Contribution Title⋆

First Author[1][0000−1111−2222−3333], Second Author[2,3][1111−2222−3333−4444], and Third Author[3][2222−−3333−4444−5555]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
`lncs@springer.com`
`http://www.springer.com/gp/computer-science/lncs`
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
`{abc,lncs}@uni-heidelberg.de`

Abstract. We introduce DetGen, tool that generates traffic to improve the ability to probe and understand model behaviour in data-driven network intrusion detection, and help explain the corresponding decisions made by a model. DetGen operates under a new design paradigm based on containerisation and reproducibility in order to closely controlling different factors that influence generated network traffic and providing cross-linkage information between captured traffic and these factors. In this work, we demonstrate how DetGen operates.

– We examine how well DetGen is able to control different types of traffic characteristics, and compare the corresponding determinism to common VM-based traffic generation setups.
– We also examine the performance of DetGen in the other direction, namely the ability to generate traffic with realistic levels heterogeneity, and compare the results against those observed in existing artificially generation NID-datasets.
– We present an exemplary dataset that is suitable for a broad probing of models trained on the CICIDS-17 dataset, as it mirrors its range of protocols and attacks.
– We demonstrate the extensive probing of an LSTM-based anomaly detection model with this dataset, and demonstrate how to lower false-positives effectively by understanding where the model fails to process particular traffic structures correctly.

1 Introduction

In this work, we introduce a new traffic generation tool that improves the ability to probe and understand model behaviour in data-driven network intrusion detection, and help explain the corresponding decisions made by a model.

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, existing datasets fall short on available ground-truth information about particular traffic characteristics that could affect model performance. Furthermore, the generation process for synthetic traffic usually does not provide sufficient structural nuances to explore the behaviour of model .insert citation

Machine-learning breakthroughs in other fields have often been reliant on a precise understanding of data structure and corresponding descriptive labelling to develop more suitable models. Initial models in *automatic speech recognition (ASR)* for example were reliant on highly sanitised and structured speech snippets in order to isolate low-level structures such as phonemes or time-warping, before the understanding of these structures lead to the success of more layered models of feed-forward and recurrent neural networks and more recently fully end-to-end trained models. Lately, datasets that contain labelled specialised speech characteristics 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 improving understanding between data and detection systems in different areas of information security. Virtual machine introspection monitors and analyses the runtime state of a system-level VM, and the inclusion of threat reports to create behavioural feature labels enriches the way executables are described [13]. Recently, data provenance tools aim to improve the representation of system executions [1] over traditional logs.

However, such efforts have not been made in network intrusion detection yet, with the current quasi-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. Data containing ground-truth on the traffic generation process to link observable structures with corresponding computational activities is rare, which has so far lead researchers to predominantly apply a number of ML-models to traffic datasets in the hope of edging out competitors. This overall lack of connection between the nature of intrusion detection data and the applied data-driven detection systems has been identified as a 'semantic gap' by Paxson and Sommer [14], and is seen to be partly responsible for the lack of success machine-learning had in network intrusion detection. This claim has been supported and partly extended by Harang [3] in 2014 and by Liu et al. in 2019 [9].

This work provides the following contributions:
In this work, we demonstrate how DetGen operates.

1. We propose DetGen, a framework based on 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 explain how DetGen enables accurate and quasi-deterministic control over traffic characteristics as well as corresponding ground truth informtation, and compare the design advantages to traditional generation set-ups.
3. We examine how well DetGen is able to control different types of traffic characteristics, and compare the corresponding determinism to common VM-based traffic generation setups.
4. We also examine the performance of DetGen in the other direction, namely the ability to generate traffic with realistic levels heterogeneity, and compare the results against those observed in existing artificially generation NID-datasets.
5. We present an exemplary dataset that is suitable for a broad probing of models trained on the CICIDS-17 dataset, as it mirrors its range of protocols and attacks.
6. We demonstrate the extensive probing of an LSTM-based anomaly detection model with this dataset, and demonstrate how to lower false-positives effectively by understanding where the model fails to process particular traffic structures correctly.

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

1.1 Outline

2 Motivation and Methodology

2.1 Motivation

Scientific machine learning model development requires both **model evaluation**, in which the overall predictive quality of a model is assessed to identify the best model, as well as **model validation**, in which the behaviour and limitations of a model is assessed through targeted **model probing**. Model validation is essential to understand how particular data structures are processed, and enables researchers to develop their models accordingly. Data generation tools for rapid model probing in other domains such as the *What-If tool* [17] underline the importance of model validation.

We designed DetGen to provide traffic with exact control over various influence factors to allow ...
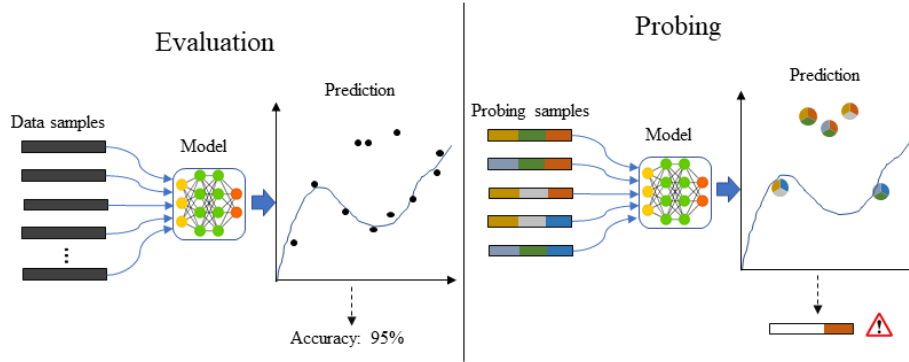
Fig. 1: Comparison between numerical model evaluation and model probing with specifically controlled data characteristics, indicated as colours.

2.2   Use-case:Improved traffic separation for a classifier with congestion level information

Our first example looks at how descriptive ground truth information on traffic characteristics can improve a traffic classification model through the analysis of data separation in dependence of different traffic features. For this, we use a recent traffic classification model by Hwang et al. [6] as an example, which aims at distinguishing various types of malicious activity from benign traffic. The model achieved some of the highest detection rates of packet-based classifiers in a recent survey [15]. The model classifies connections on a packet-level using a *Long-short-term memory* (LSTM) network [4], and is claimed to achieve detection and false-positive (FP) rates of **99.7%** and **0.03%** respectively.

We train a model on a set of different HTTP-activities in order to detect SQL-injections. Rather than providing an accurate and realistic detection setting, this example shows how traffic information can be linked to model failures and slumping performance. We use real-world HTTP-traffic from the *CAIDA anonymized traffic traces* [16] as background traffic (85% of connections) and add SQL-injection attack traffic (7.5%) as well as different HTTP-activities for probing (7.5%). In total, we use 50,000 connections for training the model, or slightly less than 2 million packets.

The initially trained model performs relatively well, with an *Area under curve* (AUC)-score[5] of **0.981**, or a detection and false positive rate[6] of **0.96%** and **2.7%**. However, these rates are still far from enabling operational deployment. Now suppose we want to improve these rates to both detect more SQL-injections and retain a lower false-positive rate.

---

[4] a deep learning design for sequential data

[5] a measure describing the overall class separation of the model
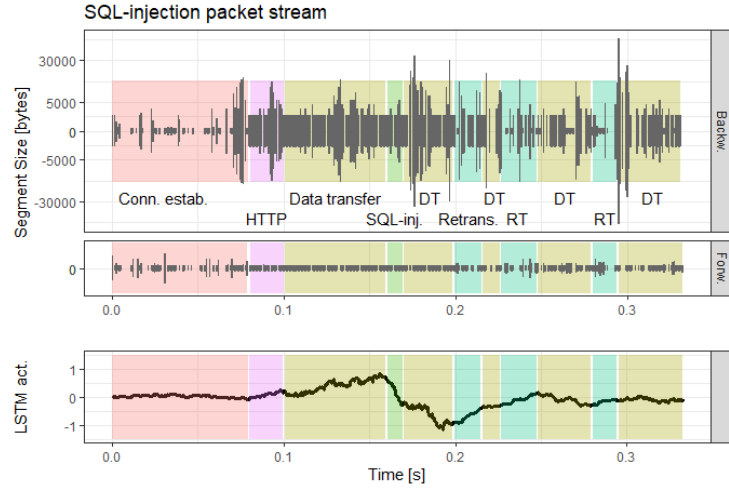
[6] tuned for the geometric mean

Fig. 2: LSTM-output activation in dependence of connection phases. Depicted are packet segment streams and their respective sizes in the forward and backward direction, with different phases in the connection coloured and labelled. Below is the LSTM-ouput activation while processing the packet streams.

We initially explore which type of connections are misclassified most often. For this, we perform a correlation analysis between the numeric or categoric labels available for the probing data, and the binary response whether the corresponding connection was misclassified. Unsurprisingly, the highest correlation to misclassification was measured for the conducted activity, with a particular attack scenario (19% correlation) and connections with multiple GET-requests (11% correlation) being confused most often. This was followed by the amount of simulated latency (12% correlation), which we are now examining closer.

Fig. 3 depicts classification scores of connections in the probing data in dependence of the emulated network latency. The left panel depicts the scores for the initially trained model, which shows that while classification scores are well separated for lower congestion, increased latency in a connection leads to a narrowing of the classification scores, especially for SQL-injection traffic. Since there are no classification scores that reach far in the opposing area, we conclude that congestion simply makes the model lose predictive certainty. Increased latency can both increase variation in observed packet interarrival times (IATs), and lead to packet out-of-order arrivals and corresponding retransmission attempts. Both of these factors can decreases the overall sequential coherence for the model, i.e. that the LSTM-model loses context too quickly either due to increased IAT variation or during retransmission sequences.

To examine the exact effect of retransmission sequences on the model output, we generate two similar connections, where one connection is subject to moderate packet loss and reordering while the other is not. We then compare how the
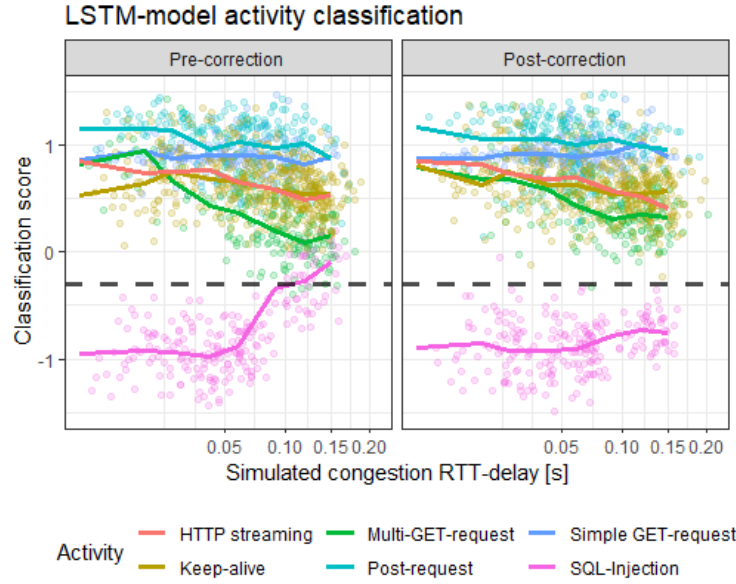
Fig. 3: Scores for the LSTM-traffic classification model in dependence of simulated network congestion, along with the classification threshold.

LSTM-output activation is affected by retransmission sequences. Fig. 2 depicts the evolution the LSTM-output layer activation in dependence of difference connection phases. Initially the model begins to view the connection as benign when processing regular traffic, until the SQL-injection is performed. The model then quickly adjusts and provides a malicious classification after processing the injection phase and the subsequent data transfer. The negative output activation is however quickly depleted once the model processes a retransmission phase, and is afterwards not able to relate the still ongoing data transfer to the injection phase. When comparing this to the connection without retransmissions, we do not encounter this depletion effect, instead the negative activation persists after the injection phase.

We try to correct the existing model with a simple fix by excluding retransmission sequences from the model input data, both during training and classification. This leads to significantly better classification results during network latency, as visible in the right panel of Fig. 3. SQL-injection scores are now far-less affected by congestion while scores for benign traffic are also less affected, albeit to a smaller degree. The overall AUC-score for the model improves to **0.997** while tuned detection rates and false positives improved to **99.1%** and **0.045%**.

3 Refining the notion of benign traffic for anomaly detection

Next, we show how ground-truth traffic information can help produce more coherent clusters and thus refine the benign traffic model in anomaly-detection. In particular, we will examine a simplified version of *Kitsune* [10], a recent deep learning anomaly-detection model based on stacked autoencoders. *Kitsune's* AUC-scores surpassed those of other state-of-the-art methods for a variety of attacks, including various types of Botnet traffic and *man-in-the-middle* attacks.

The model takes connection packet streams as input, which are pushed through an artificial information bottleneck before reconstruction, which forces the model to learn and compress reoccurring traffic structures. The compressed connection representation is essentially a positional projection into a lower-dimensional vector space, where spatial boundaries around benign traffic can be drawn. For demonstration purposes, we use a widely-used clustering approach for anomaly-detection rather than *Kitsune's* more complex ensemble method. Here, anomalous outliers are detected using the Mahalanobis-distance of a projected connection from identified cluster centers. Benign traffic should ideally be distributed evenly around the cluster centres to allow a tight borders and good separation from actual abnormal behaviour.

Unstructured datasets such as the CAIDA traffic traces assumably contain too much abnormal behaviour to train an anomaly-detection model, which is why we train the model on benign traffic from the CICIDS-17 [11] intrusion detection dataset (80%). Again, we add 20% probing traffic consists of HTTP, FTP, SSH, and SMTP communication, using a wide spectrum of settings for examination purposes. Attack data for the evaluation was again provided through the CICIDS-17 dataset, and includes access attacks such as SQL-injections or Brute-Forcing, as well as Mirai botnet traffic. We train the model with in total 150,000 connections.

3.1 Projection coherency evaluation

Like many approaches that generate representations of benign traffic for anomaly detection, *Kitsune* projects traffic events into a vector-space where traffic clusters and similarities become more apparent. In order for the projection to accurately capture important traffic structures, this projection should be consistent, i.e. traffic events with similar origins and characteristics should be projected to similar positions rather than be dispersed throughout the vector space [5].

To verify the models projection consistency, we generate traffic from near-identical conditions to provide certainty on the expected traffic similarities. We generate a small dataset that consists of HTTP-requests, file-synchronisation, and Botnet communication. For each of the three traffic types we fix four settings that vary in the performed activity and network latency, with the traffic shaping described in Section ?? being held constant within each setting except for small

variations in the transmitted message or file. Table **??** summarises the traffic for each setting.

We verify if traffic samples within each group are projected to similar areas by measuring the average and maximum Mahalanobis-distance to quantify the overall dispersion of the samples. The results are displayed in Table **??** and depicted in Fig. 4. The first thing to notice is that the model projects samples from each group within the same cluster, thus confirming the capture of a coarse traffic structure. When looking at the traffic dispersion and the corresponding Mahalanobis-distance measurements, we notice that the *multi-request HTTP* traffic as well as the *file-synchronisation* between mutliple computers is much further dispersed than in the other settings, especially when exposed to more latency. We also find that the corresponding dimension, x_3, with the most projected dispersion seems to be the same for each of the four settings. This suggests that the cause for the dispersion is the same for the different traffic types.

We now focus on the influence of input features on the projected positions exclusively in the x_3-direction. Here, we can again perform a simple correlation analysis between different the input feature values and the corresponding x_3-value. We observe that the arrival time of packet bears the most correlation (5.4%) for the selected settings. We also see that this influence is concentrated primarily on connections that are opened shortly after a previous connection, with the temporal separation between these two connections apparently being the primary cause for the spread on the x_3-axis. The connection interarrival times are naturally an important feature for *Kitsune* to detect attacks such as *Man-in-the-Middle*, which could explain the weight this feature plays in the projection process.

4   DetGen Architecture

4.1   Design overview

Detgen is a container-based network traffic generation framework that dis. 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.

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

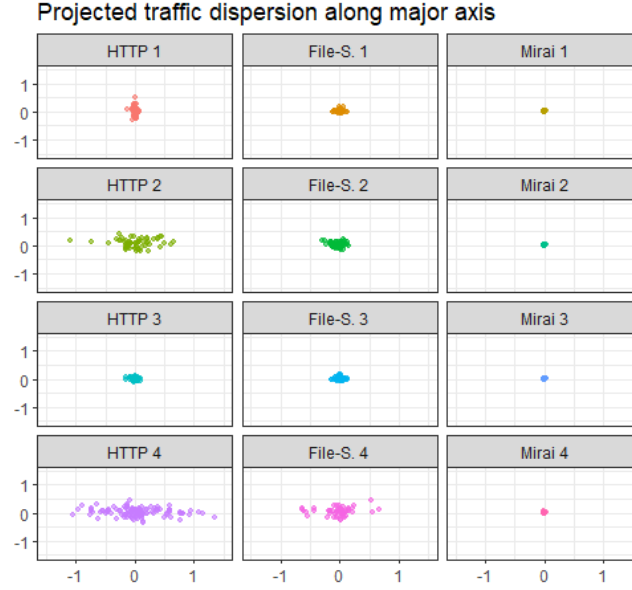Fig. 4: Dispersion of projected traffic samples from each setting, plotted along the two most dispersed axes.

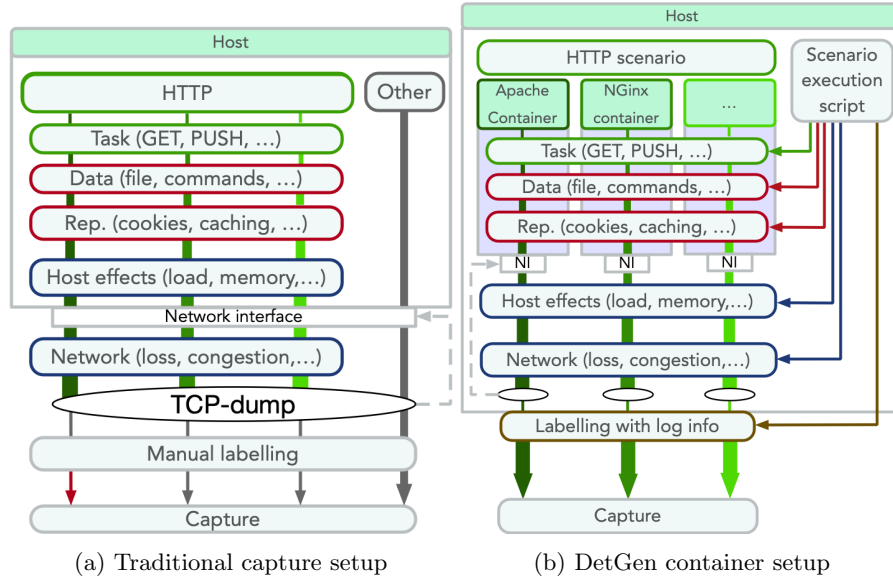(a) Traditional capture setup

(b) DetGen container setup

Fig. 5: Design comparison of traditional NIDS-data-setups and our DetGen framework

are instead kernel artifacts that can be shared simultaneously across several containers, leading to minimal CPU, memory, and networking overhead [8].

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.

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. [12]. 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 [2] 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 [4].

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 [7]. 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.

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.

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

Name	Description	#Ssc.
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 1: Currently implemented traffic scenarios along with the number of implemented subscenarios

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 1.

5   Conclusions

In this paper, demonstrated the impact of traffic generation with extensive micro-structure control as well as detailed corresponding documentation on researchers ability to evaluate and understand network intrusion detection models. We implemented and trained two state-of-the art detection models before extensively probing their behaviour and limitations when encountering different traffic types.

By using HTTP-traffic with congestion settings, we were quickly able to identify the inability of an LSTM-based classifier to handle traffic with significant retransmission rates, which enabled us to improve the model accordingly and increase detection performance by more than 2%. Similarly, the examination of projection consistency of a subspace-clustering method using traffic with artificially similar characteristics revealed an overly high sensitivity to flow interarrival times, while cluster-coherence could be increased significantly by identifying half-open connections that were dropped because of network failure as the source of overly dispersed traffic projections.

These results have encouraged us to perform more deep-going probing of data-driven network intrusion detection models. We believe that in combination with strong NID-dataset, extensive model validation and corresponding development with targeted traffic samples might hold the key to reduce false positives of detection models to an acceptable rate, as well as help models replicate detection rates in practical settings.

5.1   Difficulties and limitations

While the control of traffic micro-structures helps to understand models that perform on a packet- or connection-level, it does not replicate realistic network-wide temporal structures, such as port usage distributions or long-term temporal activity. The probing of models operating on aggregated, behavioural, or long-term features is therefore not effective, and variation in these quantities would have to be statistically estimated from other real-world traffic beforehand to allow our framework to emulate such behaviour reliably. Other datasets such as UGR-16 use this approach to fuse real-world and synthetic traffic and are currently better suited to build models of large-scale traffic structures.

Furthermore, while controlling traffic shaping factors artificially helps at identifying the limits and weak points of a model, it can exaggerate some characteristics in unrealistic ways and thus both affect the training phase of a model as well as tilt the actual detection performance of a model in either direction. Additionally, the artificial randomisation of traffic shaping factors can currently not generate the traffic diversity encountered in real-life traffic and thus only aid at exploring model limits extensively. The lack of realistic traffic heterogeneity however is at the moment significantly more pronounced in commonly used network intrusion datasets such as the CICIDS-17 dataset, where the vast majority

of successful FTP-transfers consist of a client downloading a single text file that contains the Wikipedia page for 'Encryption'.

5.2   Future work

Import of activity timeline The modelling and generation of computer network activity has been investigated extensively (citations?), and tools to automatically generate realistic network activity streams

we do not wish to address this topic here . Instead, our framework imports existing time-series of host flow activity to generate the corresponding communication. give more info on flow generation tools

We transform existing network flow series into an activity timeline by expand this. We end up with an activity timeline that contains a set of timestamps along with the corresponding scenario and the source and destination host.

References

1. M. Barre, A. Gehani, and V. Yegneswaran. Mining data provenance to detect advanced persistent threats. In *11th International Workshop on Theory and Practice of Provenance (TaPP 2019)*, 2019.
2. M. Gates and A. Warshavsky. Iperf Man Page. `https://linux.die.net/man/1/iperf`. Accessed: 2019-08-11.
3. R. Harang. Bridging the semantic gap: Human factors in anomaly-based intrusion detection systems. In *Network Science and Cybersecurity*, pages 15–37. Springer, 2014.
4. S. Hemminger et al. Network emulation with netem. In *Linux conf au*, pages 18–23, 2005.
5. X. Hou, L. Shen, K. Sun, and G. Qiu. Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141. IEEE, 2017.
6. 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.
7. 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.
8. K. Kolyshkin. Virtualization in linux. *White paper, OpenVZ*, 3:39, 2006.
9. H. Liu and B. Lang. Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, 9(20):4396, 2019.
10. Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.
11. I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.
12. 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.

13. 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.
14. 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.
15. H. Tahaei, F. Afifi, A. Asemi, F. Zaki, and N. B. Anuar. The rise of traffic classification in iot networks: A survey. *Journal of Network and Computer Applications*, 154:102538, 2020.
16. C. Walsworth, E. Aben, K. Claffy, and D. Andersen. The caida ucsd anonymized internet traces 2018,", 2018.
17. J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 26(1):56–65, 2019.