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

September 30, 2020

1 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.

2 Impact factors on traffic micro-structures

In order to provide extensive ground truth information that describe the traffic we generate, we first must understand what factors have influence on the traffic generation process. How do we verify that this list is more or less complete?

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.

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. [16] or for general application behaviour fingerprinting [15].

Transferred data The amount of transferred data obviously influences the overall packet numbers. Furthermore, the content of the data can potentially impact packet rates and sizes, such as shown by Biernacki [1] for streaming services.

Caching/Repetition effects Tools like cookies, website caching, DNS caching, known hosts in SSH, etc. can cause to parts of a communication being skipped and lead to differences in the captured traffic between initial and subsequent connections.

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. [9] for the QUIC/HTTP3 protocol.

Networking stack load TCP or IP queue filling due to other applications generating traffic can alter IATs of the traffic trace and affect available bandwidths. We should perform an experiment using iperf, once with TCP and once with UDP to quantify the effect.

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 incoming and outgoing traffic, which can again alter IATs and the overall duration of a connection. We should perform an experiment using *stress* to quantify the effect.

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

Other factors we are currently not considering:

User and scheduled activities The frequency with which a user or an application performs tasks 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.

TCP congestion management implementation Different versions of the TCP congestion manager exist on Windows and Linux such as TCP Reno/Tahoe, which can have minor influence on the traffic, as shown by Grieco et al. [3]. Existing implementations on a machine stay mostly constant, which is why we

also omit this variable at the moment. Implementations within a machine should stay constant.

Network configurations Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes, but like TCP implementations stay mostly constant for a host.

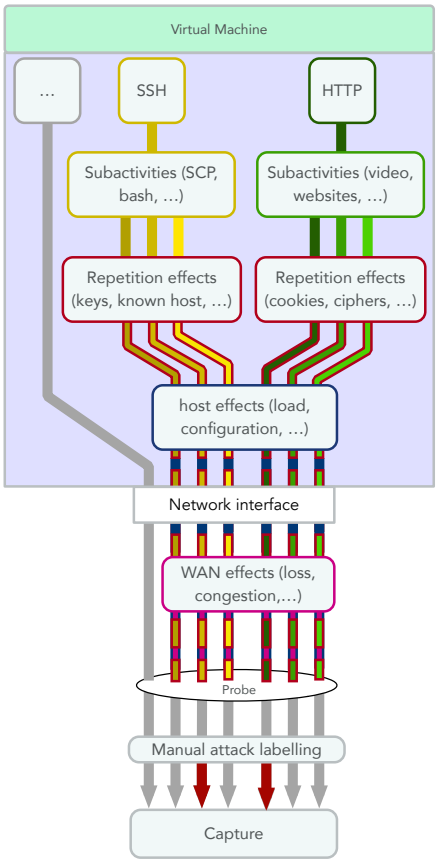
....

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

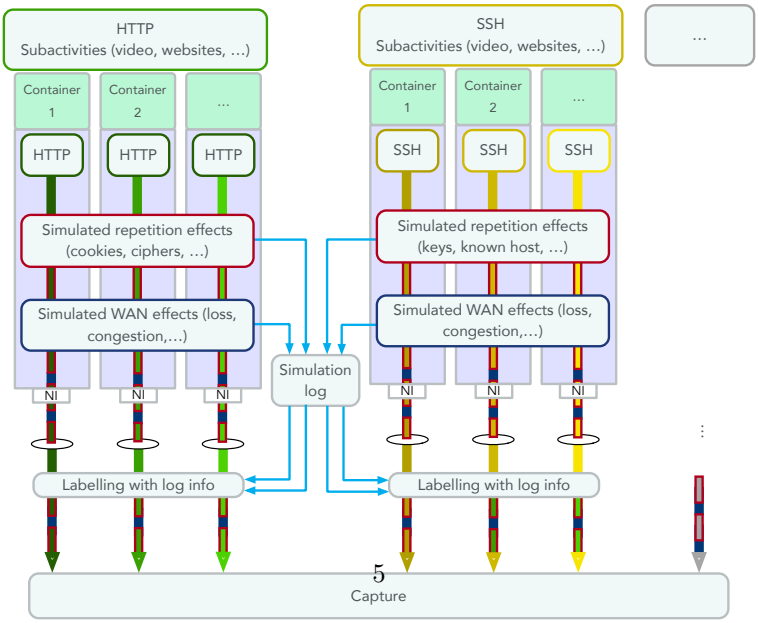
- Docker containers by design operate platform-independent without the need for muddled dependencies, and the separation of running containers from container images in contrast to VMs guarantees repeatable tasks without "code rot".
- We introduce adjustable degrees of input (files, passwords,...) and 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



(a) Traditional capture setup



(b) Our container-based capture setup

3 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 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 less generalised attack signatures. Already in 2010 Paxson and Sommer [14] 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 [4] in 2014 or by Liu et al. in 2019 [7].

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.

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 im-

prove the understanding of virtual machine-based intrusion detection and forensic memory analysis [2]. The inclusion of threat reports to create behavioral feature labels enriches the way executables are described to enhance malware modelling and detection [13].

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 [14].

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

3.1 Outline

Outline of the coming sections.

.....
.....
.....

4 Background

4.1 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.

4.2 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. A number of causes for these failings have been identified particularly for network-based methods by Sommer and Paxson [14] in 2010, which have been supported and partly extended by Harang [4] in 2014 or by Liu et al. in 2019 [7]:

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.

5 Goals and motivation

Our goal in 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. explain why focus on micro-structures and not larger structures We set out the following priorities:

Rich ground truth information Above all, our framework should 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 as well as outside factors such as network congestion or transmission failures to better facilitate understanding the effect of different traffic types on a detection system.

Reproducibility Our framework should be able to precisely reproduce any generated traffic events and corresponding network experiments in order to facilitate verifiable and comparable research.

Traffic realism on a packet level The framework should generate real, interactive network traffic that resembles common traffic structures on a packet-level. This is a necessary condition to provide a corresponding evaluation of NIDS systems with a plausible degree of scientific insight and relevance. Internet traffic overall is hard to model, which is why we narrow the scope to the generation of small-scale traffic scenarios that contain true interactions between software components in a realistic way. Furthermore, our framework should enable the inclusion of large-scale network activity profiles to leverage existing research in this area.

5.1 Motivation

While there are many potential use-cases that can benefit from a system like DetGen,

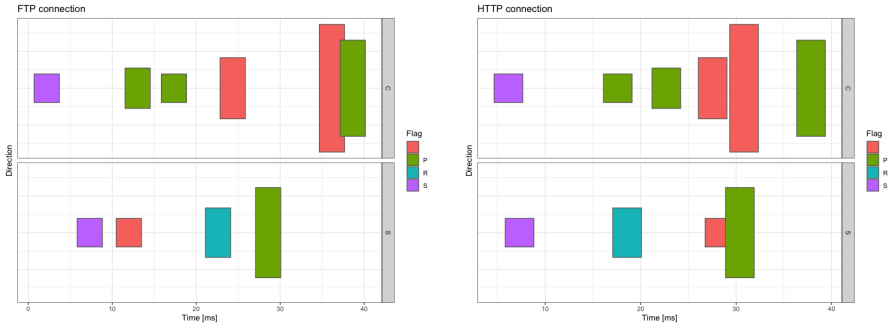
- *In-depth model evaluation*: Drawing on the extensive labelling of granular activities and reproducible traffic generation, researchers have new opportunities to examine the performance of an intrusion detection model in-depth. Packet-level structures and resulting false-positives can be better associated 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.

6 Impact factors on traffic metadata micro-structures

In order to provide extensive ground truth information that describe the traffic we generate, we first must understand what factors have influence on the traffic generation process. How do we verify that this list is more or less complete?

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.

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. [16] or for general application behaviour fingerprinting [15].



(a) Packet sequence in FTP connection (b) Packet sequence in HTTP connection

Transferred data The amount of transferred data obviously influences the overall packet numbers. Furthermore, the content of the data can potentially impact packet rates and sizes, such as shown by Biernacki [1] for streaming services.

Caching/Repetition effects Tools like cookies, website caching, DNS caching, known hosts in SSH, etc. can cause to parts of a communication being skipped and lead to differences in the captured traffic between initial and subsequent connections.

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.

insert number how often this is the case for traffic in CICIDS-17 dataset,explain procedure in appendix. Also insert image depicting the additional flows

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. [9] for the QUIC/HTTP3 protocol.

Networking stack load TCP or IP queue filling due to other applications generating traffic can alter IATs of the traffic trace and affect available bandwidths. We should perform an experiment using iperf, once with TCP and once with UDP to quantify the effect.

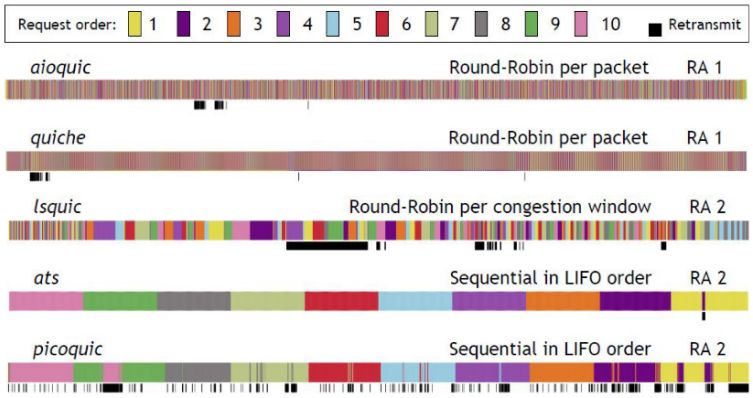


Figure 3: Comparison of QUIC connection request multiplexing for selected implementations, taken from [9]

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 incoming and outgoing traffic, which can again alter IATs and the overall duration of a connection. We should perform an experiment using *stress* to quantify the effect.

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

Other factors we are currently not considering:

User and scheduled activities The frequency with which a user or an application performs tasks 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.

TCP congestion management implementation Different versions of the TCP congestion manager exist on Windows and Linux such as TCP Reno/Tahoe, which can have minor influence on the traffic, as shown by Grieco et al. [3]. Existing implementations on a machine stay mostly constant, which is why we also omit this variable at the moment. Implementations within a machine should stay constant.

Network configurations Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes,

Contribution	How containers enable it	VM-based	Real traffic capture	Other framework
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	No knowledge about conducted activity	Technically possible, no attempts yet
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	-	Depends
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	-	Some work has been done to "replay" or artificially inject attacks
Data on attacks on container isolation	In need of existing scenarios for background data	No	No	No

Table 1: Contributions compared to existing solutions

but like TCP implementations stay mostly constant for a host.

7 DetGen Architecture

DetGen is a container-based emulator that we developed to enable repeatable, realistic, and flexible network experiments. DetGen extends the widely-used Mininet testbed insert citation by adding scenarios for benign and attack traffic generation, event labelling, and p

7.1 Design overview

The Detgen framework uses Mininet to create a network with a desired topology, along with virtual software switches, Ethernet links, routers, and firewalls. The network is then populated with containers ,which perform a variety of activities

for traffic generation. The conducted activities are composed of scripted *scenarios* (give examples here), but subject to a high degree of randomisation. The captured traffic events are labelled individually after the specific generating action.

7.2 Containerization

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 [6].

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 one-to-one correlation between containers and network traces allows us to produce labelled datasets with fully granular ground truths.

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.

The container network interface provides the connection between a network namespace and the container runtimes. We want to highlight that multiple containers can share on network interface, which enables us to generate traffic from multiple applications over one network address in order to emulate fully functional network hosts.

7.3 Scenario scripting

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 either a protocol, application or a series thereof. Examples may include an FTP interaction, a music streaming application and client, 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 **??**. 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.

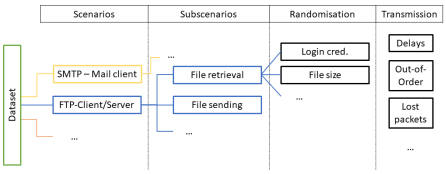


Figure 4: Visualization of the different levels at which traffic variation is introduced in DetGen.

Subscenarios

In contrast to scenarios, *subscenarios* provide a finer grain of control over the traffic to be generated, allowing the user to specify the manner in which a scenario should develop. The aim of having multiple subscenarios for each scenario is to explore the full breadth of a protocol or application's possible traffic behavior. 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 multiple subscenarios that cover this range of tasks.

Randomization within Subscenarios

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 their subscenarios 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 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.

Implementation process

7.4 Network creation and population

To enable communication between containers, we build our framework on top Mininet insert citation to create virtual networks with customizable topology. A topology can be passed to a topology-creation wrapper in matrix form, with diagonal values representing the type of device (switch, container, router, ...),

and off-diagonal indicating links. This allows the import of larger, automatically generated topologies from tools such as insert citation.

maybe something about subnets.

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.

Activity conduction

Attack traffic generation

A description of the capabilities and limitations of both metasploit and the metasploitable VM and the corresponding containers, which we will use.

7.5 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 [5] or UGR-16 [8] 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 [11] or the UNSW-16 [10] 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.

8 Fidelity confirmation experiments

This section is important to demonstrate that our data is valid and overcomes the difficulties entailed with synthetic data generation. Cordero et al. have proposed some more simple test that we can refer to first

Question to be answered: What requirements are there for the additional data, program logs and system logs, that we collect? Should we put less emphasise on these data sources in general if we are not able to perform these tests, and refer to them in future work? I am not aware of any papers that discuss these requirements in a similar way.

8.1 Traffic similarity metrics for determinism and realism evaluation

In this experiment, we compare our framework against a setup on a virtual machine to show that we provide deterministic control that is not achievable on a common setup. For that, we measure traffic similarity using a set of similarity metrics on traffic both from our and the comparison setup. The traffic is subject to varying degrees of influence from the above described impact factors. We show that when all factors are held constant, our traffic is much more similar than on a regular setup.

We should also demonstrate how difficult it is to match all flow events to specific events (I am still unsure how), and how often you get system background events (very straightforward using flow counting and port entropy).

Common setup: Network of VMs with each VM representing one host in a regular network. Multiple services with scheduled scripted activities running on a VM. Traffic captured at router or network interface.

- **Connection similarity** Comparison of flow summary statistics (IAT statistics, Packet size statistics, Flow counts, TCP window size statistics using cosine similarity and/or distribution Wilcoxon test, used in traffic classification

- **Flow level similarity** Number and order of flows as well as port numbers in a short window

8.1.1 First test: Determinism

8.1.2 Second test: Realistic diversity under all impact factors

Here, we apply the simulation of all impact factors simultaneously to our traffic and test if the traffic similarity scores are as divergent as for traffic from a real-world dataset (i.e. CAIDA) for two or three exemplary protocols, and we also compare it to the same protocols in the CICIDS-17 dataset. This will let us argue that we can achieve realistic levels of diversity (something criticised by Sommer-Paxson), and that our framework lets researchers explore higher levels of diversity than currents labelled solutions.

8.2 Explorating Artificial Delays

This section is already existing, we could potentially expand this. I think it is sufficient and analysing it more does not add much to the paper as the performance of TC netem is relatively well accepted. I think we could even move this section to the appendix.

9 Use-cases

9.1 Benefits of ground-truth labels

Possible title: **Exploring the effect of rare events to model performance**

Extensive ground-truth labels for our activities are arguably the most important contribution of the DetGen framework, so we should highlight their benefit most. Since ground-truth labels on attack data are existing in other datasets, we should emphasise the benefit of having labels for different activities. In my eyes, the most striking benefit arises for false-positive analysis, which we could then combine with showcasing the benefit of being able to generate different amounts of traffic for different activities.

Plan Implement the LSTM-model in the paper "An LSTM-Based Deep Learning Approach forClassifying Malicious Traffic at the Packet Level", train it on our data (both benign and attack traffic). Extract labels of traffic responsible for false-positives, show how much they are clustered around particular activities (potentially rare activities) compared to the overall traffic. Give potential reason for this. Generate a new dataset with increased amounts of the activities responsible for false positivies. Demonstrate that false-positives decrease.

9.2 Future work

References

[1] A. Biernacki. Analysis and modelling of traffic produced by adaptive http-based video. *Multimedia Tools and Applications*, 76(10):12347–12368, 2017.

[2] 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.

[3] L. A. Grieco and S. Mascolo. Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control. *ACM SIGCOMM Computer Communication Review*, 34(2):25–38, 2004.

[4] R. Harang. Bridging the semantic gap: Human factors in anomaly-based intrusion detection systems. In *Network Science and Cybersecurity*, pages 15–37. Springer, 2014.

[5] A. D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[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] 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.

[16] 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.