How to make traffic generation and monitoring at packet level more dynamic using containerisation

August 25, 2020

1 Introduction

Data-driven traffic analysis and attack detection is a centrepiece of network intrusion detection research, as well as many other fields citations. Realistic network traffic datasets allow engineers to build a better understanding of existing structures, motivate design choices from actual needs, and prove that proposed new systems can indeed work in practice. something to emphasise that packet traffic is important.

Unfortunately, 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. As a result, researchers are predominantly relying on synthetically generated data, which is currently either captured using scripted activities in a virtual or lab-environment, or generated using trained generative models of real-world traffic.

Existing network intrusion detection datasets are however collected in a static manner, unable to be modified, updated, or reproduced, citation and therefore can neither adapt to novel attacks or traffic evolutions, nor give researchers the flexibility to focus on particular traffic types. Furthermore, the current capture setups provide no insight in the data generation process, and it is mostly impossible to match events on the packet-level with conducted activities due to the capture setup. Existing testbeds and traffic generation frameworks are successful at emulating large-scale traffic characteristics citations, but fail to provide the necessary detail and realism for intrusion detection on a packet-level.

We propose a novel design paradigm that makes generation of network traffic and corresponding attacks significantly more flexible and offers a new level of insight and reproducibility for traffic micro-structures through the use of containerisation. By moving from virtual machines to containers, we enable the scalable, modular, and dynamic creation of network traffic datasets. Since containers can be arranged in complex settings with a few commands, it is a lot easier with containers to script a variety of network activities thus increase the heterogeneity and realism of the generated data.

This work provides the following contributions:

1. We present a novel network traffic generation framework that is designed to improve several shortcomings of current datasets and data generation frameworks for NIDS evaluation in the following aspects:
   (a) Ground truth labels documenting every conducted activities
   (b) Increased structural realism of the data on a packet-level
   (c) Tunable topology, traffic composition, and attack types
   (d) Capture of program logs and system calls for data fusion methods

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

2. We perform a number of experiments to demonstrate the fidelity to realism of the generated data.

3. We present a number of use-cases to demonstrate how the design of our framework can boost performance for ML-based network intrusion detection systems, in particular for false-positive analysis and effective model training.

1.1 Outline

Outline of the coming sections.

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

2 Goals and motivation

If we are to create *realistic* and *reproducible* intrusion detection experiments, then we need a platform with the following characteristics:

Traffic realism and reproducibility The system should generate real, interactive network traffic that resembles common traffic structures on a packet-level. The system should make it easy for researchers to verify results by enabling the duplication and running of network intrusion detection systems that are deployed within the platform.

Ground truth Activities generating traffic events should run in a reproducible manner to allow replication of traffic traces. Researchers should be able to trace the origin of traffic events to the precise generating activity.

Flexibility It should be easy to tune the composition of the generated. Similarly, it should be easy to create an experiment with any topology.

Attack diversity Intrusion detection models are dependent on attack data that represent specific attacks. The platform should therefore enable the execution or addition of a wide variety of attacks.

Low cost It should be inexpensive to duplicate an experiment, e.g. for students in a course.

Table 1 compares how well existing methods are realising these goals compared to our framework.

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 [9] 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 features from *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.

3 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

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	Background events (system activitiy, artifacts from earlier activity, ...), overlaying activities hard to separate	No knowledge about conducted activity	Technically possible, no attempts yet
Flexibility to regenerate customized datasets	Launching a network of containers is easy, modularity allows finer control of fine-grain activity	Difficult to regenerate data as set-up and launching of VM is less straightforward, might be addressed with MiniMega	No	Yes
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
Reproducability	Container state the same after repeated launches, isolation means that other task have little effect on	To a lesser degree, simultaneously conducted activity can affect each other	-	Depends
Fidelity to real traffic heterogeneity, structural richness	Not a lot except that scripting and randomising different scenarios is slightly easier	Potentially, but little attention has been paid	Yes	Not on a packet level yet
Data on attacks on container isolation	In need of existing scenarios for background data	No	No	No

Table 1: Contributions compared to existing solutions

3.1 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.

3.2 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 [7].

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.

3.3 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. [?]. 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

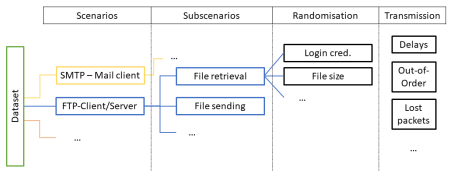

Figure 1: Visualization of the different levels at which traffic variation is introduced in DetGen.

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.

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 3.6.2, 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

3.4 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

3.5 Benefits resulting from design choices

3.6 Related work and existing datasets

3.6.1 Datasets

real: Los Alamos National Laboratory (LANL) University of Granada (UGR)

SANTA Only flows, manually labelled

CAIDA 2016

MAWI 2000

LITNET

synthetic:

CICIDS-17 UNSB-16

CIDDS-001 Python scripts are used for emulating benign user behaviour like browsing the web, sending and receiving emails, as well as exchanging files. To ensure as realistic user behaviour as possible, clients perform their activities with respect to an individual working schedule which considers working hours and lunch breaks.

NGIDS-DS Uses IXIA Perfect Storm, also collects host logs frameworks:

FLAME ID2T Use real traffic as input to add attack traffic GEN-ESIDS HTTP attack traffic after user-based attack description NDSec-1 - Also overlaying normal data with attack traffic

Moirai

A flow trace generator using graph-based traffic classification techniques extractstraffictemplatesfromrealnetw fic.Then,theirgeneratorusesthesetraffictemplatesinordertocreatenewsyntheticflow-basednetworktraffic

INSECS-DCS

Flow-based network traffic generation using Generative Adversarial Networks DoppelGANger

weak: PACGAN PCAPGAN SynGAN

surveys: A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems

On the generation of synthetic attacks

Docker data generation

weak on activity generation, only system calls: A dataset generator for next generation system call host intrusion detection systems Standardized container virtualization approach for collecting host intrusion detection data

Notes: However, currently available datasets lackreal-life characteristics of recent network traffic. ... IDS is unableto adapt to constant changes in networks (i.e. new nodes,changing traffic loads, changing topology, etc.). Networks areconstantly changing, for this reason depending solely on olddatasets doesn't help the advancement of IDS(Taxonomy ...)

Network traffic contains a vast amount of information about a network and its users which makes it notoriously difficult to release a comprehensive dataset without infringing the privacy and security rights of the network users [13]. In addition, the identification of malicious traffic in network traces is not straightforward and often requires a significant amount of manual labelling work. Currently, only four intrusion detection datasets exist that contain traffic from real networks, all of which are presented as anomimised network flows .

Rework!The most recent and notable real-world datasets have been released from the Los Alamos National Laboratory (LANL) in 2015 and 2017 [6, 14], and the University of Granada (UGR) in 2016 [8]. Both datasets contain network flow traffic data from a large number of hosts collected over multiple months, giving an accurate representation of medium- to large-scale structures in benign traffic. However, the amount of attack data is small and insufficient for accurate detection rate estimation. Furthermore, packet-level data is not available for both datasets.

Other real-world datasets, such as CAIDA 2016 [15] or MAWI 2000 [12], provide packet headers, but are unstructured and contain no labeled attack data at all.

To improve the lack of attack traffic in NIDS datasets, several artificially created datasets have been proposed. For this, a testbed of virtual machines is usually hosted in an enclosed environment to prevent any malicious code from spreading to other machines on other networks. To generate attack traffic, these machines are then subject to a selection of attack carried out by other machines in the environment. Benign traffic is generated using commercial traffic generators such as the *IXIA PerfectStorm tool*, or by scripting a selection of tasks for each machine. Synthetic datasets cover a smaller timeframe and contain traffic from a small number of hosts. Notable examples are the CIC-IDS 2017 dataset from the Canadian Institute for Cybersecurity [11], and the UNSW-NB 2015 dataset from the University of New South Wales [10]. Both datasets contain traffic from a variety of attacks, and are available as packet headers or as network flows with additional features crafted for machine-learning. While the benign traffic for the CIC-IDS 2017 data was generated using scripted tasks from a number of host profiles, the benign data for the UNSW-NB 2015 data is a mixture of captured real traffic from another subnet and traffic generated using a commercial traffic replicator.

Container networks have recently been adopted to conduct traffic generation experiments, such by Fujdiak et al. [3] who use containerized web servers to collect DoS-traffic. Furthermore, significant effort has been put into the creation of large-scale virtualization frameworks to provide automatized network testbeds [2, 1].

3.6.2 Problems in modern datasets

We can import here a lot from the existing paper, but add the following issues:

1. Invalid/Inconsistent traffic
2. Extensibility of the attacks
3. Lack of open-source implementation of attack traffic generation, which makes difficult to understand what exactly was detected
4. Lack of related data sources

3.7 Containerization with Docker and Mininet

A similar description on Containers and Docker as before, extended by a description of Mininet ......

......

......

3.7.1 Metasploit and Metasploitable containers

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

3.8 Dataset Requirements and benefits of our framework

Here, we can point to a set of requirements by Cordero et al. (https://arxiv.org/pdf/1905.00304.pdf) for generating synthetic datasets.

4 Design

Explain the general idea and benefit of using containers for traffic generation in comparison to regular testbeds, similar to the existing paper.

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

4.1 Modes of Operation

1. Stand-alone scenarios for traffic generation
   - Straightforward generation of traffic for the selected service
   - Easy to costumize and can be used as prior testing for modifying services in the other two options
2. Network-wide activity emulation
   - A topology is generated randomly with a set of hosts and corresponding services running on them
   - Scenarios are started and hooked to corresponding host network interfaces according to a launch script that emulates empirical activity measurements
   - These activity measurements come in the form of a timeline of events that can be drawn from a set of basic distributions that we provide or can be generated by third party such as the GAN in the Doppelganger paper (this way we can avoid being scrutinised for a lack of realism in this field).
3. Microservice activity emulation
   - In this mode, we set up a VM that hosts a number of service containers and emulates the situation of a typical microservice host
   - We run the same scenarios, but the client containers are located on another machine.
   - We collect system call logs from both the containers and the host
   - This is the least developed mode, but the additionally collected system calls give the most realistic picture in this state
   - It might be good to define and describe this setting already if we want to implement any container-specific attacks and models in the future

4.2 Scenarios and subscenarios

Here we describe the design process of the scenarios, just like in the previous paper.

4.2.1 Randomization

......

......

......

4.2.2 Network transmission

......

......

......

4.2.3 Implementation Process

......

......

......

4.2.4 Attack generation with Metasploit/Metasploitable

......

......

......

4.3 Network-emulation

Describe the design of how the topology is generated before the data capture is started, and how the scenarios are launched and stopped during the data capture.

4.3.1 Topology generation

......

......

......

4.3.2 Launch-script

......

......

......

4.3.3 Dataset coalescence

......

......

......

4.3.4 Activity timeline input

Here, we describe how the activity timeline in the launch script can come from unsophisticated distributions that we include, or from sophisticated 3rd party models such as Doppelganger. Since the modelling of computer activity is a field of ongoing research, it seems best to shift the responsibility for realistic models away by allowing this input.

4.4 Microservice-emulation mode

......

......

......

5 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.

5.1 Data correctness tests

This section is concerned with dataset defects, artifacts, or invalid data (inconsistent MTU etc.). These are very straightforward to test and should not take up much space.

5.2 Diversity tests

These tests, also from Cordero et al. quantify diversity via the entropy of different quantities such as IP diversity, Time-to-Live, Maximum-segment-size, Window size, ToS. I think we should keep this relatively short and omit comparison to other datasets since this is already done by Cordero et al.

5.3 Structural dataset dimensionality

Autoencoders are often used to compress non-deterministic, noisy data. Bahadur et al. have developed a procedure to estimate the ,,dimensionality" (to be understood as the complexity) of a dataset using variational autoencoders. I believe we can transfer this concept to sequence compression and estimate the overall complexity of connection sequences in our framework with both real traffic captures and existing network intrusion datasets.

Showing that our data is closer to real-world data would be a good test for "artificially predictable patterns", as described by Cordero et al., and go hand-in-hand with demonstrating the benefits of our framework for the training of deep-learning models. The importance of data that is less artificially predictable and closer to real-life traffic in terms of statistical variations lies in its suitability as a benchmark for detection rates, since less complex data is easier to train on and yields unrealistically high detection rates.

5.4 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.

6 Use-cases

6.1 Benefits of ground-truth labels/dynamic dataset generation

Possible title: **Dataset tuning to decrease false-positives**

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.

6.1.1 Benefits of structural richness

Possible title: **Harder benchmarks**

As described above, the importance of data that is less artificially predictable and closer to real-life traffic in terms of statistical variations lies in its suitability as a benchmark for detection rates. In particular, we want to demonstrate that our data functions is a more difficult and realistic benchmark that is less prone to inflating detection rates than existing datasets, something that is often a point of criticism for models evaluated on synthetic data.

To show that the training and detection is harder on our data, we could generate a dataset with similar attacks and services as the CICIDS-17 dataset, and train the above described LSTM model on both datasets. We could show that the training loss goes down more slowly on our data, as well as other metrics (increased validation loss –¿ overfitting etc.). We could then go ahead and show that the same attacks are detected easier by the same model in the CICIDS-17 data than in our data, concluding that it is a less realistic benchmark.

It would be good to also include a comparison with actual real-world traffic here to bolster our conclusion, but due to the lack of structured real-life datasets it is difficult to create a fair and scientific comparison.

6.2 Show utility of flexible topology

This is another possibility to demonstrate that the flexibility provided through containerisation allows for better benchmarking and more in-depth evaluation.

Methods aiming at modelling network structures are an established method for botnet and pivoting detection. Even though the network topology is a crucial variable in their training, their evaluation is to my knowledge only done on single static datasets such as the LANL-15.

My idea is that we generate about 10 datasets with different topologies, (especially different numbers of subnets and servers), and highlight the variation in prediction accuracy, i.e. the accuracy on one dataset is significantly higher/lower, and the average across the different datasets is a better indicator that eliminates the topology as a variable. We do not necessarily need to implement any attack traffic here since the modelling accuracy on benign traffic should be sufficiently quantifiable.

A suitable candidate for the evaluation would be the paper ”Link prediction in dynamic networks using randomdot product graphs”, which comes from people at Imperial College that I know. The model tries to give a probability for each connection to appear in a network, in order to spot connections between unlikely pairs as anomalous behaviour. I could ask the people for the implemented model so we do not have to do it ourselves, or we could even have a chat with them.

6.3 Customisable attack traffic by using metasploitable

Rob and I discussed that by using a combination of a metasploit-attack container and a metasploitable-victim container, we could generate and embed a significantly larger number of attack traffic types more efficiently than we are currently. Since we can attach the metasploitable-container to the network interface of regular containers, we can embed the implemented vulnerabilities very easily in a given scenario. Furthermore, since both containers are well maintained, we can keep the attack catalogue up-to-date.

I think the best use-case is to showcase the process of adding a new type of attack to the dataset, embedding it in a proper way to a given scenario, generating data from it. We could additionally implement a corresponding detection method, but I think this would not demonstrate anything.

6.4 Simple use-case for the microservice mode?

This use-case depends a lot on the state of the framework, but we could in principle show the benefits of the framework for the recently published model in "AppMine: Behavioral Analytics for Web Application Vulnerability Detection" on a more extensive dataset, since they only had very limited data available.

7 Conclusions

7.1 Difficulties and limitations

7.2 Future work

References

[1] S. Badiger, S. Baheti, and Y. Simmhan. Violet: A large-scale virtual environment for internet of things. In *European Conference on Parallel Processing*, pages 309–324. Springer, 2018.

[2] J. Crussell, J. Erickson, D. Fritz, and J. Floren. minimega v. 3.0. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2015.

[3] R. Fujdiak, V. Uher, P. Mlynek, P. Blazek, J. Slacik, V. Sedlacek, J. Misurec, M. Volkova, and P. Chmelar. Ip traffic generator using container virtualization technology. In *2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 1–6. IEEE, 2018.

[4] D. Inc. Docker. `https://www.docker.com/`. Accessed: 2019-08-11.

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

[6] A. D. Kent. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press, June 2015.

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

[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] 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.

[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, Nov. 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] C. Sony and K. Cho. Traffic data repository at the wide project. In *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, pages 263–270, 2000.

[13] A. Sperotto, R. Sadre, F. Van Vliet, and A. Pras. A labeled data set for flow-based intrusion detection. In *International Workshop on IP Operations and Management*, pages 39–50. Springer, 2009.

[14] M. J. M. Turcotte, A. D. Kent, and C. Hash. Unified Host and Network Data Set. *ArXiv e-prints*, Aug. 2017.

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