable them to analyse weak points and model failures. As an example, results in *automatic speech recognition (ARS)* were not achieved by immediately training models on simply large annotated datasets. Initial models were reliant on highly sanitised and structured speech snippets in order to isolate low-level structures such as phonemes or time-warping. Lately, datasets that contain labelled specialised speech characteristics with varying intensity enable researchers to better understand ASR weak points such as emotional speech (RAVDESS), accents (Speech Accent Archive), or background noise (Urban Sound Dataset).

In a similar fashion, several approaches to enhance the way information is collected and presented have been successful in closing semantic gaps between data and detection systems in other areas of information security. Virtual machine introspection monitors and analyses the runtime state of a system-level VM to improve the understanding of virtual machine-based intrusion detection and forensic memory analysis [4]. The inclusion of threat reports to create behavioral feature labels enriches the way executables are described to enhance malware modelling and detection [20].

However, such efforts have not been made in network intrusion detection yet, with the current benchmark datasets paying more attention to the inclusion of a wide variety of attacks rather than the close control and detailed documentation of the generated traffic structures. This has so far lead to researchers predominantly applying of a number of ML-models directly to general traffic datasets in the hope of edging out competitors without analysing what traffic causes the model to fail and how design choices could prevent that.

This work provides the following contributions:

(1) We propose a novel design paradigm for generating reproducible small-scale traffic structures with ground-truth labels that contain extensive information about the computational interactions behind it.
(2) We present a novel and extensible network traffic generation framework called *DetGen* that implements our design paradigms to improve several shortcomings of current data generation frameworks for NIDS evaluation.
(3) We perform a number of experiments to demonstrate the fidelity to realism of the generated data.
(4) We present a number of use-cases to demonstrate how the design of our framework can boost evaluation and enhance understanding of ML-based network intrusion

detection systems to close the semantic gap described by Sommer and Paxson [21].

This framework is openly accessible for researchers and allows for straightforward customization.

1.1 Outline

Outline of the coming sections.

......

......

......

2 BACKGROUND AND MOTIVATION

2.1 Misuse and machine learning

Network intrusion detection is the field of detecting intrusions in a network by analysing captured traffic traces exchanged between computers in the network. Most commonly used are misuse detection systems identify known signatures of bad behaviour in traffic such as malicious packet payloads or rule-based patterns concerning port usage and/or packet sequences. Although very efficient, these methods are reliant on precise details on known attacks in the form of signature databases. Significant efforts have been invested in developing machine-learning based methods that are trained on large amounts of traffic to develop a more generalisable distinction between benign and malicious behaviour to remove the need of attack signatures and enable the detection of zero-day attacks.

2.2 Existing problems

Machine-learning based network intrusion detection has been subject to extensive criticism due to being unable to deliver sufficient detection rates at an acceptable false-positive rate in actual deployment. Two main causes for these failings have been identified particularly for network-based methods by Sommer and Paxson [21] in 2010, which have been supported and partly extended by Harang [7] in 2014 or by Liu et al. in 2019 [13]:

Semantic gap between between results and their operational interpretation. Arguably the biggest concern expressed by Sommer and Paxson is that methods lack a deep semantic insight into a system's capabilities and limitations and are instead treated as black boxes. The authors here draw comparisons to other areas of machine learning such as character recognition where the precise understanding of the data structure and how existing systems process it have lead to breakthroughs such as the convolutional layers that process the data in a more adequate way. In network intrusion detection, different methods are thrown at existing data without thorough analysis where the system performs well and where it fails or breaks, and what the reasons for this are. The authors recommend to researchers to narrow the scope to more specific applications and closely examine what types of traffic trigger which responses by the system in order to develop a better understanding of where and how future systems can be designed to better suit this particular type of data and application.

Fundamental difficulties for conducting sound evaluation. The semantic gap stems in part from persistent difficulties for researchers to evaluate their system thoroughly and in a comparable and reproducible manner due to a lack of appropriate public datasets. Privacy and security concerns discourage network administrators to release rich and realistic datasets for the public, leading to publicly available real-world datasets being the exception and missing informative features such as captured packets or consistent IP-addresses. This forces researchers to generate synthetic datasets using small virtual networks, and restricts the diversity and coverage of traffic researchers are able to examine.

Furthermore, the labelling process is significantly more difficult in network intrusion detection than in other domains with easier interpretable data. Often, only traffic directly involved in an attack is labelled manually, with all other traffic receiving the same 'Benign' label. This lack of informative labels impedes researchers abilities to analyse different types of traffic and thus understand the properties of their system.

The lack of benchmark datasets often forces researchers to assemble their own data, which is mostly done in a non-reproducible way, leading to unverifiable detection rates and incomparable results.

Other problems identified by Sommer and Paxson include the diversity of network traffic, the high cost of errors, and lacking computational speed or detection systems.

2.3 Goals and motivation

Our motivation for building *DetGen* is to provide a framework that generates information-rich and reproducible network traffic to help researchers understand traffic micro-structures and how they impact the performance of detection models in order to close the existing semantic gap and provide reproducible and verifiable network experiments. We focus on traffic micro-structures because even though many NID systems operate on this level, there exists little comprehensive research on general traffic behaviour[1] on the packet level, whereas longterm or network-wide traffic structures are far better understood should I insert citations here? [24].

We position our framework against NID datasets and data generation setups, such as those used in the CICIDS-17 or the UNSW-15 datasets citemoustafa2015unsw,sharafaldin2018towards, which are are predominantly used to evaluate network intrusion detection systems. We also aim to improve on general-purpose datasets and traffic generation setups such as the CAIDA anonymised traffic traces [23] or the insert framework, which offer real-world traffic rich in structure, but with no information or control over the factors responsible for shaping the corresponding data. For this, we emphasised the following aspects:

[1]Exceptions being on models for application fingerprinting

1. Rich control and ground truth information. Attention in the setup of typical lab-capture environments is put primarily into attack diversity and realistic network topologies and to some extent to the overall generation mechanisms of benign traffic. No attention so far has been spend on controlling and monitoring the different factors, described in Section 3, that influence how traffic, benign and malicious, is shaped in the generation process.

Our framework should above all produce ground truth information about the underlying activities of all captured traffic. This information should not only distinguish between benign and malicious activity, but give detailed information about the conducted computational activities. Furthermore, our framework should control and record all necessary factors that impact and shape the generated traffic such as network congestion or transmission failures to better facilitate understanding the effect of different traffic structures and particular phenomena on a detection system.

2. Reproducibility. The scientific method dictates that experiments must be reproduced before they are considered valid. Typical setups generate and collect data in a one-shot manner, without control over various quasi-random influences that affect the capture process. Furthermore, the particular setups are often complex and difficult to recreate. This makes is difficult for researchers to reproduce datasets and corresponding network experiments, especially when proprietary customized datasets are used.

Our framework should be able to precisely reproduce any generated traffic events and corresponding network experiments in order to facilitate verifiable and comparable research. Data should be produced in a controllable manner, with typically random impacts on traffic capture such as transferred data or host load being randomised and monitored in a controlled fashion to enable precise reproduction. We aim at avoiding complicated setups and plattform dependencies in the generation process.

3. Traffic realism on a packet level. As described in the first paragraph, attack-focused synthetic intrusion detection data setups so far have paid little attention to the realism of traffic on a packet-level, and exhibit far less event diversity than real-world captures due to the neglect of request diversity and the shielding from real-world factors such as excessive host load or network failures. We provide examples of this in Section insert Section number.

The DetGen framework should address these issues and produce traffic that exhibits realistic levels of structural trafficdiversity and rare events. This is a necessary condition to provide a corresponding evaluation of NIDS systems with a plausible degree of scientific insight and relevance.

Advantages for network experiments

- *In-depth model evaluation*: Drawing on the extensive labelling of granular activities and reproducible traffic generation, researchers have new opportunities to examine the performance of an intrusion detection model in-depth. Packet-level structures and resulting false-positives can be better associated with activities, which helps correct models better for identified weaknesses. Granular activities can be studied in a less noisy environment due to isolation and reproducibility.
- *Focus and understand novel attacks and traffic types*: Instead of being restricted to a restricted set of attacks and traffic types, researchers using DetGen can easily embed novel attacks such as the eternal blue exploit or new traffic types such as QUIC in a given network setup without abandoning the overall network coherence of the data.
- *Reproducible, open research*: Scientific experiments should be reproduced to be considered valid, and the use of containers has recently been promoted to enable easy reproduction of computational work by reducing the need for plattform and library dependencies. Network researchers can use DetGen to allow for the easy reproduction of generated network settings, generated data, and deployed network intrusion solutions.

		IDS-datasets			
	Detgen	VM-based	Generator-based	Real-world	Better name
Influence-monitoring	✓				
Influence-control	✓	limited	(✓)		
Reproducibility	✓	(✓)	(✓)		✓
Microstructure realism	✓		???	✓	

Table 1: Contributions compared to existing solutions

3 IMPACT FACTORS ON TRAFFIC MICRO-STRUCTURES

In order to enable sufficient and reproducible control over the generated traffic and provide the corresponding descriptive ground truth information , we first must understand what factors shape the traffic generation process. Computer communication involves a myriad of different computational aspects, and no research so far has been conducted to quantify how much influence each of them has on traffic structures. The following list highlights the most important influence factors on the traffic micro-structures observed on individual devices, as shown by other researchers or our own experiments: How do we verify that this list is more or less complete?

1. Application layer protocols. Without doubt the biggest impact on the captured traffic micro-structures is the choice or combination of the application layer protocols. Protocols such as HTTP/TLS perform vastly different tasks than protocols such as Peer-2-Peer or SMB, and thus perform different handshakes, experience different waiting times, transfer data in different intervals, or trigger different additional connections.

2. Performed task and application. The conducted computational task ultimately drives the communication between computers, and thus hugely influences characteristics such as the direction of data transfer, the duration and packet rate, packet sizes as well as the number of connections and performed protocol handshakes to conclude the task. Furthermore, the application used for the task has a significant influence on the generated traffic, as shown for different browser choices by Yen et al. [25] or for general application behaviour fingerprinting [22].

3. Transferred data. The amount of transferred data influences the overall packet numbers. Furthermore, the content of the data can potentially impact packet rates and sizes, such as shown by Biernacki [2] for streaming services.

Time	Source-IP	Destination-IP	**Dest. Port**
13:45:56.8	192.168.10.9	192.168.10.50	**21**
13:45:56.9	192.168.10.9	192.168.10.50	**10602**
13:45:57.5	192.168.10.9	69.168.97.166	**443**
13:45:59.1	192.168.10.9	192.168.10.3	**53**
13:46:00.1	192.168.10.9	205.174.165.73	**8080**

Table 2: Exemplary activity interval for host 192.168.10.9 in the CICIDS-17 dataset, containing FTP-, HTTPS- and DNS-, as well as additional unknown activity.

4. Caching/Repetition effects. Tools like cookies, website caching, DNS caching, known hosts in SSH, etc. remove one or more information retrieval requests from the communication, which can lead to altered packet sequences, less connections being established. For caching, this can result in less than 10% of packets being transferred, as shown by Fricker et al. [5].

5. Captured traffic from background activity. In traditional setups, all traffic generated on a host is recorded in the same capture, which makes it hard if not impossible to disentangle traffic from different activities and match them to their origin. Capturing background traffic typically leads to additional flows within the given time interval. 74% of SSH-connections and more than 95% of FTP- and HTTPS-connections in the CICIDS-17 dataset lie within a 5-second interval of connections from other background activity on the same network interface, as depicted in Table 2.

6. Application layer implementations. Different implementations for TLS, HTTP, etc. can yield different computational performance and can perform handshakes in slightly different ways. Furthermore, things like multiplexing channel prioritisation can have tremendous impact on the IAT times and the overall duration of the transfer, as shown in a study by Marx et al. [15] for the QUIC/HTTP3 protocol.

7. Host level load. In a similar manner, other applications exhibiting significant computational load (CPU, memory, I/O) on the host machine can affect the processing speed of

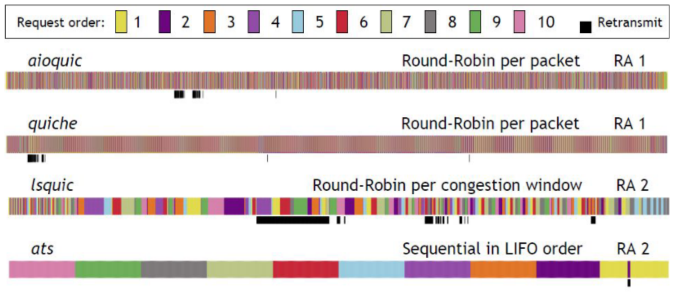


Figure 1: Comparison of QUIC connection request multiplexing for selected implementations, taken from [15].

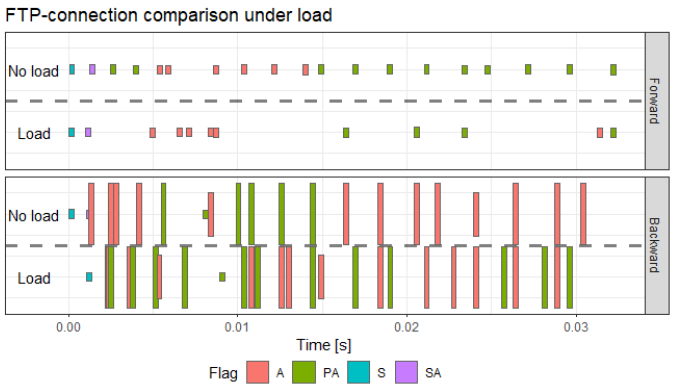


Figure 2: Packet-sequence structure similarity comparison for FTP-activity under different load and otherwise constant settings. Colours indicate packet flags while the height of the packets indicates their size. Note that under load, the host sends significantly less packets.

incoming and outgoing traffic, which can again alter IATs and the overall duration of a connection. An example of this is visible in Fig. 2, where the host sends significantly less ack-packets when under heavy computational load.

8. LAN and WAN congestion. Low available bandwith, long RTTs, or packet loss can have a significant effect on TCP congestion control mechanisms, which in turn influence frame-sizes, IATs, window sizes, and the overall temporal characteristic of the sequence. do we need to verify this? Seems very clear

We designed DetGen to control and monitor these factors in order to let researchers explore the impact of different aspects on their traffic models. We omitted some factors that can influence traffic structures, since these act either on a larger scale rather than micro-structures or correspond to exotic settings that are outside of our traffic generation scope. Among them are the following:

1. User and scheduled activities. The choice and usage frequency of an application and task by a user governs the larger-scale temporal characteristic of a traffic capture. Since we are focusing on the traffic micro-structures here, we currently omit this impact factor from our analysis.

2. Networking stack load. TCP or IP queue filling of the kernel networking stack can increase packet waiting times and therefore IATs of the traffic trace, as shown by [17]. In practice, this effect seems to be constrained to large WAN-servers and routers, and we did not find any significant effect on the described traffic similarity measures for various amounts of load generated with iPerf for a regular UNIX host.

3. Network configurations. Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes, are standardised for most networks where to find a proof for that?.

4 DETGEN ARCHITECTURE

4.1 Design overview

Detgen is a container-based network traffic generation framework that we developed to enable repeatable, controllable, and informative network experiments. In contrast to the pool of programs running in a VM-setup, DetGen separates program executions and traffic capture into distinct containerised environments in order to shield the generated traffic from external influences and enable the fine-grained control of traffic shaping factors.

Traffic is generated from a set of scripted *scenarios* (give examples here) that strictly control corresponding influence factors and offer the researcher to modify and label the conducted activity from a variety of angles and randomisations. Containers communicate in a virtual network created with Mininet along with virtual software switches, Ethernet links, routers, and firewalls.

4.2 Containerization and activity isolation

Containers are standalone packages that contain an application along with all necessary dependencies using OS-level virtualization. In contrast with standard Virtual machines (VMs), containers forego a hypervisor and the shared resources are instead kernel artifacts that can be shared simultaneously across several containers, leading to minimal CPU, memory, and networking overhead [12].

Due to the separation of processes, containers provide significantly more isolation of programs from external effects than regular OS-level execution. This isolation enables us to monitor processes better and create more accurate links between traffic events and individual activities than on a virtual machine were multiple processes run in parallel, which can all generate traffic. The corresponding one-to-one correlation between processes and network traces allows us to produce labelled datasets with significantly more granular ground truth information.

Insert some experimental result here.

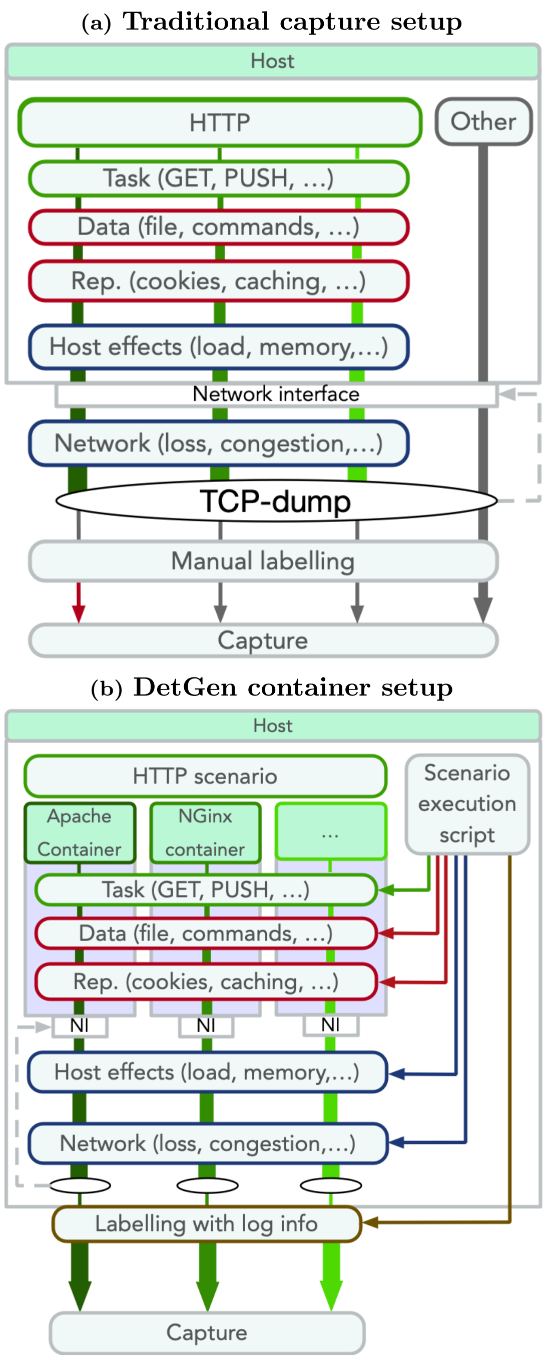

(a) Traditional capture setup
Host
HTTP
Other
Task (GET, PUSH, ...)
Data (file, commands, ...)
Rep. (cookies, caching, ...)
Host effects (load, memory,...)
Network interface
Network (loss, congestion,...)
TCP-dump
Manual labelling
Capture
(b) DetGen container setup
Host
HTTP scenario
Scenario execution script
Apache Container
NGinx container
...
Task (GET, PUSH, ...)
Data (file, commands, ...)
Rep. (cookies, caching, ...)
NI
NI
NI
Host effects (load, memory,...)
Network (loss, congestion,...)
Labelling with log info
Capture

Figure 3: Design comparison of traditional NIDS-data-setups and our DetGen

Containers are specified in an image-layer, which is unaffected during the container execution. This allows containers to be run repeatedly whilst always starting from an identical state. In combination with the container isolation, this allows us to perform network experiments that can be easily reproduced by anyone on any plattform insert citation.

4.3 Activity generation

Scenario. We define a *scenario* as a series of Docker containers conducting a specific interaction, whereby all resulting network traffic is captured from each container's perspective. This constructs network datasets with total interaction capture, as described by Shiravi et al. [19]. Each scenario produces traffic from a specific setting with two (client/server) or more containers. Examples may include an FTP interaction, a music streaming application, an online login form paired with an SQL database, or a C&C server communicating with an open backdoor. A full list of currently implemented scenarios can be found in Section 4.6. Each scenario is designed to be easily started via a single script and can be repeated indefinitely without further instructions, therefore allowing the generation of large amounts of data. Our framework is modular, so that individual scenarios are configured, stored, and launched independently. Adding or reconfiguring a scenario has no effect on the remaining framework.

When composing different settings, we most emphasised the inclusion of different **application layer protocols** such as HTTP or SSH, followed by the inclusion of different corresponding **applications** such as NGINX or Apache that steer the communication. We are currently aiming to also include options to use different **application layer implementations** such as TLS1.3 vs TLS1.2.

Task. In order to provide a finer grain of control over the traffic to be generated, we create a catalogue of different tasks that allow the user to specify the manner in which a scenario should develop. The aim of having multiple tasks for each scenario is to explore the full breadth of a protocol or application's possible traffic behaviour. For instance, the SSH protocol can be used to access the servers console, to retrieve or send files, or for port forwarding, all of which may or may not be successful. It is therefore appropriate to script a number of tasks that cover this range of tasks.

To implement a catalogue of tasks, we first examine the functionality of the underlying protocol and scenario setting before proceeding to adding tasks to the catalogue. To explore the breadth of the corresponding traffic structures efficiently, we prioritise to add tasks that cover aspects such as direction of file transfers (e.g. GET vs POST for HTTP), the amount of data transferred (e.g. HEAD/DELETE vs GET/PUT), or the duration of the interaction (e.g. persistent vs non-persistent tasks) as much as possible. For each task, we furthermore add different failure options for the interaction to not be successful (e.g. wrong password or file directory).

Since we always launch containers from the same state, we prevent traffic impact from **repetition effects** such as caching or known hosts. If an application provides caching possibilities, we implement this as an option to be specified before the traffic generation process.

Input randomization. Scripting activities that are otherwise conducted by human operators often leads to a loss of random variation that is normally inherent to the activity. As mentioned in Section ??, the majority of successful FTP transfers in the CIC-IDS 2017 data consist of a client downloading a single text file. In reality, file sizes, log-in credentials, and many other variables included in an activity are more or less drawn randomly, which naturally influences traffic quantities such as packet sizes or numbers.

We identify variable input parameters within scenarios and corresponding tasks and systematically draw them randomly from suitable distributions. Passwords and usernames, for instance, are generated as a random sequence of letters with a length drawn from a truncated Cauchy distribution, before they are passed to the corresponding container. Files to be transmitted are selected at random from a larger set of files, covering different sizes and file names.

4.4 Simulation of external influence

Network effects. Docker communication takes place over virtual bridge networks,

Communication between containers takes place over a virtual Mininet bridge network, which provides far higher and more reliable throughput than in real-world networks. Gates and Warshavsky [6] measured a bandwidth of over 90 Gbits/s without any lost packets using iPerf.8 This allows us to guarantee reliable and reproducible communication and thus remove external network effects on the captured traffic.

Virtual bridge networks furthermore enable us to retard and control the network reliability and congestion to a realistic level by using emulation tools. NetEm is an enhancement of the Linux traffic control facilities for emulating properties of wide area networks such as high latency, low bandwidth or packet corruption by adding delay, packet loss, duplication etc. to packets outgoing from a selected network interface [8].

We apply NetEm via a wrapping script to to the network interface of a given container, providing us with the flexibility to set each container's network settings uniquely. In particular, packet delays are drawn from a Paretonormal-distribution while packet loss and corruption is drawn from a binomial distribution, which has been found to emulate real-world settings well [10]. Distribution parameters such as mean or correlation as well as available bandwidth can either be manually specified or drawn randomly before the traffic generation process.

4.4.1 Host load. We simulate excessive computational load on the host with the tool *stress-ng*, a Linux workload generator. Currently, we only stress the CPU of the host, which is controlled by the number of workers spawned. Future work will also include stressing the memory of a system. We have investigated how stress on the network sockets affects the traffic we capture without any visible effect, which is why we omit this variable here.

4.5 Activity execution

Execution script. DetGen generates traffic through executing execution script that are specific to the particular scenario. The script creates the virtual network and populates it with the corresponding containers. The container network interfaces of the containers are then subjected to the NetEm chosen settings and the host is assigned the respective load, before the inputs for the chosen task are prepared and mounted to the containers.

The user can then choose how long and how often to execute the scenario. Once the activity is terminated, the script takes down the network and containers, and repeats the process for the next repetition. Randomised settings are drawn anew for each repetition.

Labelling and traffic separation. Each container network interface is hooked to a *tcpdump*-container that records the packets that arrive or leave on this interface. Combined with the described process isolation, this setting allows us to exclusively capture traffic that corresponds to the conducted activity and exclude any background events. The captured traffic is then saved and labelled as a pcap-file. The execution script then stores all parameters (conducted task, mean packet delay,...) and descriptive values (input file size, communication failure, ...) for the chosen settings in a file along with the corresponding pcap-filename.

4.6 Existing Scenarios

Our framework contains 29 scenarios, each simulating a different benign or malicious interaction. The protocols underlying benign scenarios were chosen based on their prevalence in existing network traffic datasets.These datasets consist of common internet protocols such as HTTP, SSL, DNS, and SSH. According to our evaluation, our scenarios can generate datasets containing the protocols that make up at least 87.8% (MAWI), 98.3% (CIC-IDS 2017), 65.6% (UNSW NB15), and 94.5% (ISCX Botnet) of network flows in the respective dataset. Our evaluation shows that some protocols that make up a substantial amount of real-world traffic are glaringly omitted by current synthetic datasets, such as BitTorrent or video streaming protocols, which we decided to include.

In total, we produced 17 benign scenarios, each related to a specific protocol or application. Further scenarios can be added in the future, and we do not claim that the current list exhaustive. Most of these benign scenarios also contain many subscenarios where applicable.

The remaining 12 scenarios generate traffic caused by malicious behavior. These scenarios cover a wide variety of major attack classes including DoS, Botnet, Bruteforcing, Data Exfiltration, Web Attacks, Remote Code Execution, Stepping Stones, and Cryptojacking. Scenarios such as stepping stone behavior or Cryptojacking previously had no available datasets for study despite need from academic and industrial researchers.

We provide a complete list of implemented scenarios in Table 3.

Name	Description	#Ssc.
Ping	Client pinging DNS server	1
Nginx	Client accessing Nginx server	2
Apache	Client accessing Apache server	2
SSH	Client communicating with SSHD server	5
VSFTPD	Client communicating with VSFTPD server	12
Wordpress	Client accessing Wordpress site	5
Syncthing	Clients synchronize files via Syncthing	7
mailx	Mailx instance sending emails over SMTP	5
IRC	Clients communicate via IRCd	4
BitTorrent	Download and seed torrents	3
SQL	Apache with MySQL	4
NTP	NTP client	2
Mopidy	Music Streaming	5
RTMP	Video Streaming Server	3
WAN Wget	Download websites	5
SSH B.force	Bruteforcing a password over SSH	3
URL Fuzz	Bruteforcing URL	1
Basic B.force	Bruteforcing Basic Authentication	2
Goldeneye	DoS attack on Web Server	1
Slowhttptest	DoS attack on Web Server	4
Mirai	Mirai botnet DDoS	3
Heartbleed	Heartbleed exploit	1
Ares	Backdoored Server	3
Cryptojacking	Cryptomining malware	1
XXE	External XML Entity	3
SQLi	SQL injection attack	2
Stepstone	Relayed traffic using SSH-tunnels	2

Table 3: Currently implemented traffic scenarios along with the number of implemented subscenarios

5 FIDELITY CONFIRMATION EXPERIMENTS

5.1 Traffic control and generation determinism

We now assess the claim of control over the outlined traffic influence factors, and how similar traffic generated with the same settings looks like. We also demonstrate that this level of control is not achievable on regular VM-based NIDS-traffic-generation setup.

To do so, we generate traffic from settings within which all controllable influence factors are held constant, both with DetGen framework and with a regular VM-based setup. Traffic samples from each setting should then be as similar as possible to provide sufficient experimental determinism. To measure how similar two traffic samples are, we devise a set of similarity metrics that measure dissimilarity of overall connection characteristics, connection sequence characteristics, and packet sequence characteristics:

- **Overall connection similarity** We collect 80 flow summary statistics (IAT and packet size, TCP window sizes, flag occurrences, burst and idle periods). We compress this information using PCA to 8 significant dimensions, and measure the cosine similarity between connections, which is also used in general traffic classification [1].
- **Connection sequence similarity** To quantify the similarity of a sequence of connections in a retrieval window, we use the following features to describe the window, such as used by Yen et al. [25]: The number of connections, average and max/min flow duration and size, number of distinct IP and ports addresses contacted. We then again measure the cosine similarity based on these features between different windows.
- **Sequential similarity** To quantify the similarity of packet sequences in traffic captures, we proceed in a similar fashion and assign packets a discrete state according to their flags, direction, sizes, and interarrival times. We then calculate the Markovian probability of each packet state conditional on the previous packet. We do this for sequences of 15 packets at the start, the middle, and the end of a connection, and use the average sequence likelihood of each group as a similarity measure. If connections are completely similar, the conditional probabilities and thus the likelihoods should converge to one.

In contrast to

We now apply the described similarity measures to different groups of traffic traces to evaluate the determinism of our data generation process. We vary the control settings of the described influence factors, described in Section 3, for each group, but keep them constant within each group. We use three different traffic types (HTTP, file-syncing (port 8384), and botnet), for which we both perform different application-specific activities and create different overall settings.

We furthermore perform the same HTTP-activities on two regular VMs without containerisation, but with the same settings.

Table 4 summarises the activity and settings for the different traffic groups, along with the corresponding dissimilarity scores for each group. As visible, the scores yield less than 1% of the dissimilarity observed on average for each protocol. Scores are especially low when compared to traffic groups collected in the VM setting, which is also visible in Fig. 4 for the HTTP-traffic. Dissimilarity scores for the VM-setting are most notably higher for the flow-metric, caused by additional background flows frequently captured. While sequential dissimilarity is roughly the same for the DetGen- and the VM-settings, overall connection similarity for the VM-setting sees significantly more spread in the dissimilarity scores when computational load is introduced.

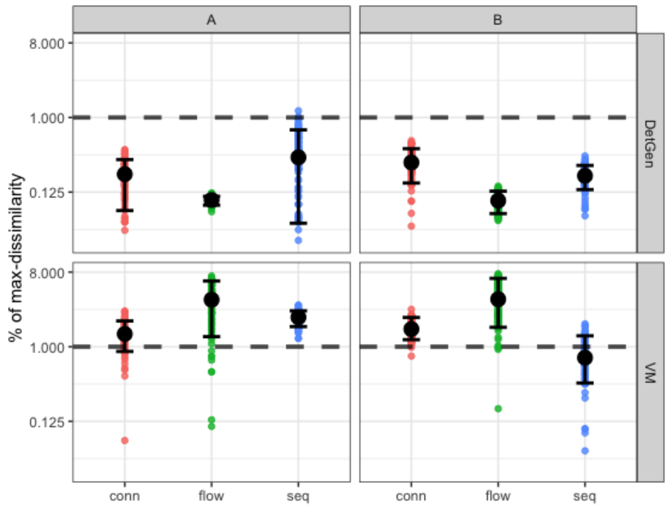


Figure 4: Comparison of HTTP-group dissimilarity scores for the DetGen-framework and a regular VM-setup, on a logarithmic scale. Samples from the VM-setting are consistently more dissimilar, in particular for flow-based metrics, where the average dissimilarity is more than 30 times higher than for the DetGen setting.

Label	Overall Setting	HTTP	File-Sync
A	Little congestion, high comp. load	Get-request on NGINX/wget, small file	Sequence 1, syncing to two computers
		0.20%, 0.0%, 0.18%	0.12%, 0.0%, 0.19%
B	Moderate congestion, low comp. load	Multi-request for different fixed files, NGINX	Sequence 2, syncing to four computers
		0.27%, 0.30%, 0.17%	0.11%, 0.23%, 0.24%
C	High congestion, no comp. load	Post-request on Apache/wget, large file	Sequence 3, syncing to two computers
		0.41%, 0.01%, 0.29%	0.39%, 0.0%, 0.21%
D	High congestion, high comp. load	Multi-request for different fixed files with caching, Apache	Sequence 4, syncing to four computers
		0.55%, 0.39%, 0.20%	0.31%, 0.15%, 0.38%
VM	comparison	0.55%, 0.39%, 0.20%	0.31%, 0.15%, 0.38%

Table 4: Outline of the traffic groups used for the determinism evaluation, along with the average dissimilarity percentages for each traffic group (red=connection similarity, green=flow-level similarity, blue=sequential similarity)

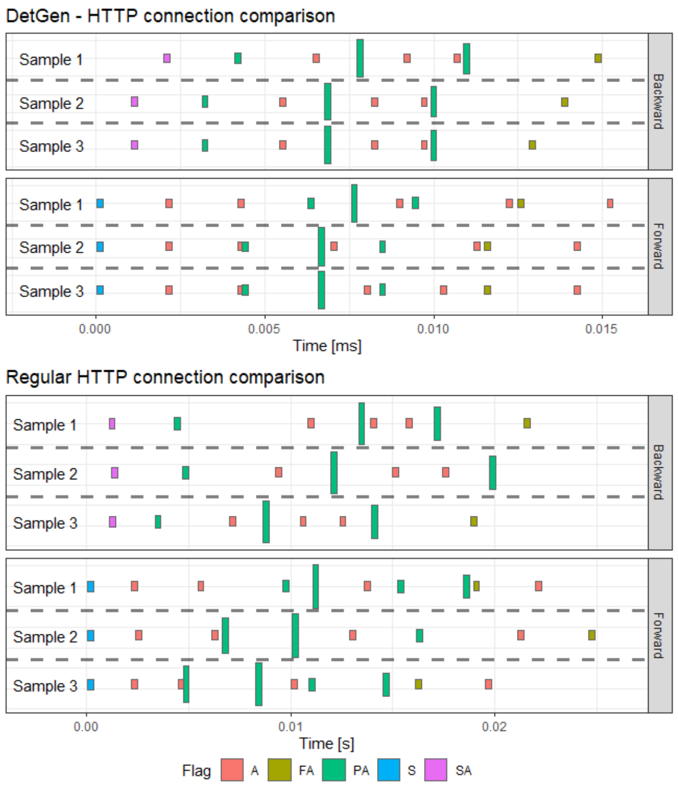


Figure 5: Packet-sequence structure similarity comparison for HTTP-activity under constant settings generated by the DetGen framework (left) and in a regular setting (right). Colours indicate packet flags while the height of the packets indicates their size. Note that in addition to more differences in the timing, the packet sizes vary more in the regular setting.

5.1.1 Realistic diversity level. The above test demonstrated that while holding settings constant, DetGen generates traffic with high similarity. We now prove that this does not impede DetGen's capability to generate realistic amounts of traffic diversity observed in real-world traffic. For this, we create a mixed dataset with varying settings for each protocol to explore the produced traffic diversity. Again, we use the proposed traffic similarity metrics to measure the overall traffic dissimilarity. We compare the observed dissimilarity for each protocol with that observed in real-world traffic from the CAIDA-2018 anonymized traffic traces insert citation, as well as the widely used synthetic CICIDS-17 intrusion detection dataset insert citation. Ideally, our generated traffic exhibits similar overall traffic diversity as the CAIDA real-world traffic.

We examine the achievable traffic diversity for HTTP-traffic, since large amounts of this protocol are present both in the CAIDA and the CICIDS-17 dataset. We randomly draw settings for all impact factors simultaneously during the traffic generation using realistic parameter sampling distributions, as described in Section insert correct section. After the traffic collection, we pair HTTP-connections at random to examine their dissimilarity. Figure 6 shows the dissimilarity distribution for the three traffic metrics for each dataset. As visible, neither the data from DetGen nor the CICIDS-17 dataset fully achieves the diversity of the CAIDA-data. However, DetGen achieves significantly more diversity for overall connection features and for packet sequences, areas where the data in the CICIDS-17 data is clustered very narrowly. insert distribution comparison measures

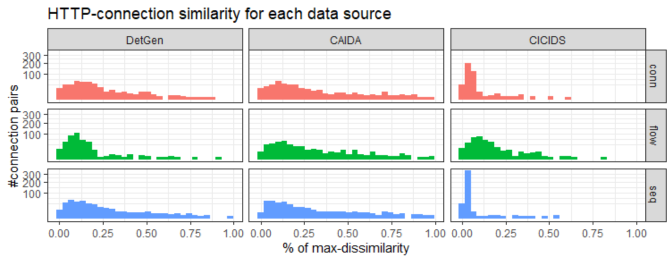


Figure 6: Scores for the LSTM-traffic classification model in dependence of simulated network congestion, along with the classification threshold

6 USE-CASES

6.1 Impacts of ground-truth information on model understanding

Extensive ground-truth labels on different traffic influences are arguably the most important contribution of the DetGen framework. We demonstrate the benefits of this additional information for model-understanding on two examples using a traffic classification model by Hwang et al. [9] and a highly regarded anomaly detection model by Casas et al. [3].

6.1.1 Improving traffic classification with congestion information. Our first use-case looks at congestion information to improve a recent state-of-the-art traffic classification model by Hwang et al. [9] that aims at identifying known malicious behaviour. The model classifies connections on a packet-level using an LSTM-network[2], and achieves detection and false-positive (FP) rates of **99.7%** and **0.03%** respectively. We train the model on a set of different HTTP-activities to detect SQL-injections. Rather than providing an accurate and realistic detection setting, this use-case shows how traffic information can be linked to model failures and slumping performance. We use HTTP-traffic from the CAIDA data as background traffic (95% of connections) and the SQL-injection attack traffic (2.5%) as well as different HTTP-activities for analysis (2.5%) using the DetGen-framework.

The initial model overall performs well, with a detection rate of 99.6%, but an improvable FP-rate of 0.8%. To explore potential causes of misclassification, we analysed detection

[2]Long-short-term-memory neural network

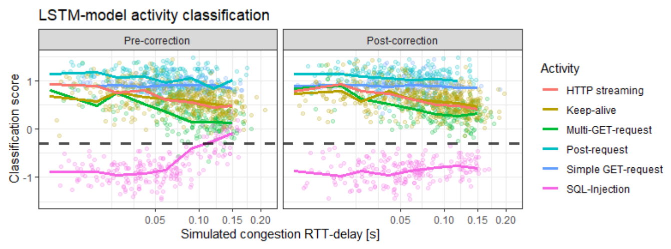


Figure 7: Scores for the LSTM-traffic classification model in dependence of simulated network congestion, along with the classification threshold

rates in dependence of different traffic influences. Looking at the left panel of Figure 7, which depicts classification scores in dependence of simulated network congestion, we learn that while classification scores are well separated for lower congestion, increased latency leads to misclassification, especially for SQL-injection traffic. A plausible cause would be that the increased amount of retransmission sequences decreases the overall sequential coherence for the model. We try to correct the existing model by excluding retransmission sequences from the model input data, which leads to significantly better classification results during network latency, as visible in the right panel of Figure 7. Overall detection rates and false positives improved to **99.9%** and **0.045%**.

6.2 Refining the notion of benign traffic for anomaly detection

Our second use-case looks at the effect of sudden connection terminations on an anomaly-detection model by Casas et al. [3]. The model takes a number of flow summary statistics as input, which include such as packet size and interarrival statistics, number of idle and transfer periods, flag occurrencies etc. as input and projects it into different subspaces, where the connections are clustered. Anomalous outliers are detected by accumulating the Mahalanobis-distance from the cluster centers from each subspace.

We train the model on benign traffic from the CICIDS-17 intrusion detection dataset (80%). Since intrusion detection datasets often lack sufficient events that necessarily occur, but with lesser frequency (such as communication failures), we also include traffic generated using DetGen (HTTP, FTP, SSH, and SMTP, 20%) using a wide spectrum of settings. Attack data for the evaluation was again provided through the CICIDS-17 dataset.

In the evaluation of the model, it became apparent that the model often assigns Brute-Force Web attacks and some HTTP-traffic from a particular cluster similar anomaly scores. When looking at this cluster, we notice that despite most connections being well centred well together, some connections are scattered much more broadly across the cluster and mix with Brute-Force traffic. This is visible in Figure 8. When looking at the activity labels from DetGen, we notice that most of this traffic corresponds to half-open connections

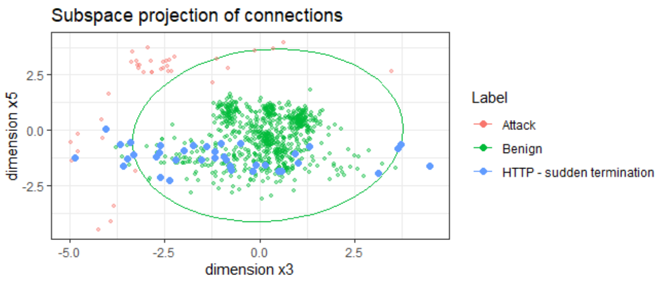


Figure 8: Scores for the LSTM-traffic classification model in dependence of simulated network congestion, along with the classification threshold

which were dropped by the server due to network failure. One defining characteristic of such connections are that they are not closed with a termination handshake using FIN-flags. We therefore included an additional flow-summary describing whether the connection was terminated properly, which lead to half-open connections being projected very differently into a new cluster with well defined borders.

6.3 Investigating adversarial perturbation attacks and model inversion

7 CURRENT DATA SITUATION

Currently, intrusion detection researchers predominantly rely on public, synthetically generated datasets, on which NID systems are evaluated subsequently. *Real-world datasets* such as LANL-15 [11] or UGR-16 [14] provide the highest amount of traffic realism, but often lack detailed information such as packet captures due to privacy reasons, and give close to no information on the content of the provided data.

Synthetic datasets such as the CICIDS-17 [18] or the UNSW-16 [16] datasets are typically captured in virtual environments that simulate commercial networks with virtual machines. Traffic is generated from scripted activity, and attack data either injected or generated from carefully inserted vulnerabilities. The arranged settings normally lack the flexibility to generate customized data and by design only provide very limited attack diversity.

Attack traffic generators typically aim at providing traces from a diverse set of attacks, and injecting them into existing traffic captures in various ways. Moirai citation for example calculates several quantitative characteristics to better embed the attack traffic. However, most of the issues surrounding real-world traffic captures remain, and there is concern about the realism of injected attack traffic citation.

Recently, some effort have been made to to generate completely artificial traffic data with *generative adversarial networks* (GANs) trained on real-world traffic. While examples such as DoppelGANger or Ring et al. citation are successful at generating realistic large-scale network features such as activity levels or connection graphs, they are not aimed at intrusion detection and do not provide the necessary granularity to model connection- or packet-level features.

Testbeds such as Mininet offer tremendous flexibility, but are so far not targeted for intrusion detection and lack suitable small-scale traffic generation tools, labelling capabilities, or attack scenarios.

7.1 Future work

CONTRIBUTION PAGE

Framework

Controlling traffic influences and detailed label creation.

- Attention in the setup of lab-capture is most put into attack inclusion and larger network topology, less so into the generation benign traffic, and no attention so far has been spend on controlling the different factors of influence given in Table **??**.
  - We discuss the different influence layers, for well known ones we refer to previous work, for others we demonstrate the effect in experiments. We discuss how we control the influence either through simulation or specific setups
  -
- Traditional lab-capture setups that consist of a network of virtual or test machines typically capture all traffic in one capture-*channel* at the router without a distinction of different traffic types. Variation in traffic is however introduced at many levels (application type, specific application task, caching/cookies, load/buffering, congestion/loss, etc.), for which information is not recorded and impossible to recreate through the usual manual or rule-based labelling that is used to identify malicious activity.
- Through the use of containers, our testbed design facilitates different capture channels and enables us to record information on these variations for a more complete and granular data capture than traditional setups. Containerised applications isolate task executions more than VM-setups and by capturing traffic directly at each container network interface, we can collect traffic from different applications in different places that would otherwise be merged into the same capture due to being send via the same network interface.
- We combine this setup with a set of scripted activities with behaviours (FTP-file-transfer, HTTP-website fetch,...) and sub-behaviours (waiting time, transfer fails, ...) that are precisely defined and for which we can extract information-rich labels. This way, we can extract traffic labels containing ground-truth with a very granular resolution on the generating activity.
- **Closing the semantic gap**
  - Granular information about the activities and settings generating particular traffic traces enables researchers to match particular traffic structures or model misbehaviour with corresponding computational actions, something that is not possible with current datasets and testbeds. This should help researchers to better understand the logical small-scale structures in network traffic and understand the effect of different types of traffic on models.
  - Furthermore, the separation of applications through the use of containers makes our framework modular and enables researchers to quickly swap particular containers to measure the effect of different implementations, or to add different attack types to a setting with *Metasploitable*-containers without having to worry about version-dependent vulnerabilities.

Reproducibility. Several factors in our design contribute to making the data and corresponding network experiments particularly reproducible:

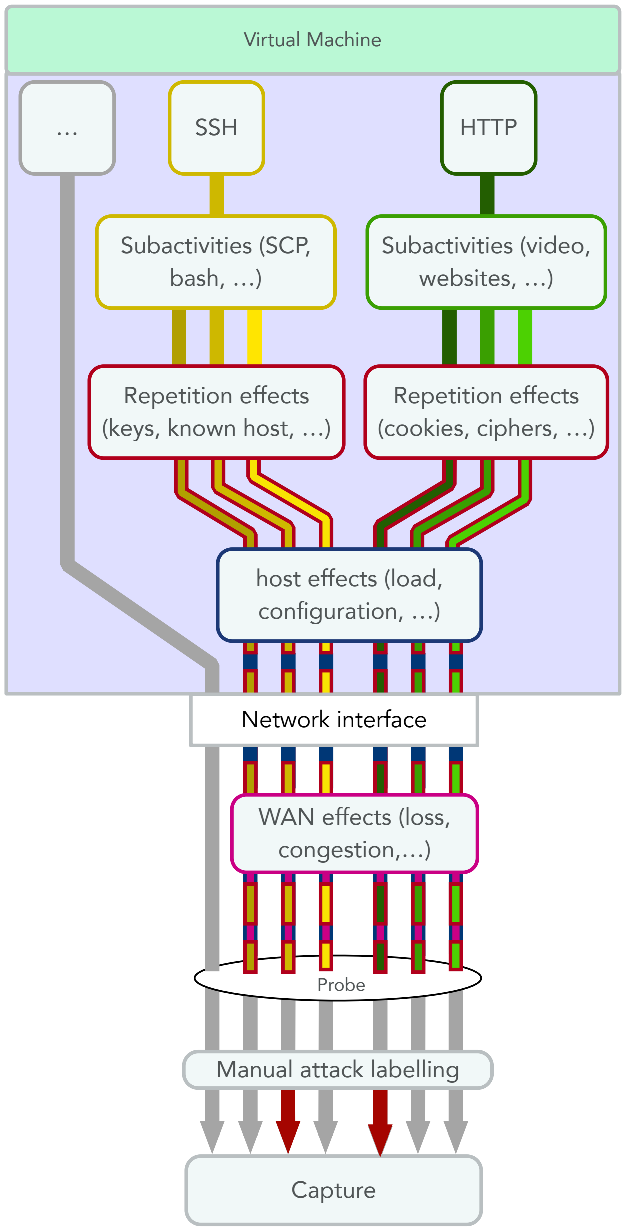
- Docker containers by design operate platform-independent without the need for muddled dependencies, and the separation of running containers from container images in contrast to VMs guarantees repeatable tasks without "code rot".
- We introduce adjustable degrees of input (files, passwords,...) and traffic (congestion, network fails) randomisation for our scenarios, with labels describing the particular setting being extracted. This way, similar scenarios and settings can be easily recreated.
- Intrusion detection systems in the form of firewalls or ... can be included directly in the framework as containers to enable other researchers to verify the achieved results.

Analysis

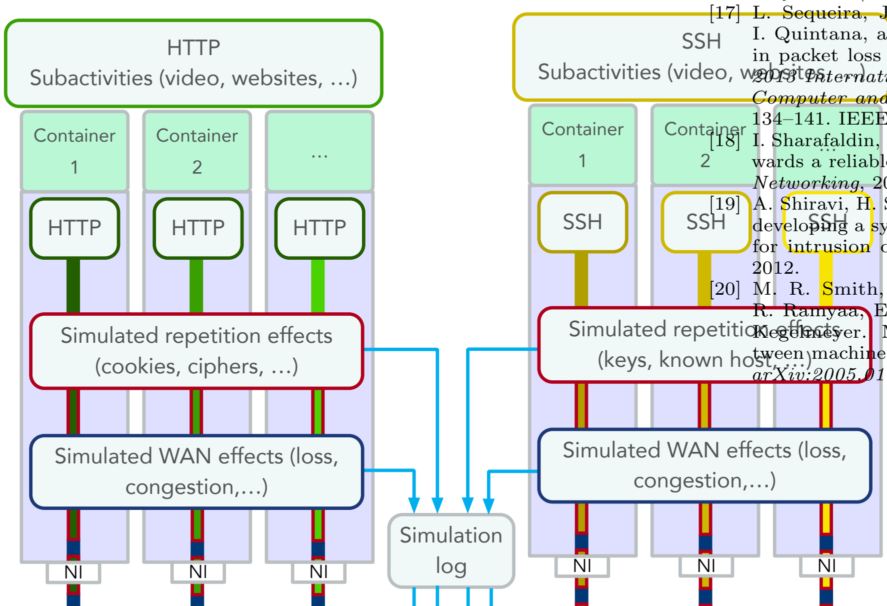
Details on our use-cases and framework analysis

REFERENCES

[1] Y. Aun, S. Manickam, and S. Karuppayah. A review on features' robustness in high diversity mobile traffic classifications. *International journal of communication networks and information security*, 9(2):294, 2017.
[2] A. Biernacki. Analysis and modelling of traffic produced by adaptive http-based video. *Multimedia Tools and Applications*, 76(10):12347–12368, 2017.
[3] P. Casas, J. Mazel, and P. Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772–783, 2012.
[4] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *2011 IEEE symposium on security and privacy*, pages 297–312. IEEE, 2011.
[5] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 310–315. IEEE, 2012.
[6] M. Gates and A. Warshavsky. Iperf Man Page. https://linux.die.net/man/1/iperf. Accessed: 2019-08-11.
[7] R. Harang. Bridging the semantic gap: Human factors in anomaly-based intrusion detection systems. In *Network Science and Cybersecurity*, pages 15–37. Springer, 2014.
[8] S. Hemminger et al. Network emulation with netem. In *Linux conf au*, pages 18–23, 2005.
[9] R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang. An lstm-based deep learning approach for classifying malicious traffic at the packet level. *Applied Sciences*, 9(16):3414, 2019.
[10] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. I. Wang. An empirical study of netem network emulation functionalities. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2011.
[11] A. D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.


Virtual Machine
…
SSH
HTTP
Subactivities (SCP, bash, …)
Subactivities (video, websites, …)
Repetition effects (keys, known host, …)
Repetition effects (cookies, ciphers, …)
host effects (load, configuration, …)
Network interface
WAN effects (loss, congestion,…)
Probe
Manual attack labelling
Capture

(a) **Traditional capture setup**


HTTP
Subactivities (video, websites, …)
Container 1
Container 2
…
HTTP
HTTP
HTTP
Simulated repetition effects (cookies, ciphers, …)
Simulated WAN effects (loss, congestion,…)
NI
NI
NI
Simulation log
SSH
Subactivities (video, websites, …)
Container 1
Container 2
SSH
SSH
SSH
Simulated repetition effects (keys, known host, …)
Simulated WAN effects (loss, congestion,…)
NI
NI
NI

Contribution	How containers enable it	VM-based	F c
Ground truth labels on granular activities	Isolated process in container, no additional events. Easy to separate even when containers attached to same network interface	background events (system activitiy, artifacts from earlier activity, …), overlaying activities hard to separate	N e c a
Reproducability	Container state the same after repeated launches, isolation means that other task have little effect on, plattform independence	To a lesser degree, simultaneously conducted activity can affect each other	-
Easy to update, include new attacks	Attack/victim containers remove dependence on vulnerabilities, can be hooked to existing scenarios with shared resources	Since attacks happen on the VM, it is very hard to compose a VM with enough vulnerabilities to include many attacks	-
Data on attacks on container isolation	In need of existing scenarios for background data	No	N

Table 5: Contributions compared to existing solutions

[12] K. Kolyshkin. Virtualization in linux. *White paper, OpenVZ*, 3:39, 2006.

[13] H. Liu and B. Lang. Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, 9(20):4396, 2019.

[14] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón. Ugr '16: A new dataset for the evaluation of cyclostationarity-based network idss. *Computers & Security*, 73:411–424, 2018.

[15] R. Marx, J. Herbots, W. Lamotte, and P. Quax. Same standards, different decisions: A study of quic and http/3 implementation diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, pages 14–20, 2020.

[16] N. Moustafa and J. 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. IEEE, 2015.

[17] L. Sequeira, J. Fernández-Navajas, L. Casadesus, J. Saldana, I. Quintana, and J. Ruiz-Mas. The influence of the buffer size in packet loss for competing multimedia and bursty traffic. In *2013 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 134–141. IEEE, 2013.

[18] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani. Towards a reliable intrusion detection benchmark dataset. *Software Networking*, 2018(1):177–200, 2018.

[19] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374, 2012.

[20] M. R. Smith, N. T. Johnson, J. B. Ingram, A. J. Carbajal, R. Ramyaa, E. Domschot, C. C. Lamb, S. J. Verzi, and W. P. Kegelmeyer. Mind the gap: On bridging the semantic gap between machine learning and information security. *arXiv preprint arXiv:2005.01800*, 2020.

[21] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[22] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are? smartphone fingerprinting via application behaviour. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 7–12, 2013.

[23] C. Walsworth, E. Aben, K. Claffy, and D. Andersen. The caida ucsd anonymized internet traces 2012,", 2015.

[24] W. Willinger, V. Paxson, and M. S. Taqqu. Self-similarity and heavy tails: Structural modeling of network traffic. *A practical guide to heavy tails: statistical techniques and applications*, 23:27–53, 1998.

[25] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter. Browser fingerprinting from coarse traffic summaries: Techniques and implications. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 157–175. Springer, 2009.