















# Dataset: Container Escape Detection for Edge Devices

James Pope<sup>1</sup>, Francesco Raimondo<sup>1</sup>, Vijay Kumar<sup>4</sup>, Ryan McConville<sup>1</sup>, Rob Piechocki<sup>1</sup>, George Oikonomou<sup>1</sup>, Thomas Pasquier<sup>2</sup>, Bo Luo<sup>3</sup>, Dan Howarth<sup>3</sup>, Ioannis Mavromatis<sup>4</sup>, Pietro Carnelli<sup>4</sup>, Adrian Sanchez-Mompo<sup>4</sup>, Theodoros Spyridopoulos<sup>4</sup>, and Aftab Khan<sup>4</sup>

<sup>1</sup>University of Bristol, Bristol, UK

<sup>2</sup>University of British Columbia, Vancouver, CA

<sup>3</sup>Smartia Ltd, Bristol, UK

<sup>4</sup>Toshiba Europe Limited, Bristol Research and Innovation Laboratory, Bristol, UK

james.pope@bristol.ac.uk

## ABSTRACT

Edge computing is rapidly changing the IoT-Cloud landscape. Various testbeds are now able to run multiple Docker-like containers developed and deployed by end-users on edge devices. However, this capability may allow an attacker to deploy a malicious container on the host and compromise it. This paper presents a dataset based on the Linux Auditing System, which contains malicious and benign container activity. We developed two malicious scenarios, a denial of service and a privilege escalation attack, where an adversary uses a container to compromise the edge device. Furthermore, we deployed benign user containers to run in parallel with the malicious containers. Container activity can be captured through the host system via system calls. Our time series auditd dataset contains partial labels for the benign and malicious related system calls. Generating the dataset is largely automated using a provided *AutoCES* framework. We also present a semi-supervised machine learning use case with the collected data to demonstrate its utility. The dataset and framework code are open-source and publicly available.

## CCS CONCEPTS

• **Information systems** → **Data mining**; • **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**.

## KEYWORDS

Datasets, Container Escape, Anomaly Detection, Cybersecurity

## ACM Reference Format:

James Pope, Francesco Raimondo, Vijay Kumar, Ryan McConville, Rob Piechocki, George Oikonomou, Thomas Pasquier, Bo Luo, Dan Howarth, Ioannis Mavromatis, Pietro Carnelli, Adrian Sanchez-Mompo, Theodoros Spyridopoulos, and Aftab Khan. 2021. Dataset: Container Escape Detection for Edge Devices. In *ACM DATA Workshop co-located with SenSys'21 and*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

*ACM DATA Workshop co-located with SenSys'21 and BuildSys'21, November 17, 2021, Coimbra, Portugal*

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-9097-2/21/11...\$15.00

<https://doi.org/10.1145/3485730.3494114>

*BuildSys'21, November 17, 2021, Coimbra, Portugal*. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3485730.3494114>

## 1 INTRODUCTION

The confluence of the Internet of Things (IoT) and Cloud Computing research has led to a canonical architecture consisting of three standard tiers: the *backend*, the *edge*, and the *endpoint* [1, 4, 18]. The endpoints are resource-constrained IoT devices with limited computing and energy resources and equipped with various sensors. The endpoints send their sensor data to an edge device, usually over LR-WPAN communication interfaces. The edge devices, typically part of an infrastructure (e.g., building, smart city), collect endpoint data and communicate to the backend. The edge layer, with more computational resources than the endpoints, provides end-users with the capability to run their applications with lower latency while communicating with the endpoints.

An example of such an architecture is the Urban Multi Wireless Broadband and IoT Testing for Local Authority and Industrial Applications (UMBRELLA) platform [6]. UMBRELLA allows the users to develop different applications and deploy them as containers on the edge nodes. The applications range from air quality monitoring, street light maintenance, robotics applications, private 5G for warehousing, logistics, and large-scale wireless testing. The UMBRELLA platform consists of 230 nodes, installed on public lampposts spread across 7.2km of South Gloucestershire roads and equipped with approx. 1500 sensors [9]. Developing applications running in containers on edge devices makes shipping, testing, and deploying easier. However, it also opens security and privacy issues where malicious applications running in a container may compromise the edge device and possibly the whole network. This paper describes the *AutoCES* framework and the data generated for an archetypal edge device with two container escape scenarios. The *AutoCES* framework supports experiments with varying auditing rules, container escape scenarios with configurable annotations, and workload background activity.

Figure 1 depicts the UMBRELLA hardware [9] used as the edge device. The authors were provided administrative access to the Umbrella nodes and configured them for the experiments as per the requirement. UMBRELLA uses Docker as the supported containerisation implementation [10]. The node (host system) runs *auditd* tool, part of the Linux Auditing System, to collect system calls between the kernel and applications, including those running in containers. We used awesome-docker containers [3] to generate benign applications using docker-compose container management

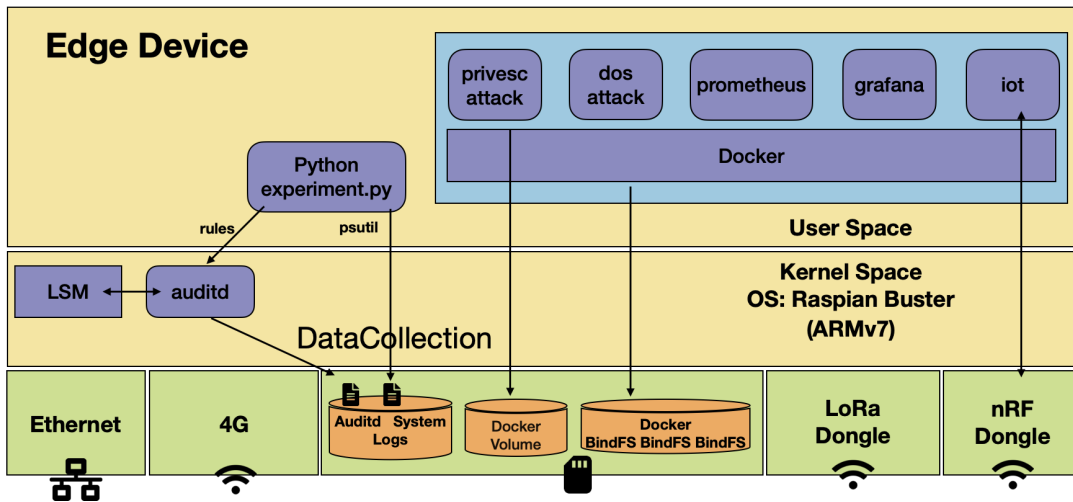


Figure 1: Edge Device (UMBRELLA Node) Overview

tool [2] and generate background container activity. We also developed a container to collect temperature sensor data from several remote IoT devices. We use custom scripts to simulate Denial of Service (DoS) attacks and privilege escalation attacks to generate malicious applications. Finally, we describe a use case with the dataset using a semi-supervised machine approach that could support anomaly detection. There are existing similar datasets. For example, Tien, et al. [16] provide a container-based dataset for anomaly detection. However, it lacks container escape scenarios and also lacks a framework for generating new datasets. To the best of our knowledge, there is not an equivalent kernel-based container escape dataset. The contributions of the paper are as follows:

- Annotated container-escape dataset.
- Open-source *AutoCES* framework for generating new container-escape datasets.

The paper is structured as follows: § 2 provide details about the hardware, framework, and how the data was collected. § 3 provides the container escape scenarios. § 4 provides the semi-supervised use-case. § 5 concludes our work.

## 2 DATA COLLECTION

This section provides details on the edge devices used for gathering logs of the different container escape scenarios, *AutoCES* framework, auditd configuration, and system logging details.

### 2.1 Edge Devices

The UMBRELLA platform is a large-scale real-world testbed for IoT applications [6]. The UMBRELLA edge device [9] includes a Raspberry Pi 3 Compute Module [14], a Jetson Nano [12], and various communications interfaces. The communication interfaces include multiple radios such as 802.15.4 (using nRF52480) and LoRa. Finally, the edge node connects to the backend using either fiber (about half of the network) or WiFi (the other half of the network - the fiber nodes act as WiFi gateways). We perform the dataset collection on the Raspberry Pi component. The UMBRELLA Raspberry Pi runs

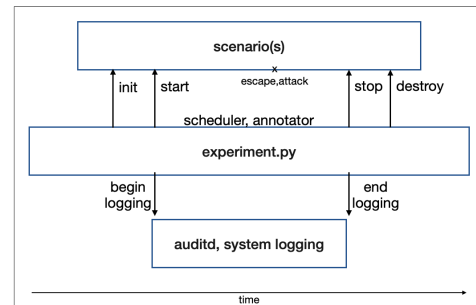


Figure 2: Framework/Experiment Timeline

Raspbian GNU/Linux 10 (buster), Linux version 5.4.83-v7+ on an ARMv7, 32-bit processor. The UMBRELLA is set up with Docker to run containers for various applications on the Edge. Figure 1 depicts the UMBRELLA environment.

We also provide a Raspbian Buster virtual machine (VM) and provide instructions for its configuration [13]. Though this is more artificial and lacks many of the peripherals of the UMBRELLA device, it allows users to generate their own datasets.

### 2.2 Framework

The *AutoCES* framework is a set of Python scripts that essentially automate running experiments to generate a dataset. It allows the user to configure the time duration of the experiments, when the attack starts and stops, configure the auditd and system logging. For instance, a user can run the experiment for 10 mins in which the attack starts at 5 mins, stops after 2 mins. In such a case, an annotation file would annotate the auditd logs as malicious for the duration of 5 (attack start time) - 7 (attack stop time). The *AutoCES* would label the rest as normal traffic. *AutoCES* allows having partial annotations that support container escape scenarios. Further, the framework configures the auditd rules and starts/stops it to provide efficient logging and automatically generates the log

files. The framework can be used on the UMBRELLA edge device and Raspian Buster VM (simulated edge device). The framework is structured to generically allow various scenarios to run during an experiment set up the logging. Figure 2 depicts the timeline of a scenario running. The *init* method allows the scenario to set up the initial configuration, such as activities that should not be included in the logging (e.g., starting a Docker container). The *init* also provides the scenario with a *scheduler* and *annotator* function to allow creating events and annotating the event time to the annotation log file.

### 2.3 Auditd Logging

The Auditd monitoring tool is based on the Linux Security Module (LSM). auditd allows access points to log kernel routines, including system calls, logins, password changes, file access, and others. The auditd tool is configured using a rules language and comes with a default set of pre-configured rules. For the dataset, we chose to use the following pre-configured rules files [15].

- 10-procmon.rules: monitor process executions
- 30-stig.rules: requirements set by Security Technical Implementation Guides (STIG)
- 31-privileged.rules: monitor use of privileged commands
- 33-docker.rules (custom): monitor Docker container activity (file-based)
- 41-containers.rules: log container events
- 43-module-load.rules: monitor kernel module insertion
- 71-networking.rules: monitor incoming/outgoing network connections
- 99-finalize.rules - Finalize (immutable)

We added a custom rules file, 33-docker.rules, for better monitoring Docker container activity. Additionally, when the framework starts, it finds the process identifier of the *docker-proxy* process and adds a rule to log its activity. A rule is also added to exclude the experiment process from the auditd log.

We note that we could have used a *log everything* rule. However, we found that with the minimal workload, the logging alone consumed between 30-50% of the CPU, and a more realistic scenario would use these recommended pre-configured rules.

### 2.4 System Logging

The Python psutil library (version 5.8) is used for logging system information such as CPU/memory usage and determining the logs collection overhead. Though auditd captures system calls, it does not capture system information such as CPU and disk utilization. To account for this, we also log *system* information. The idea to include system logging is to check the possibility of detecting malicious activity by correlating auditd and system logging data. Each system logging event has roughly 18 records. The following is a summary of a system logging event record. The dataset is produced with one system logging event per second though the framework allows this to be specified.

```
"timestamp": 1627509275.858,
"cpu_percent_percpu": ["..."],
"getloadavg": ["..."],
"virtual_memory": ["..."],
"disk_usage": ["..."],
```

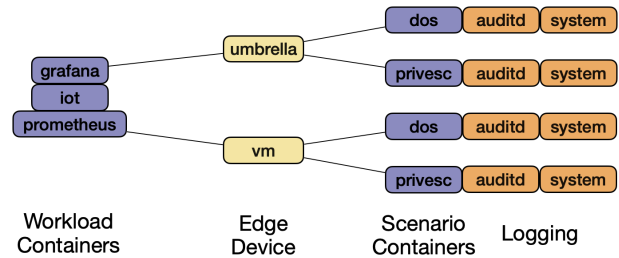


Figure 3: Experiment Configurations

```
"net_if_stats": ["..."],
...
```

## 3 SCENARIOS

The *AutoCES* allows the user to configure the experiments and execute them on the UMBRELLA edge device or VM. Currently the UMBRELLA testbed allows users the ability to run experiments on the platform [6]. However, the auditd utility would need to be enabled. The user can then decide which scenarios to use via the framework.

The user can configure the experiment based on four parameters: i) where to run the experiment *Edge*: umbrella or the Raspian Buster VM; ii) *Scenarios*: the scenarios to run (DoS, privilege escalation); iii) *Workload*: the benign background services running; iv) *Logging*: represents the logging system(s) to be enabled. Each experiment consists of the following.

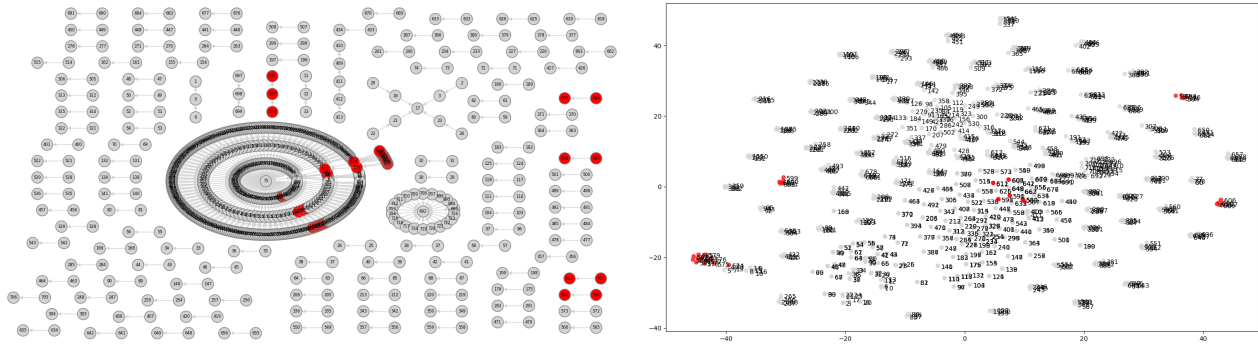
- Edge Device = {umbrella, vm}
- Scenario Containers = {dos, privesc}
- Workload Containers = {iot, prometheus, grafana}
- Logging = {auditd, system}

Every experiment includes all logging and workload options but can choose which edge and scenario. Figure 3 shows the four different experimental configurations. This section further describes the container escape scenarios and workload containers. The container escapes were partially derived from Wilhelm [17] and require running the containers in a reduced security mode (e.g. unconfined security option or privileged mode or added Linux capabilities). Though possibly preventable (e.g. confined apparmor, seccomp), misconfiguration always remains as a possibility. Regardless, the dataset remains valid providing information about container escape behaviour.

A single experiment consists of the auditd and system files, an annotation file (provides timestamps when the attack starts), and a rules file (derived from *auditctl -l*). Each experiment is 15 minutes long and contains one escape-attack that is randomly scheduled to occur within the interval. There are 64 experiments for each configuration shown in Figure 3. The dataset consists of these four configurations for a total of 256 experiments.

### 3.1 Container Escape with Denial of Service

This scenario first performs a container escape by mounting a filesystem (not a Docker volume) and writing to a shell script. The container then uses the *cgroup* notify/release mechanism to execute the shell on the host. The shell script currently consumes 2 CPU's



(a) Example Process Graph. Nodes are the process\_id and the edges are the relationships between parent and child. The nodes in red represents processes that were involved during the attack.

(b) Node Embedding for Example Process Graph. Node embeddings help to find the processes involved during attack phase as they cluster together well.

Figure 4: Auditd to Graph

for 20 seconds (this is configurable). The idea is to simulate a DoS type attack or an unauthorised usage (e.g. Bitcoin mining). This scenario runs in a *ubuntu* container. The framework launches the following Docker command to start the container.

```
docker run -d=true --name=ESCAPE_DOS --rm -it
--cap-add=SYS_ADMIN --security-opt
apparmor=unconfined ubuntu_shell_dos bash
```

### 3.2 Container Escape with Privilege Escalation

The privilege escalation scenario, denoted *privesc*, uses the volume feature of Docker to mount a file system over the host device. The container then writes to the */etc/sudoers.d* directory adding a file that modifies a user's privilege allowing them to execute *sudo* commands without a password (i.e. increased privileges). This scenario runs in an *alpine* container. Following is the Docker command used to start the container.

```
docker run -d=true --rm --name ESCAPE_PRIVESC
-v /:/privesc -it alpine_volume_privesc /bin/sh'
```

### 3.3 Workload Containers

The workload containers are intended to provide more *realistic* activity on the edge device in addition to the container escape activity. Instead of attempting to define *realistic*, we submit that an edge device would have a container interacting with remote endpoints collecting their data, storing, possibly analysing, and visualising the data.

The *iot* container communicates with three remote endpoints that send a temperature sensor reading every five seconds. The communication includes authenticated encryption and a simplified key transport protocol. We use the awesome-compose [3] setup for the *prometheus* container providing time-series data storage and the *grafana* container providing a web interface for the visualization of data.

Notable deficiencies are that the *iot* container should be feeding data to the *prometheus* container and there should be some additional workload applied to the *grafana* container to simulate

user interaction. These are left as future work though we note the framework would also support this more integrated setup.

## 4 SEMI-SUPERVISED MACHINE LEARNING USE CASE

We demonstrate the utility of the dataset with a use case to assist in data labelling. The auditd data is first converted into a *process graph*, which depicts the parent-child relationship between processes. There are certainly other conversions. For example, Sadegh, et al., [11] convert auditd data into a provenance graph to assist in hunting for cyber threats. A node embedding technique [8] then takes the graph's nodes and automatically transforms them into vectors suitable for subsequent processing, including classification.

Figure 4a shows the process graph converted from a DoS experiment run on the UMBRELLA. The red nodes are labelled as *malicious* and the remaining nodes are labelled as *benign*. The red nodes are processes active during the annotated attack interval. This mostly includes system calls that are part of the escape/attack but also includes some innocuous system calls. Each graph's node is then converted into a vector in 128-dimensions using the graph information only. We used the node2vec [8] implementation in the StellarGraph library [5] with 100 random walks per node. Figure 4b shows the node's vectors visualised using t-SNE to project onto two dimensions. The results show that the node's vector representation is similar to the original graph and has clear clusters of the processes involved in the DoS. The node's vectors were then used to train a logistic regression classifier that achieved 97% F1 score to predict whether the process was *benign* or *malicious*. Only 10% of the data was used for training while the remaining 90% was used for testing. Thus, a small set of training instances could help to annotate the remaining instances in a semi-supervised situation. These results provide evidence for the utility of the dataset. As previously mentioned, there are certainly other use cases, notably, extracting features from the auditd logs along with the time interval based annotations to train an autoencoder for anomaly detection.

## 5 CONCLUSION

The dataset fills an important gap for security researchers and practitioners dealing with edge container behaviour. The dataset includes novel, partially annotated container escape information with reasonable background activity. The dataset was generated from an exemplary edge device and also from a more artificial virtual machine. The presented framework automates experiments and allows users to easily generate new datasets. We also showed the utility of the dataset with a supervised machine learning use case. The paper also has limitations. Firstly, the dataset currently only provides partial annotations with time intervals, including benign and malicious system calls. For instance, the time interval marked malicious will also contain benign system calls. Secondly, the dataset currently has only two container escape scenarios (DoS and Privilege Escalation). Container escapes and hardening are vast topics [7]. However, we chose these two attacks as they are prominent approaches and were reasonably easy to configure. Finally, our selection of benign background container activity and auditing rules may not be sufficient for other use cases.

## ACKNOWLEDGMENTS

This work was supported by UK Research and Innovation, Innovate UK [grant number 53707].

## REFERENCES

- [1] Fei Chen, Duming Luo, Tao Xiang, Ping Chen, Junfeng Fan, and Hong-Linh Truong. 2021. IoT Cloud Security Review: A Case Study Approach Using Emerging Consumer-Oriented Applications. *ACM Comput. Surv.* 54, 4, Article 75 (May 2021), 36 pages. <https://doi.org/10.1145/3447625>
- [2] Docker Community. [n.d.]. Docker Compose. <https://github.com/docker/compose>.
- [3] Docker Compose Community. 2021. Awesome Compose. <https://github.com/docker/awesome-compose>. Accessed: 2021-09-10.
- [4] Ademir F. da Silva, Ricardo L. Ohta, Marcelo N. dos Santos, and Alecio P.D. Binotto. 2016. A Cloud-based Architecture for the Internet of Things targeting Industrial Devices Remote Monitoring and Control. *IFAC-PapersOnLine* 49, 30 (2016), 108–113. <https://doi.org/10.1016/j.ifacol.2016.11.137> 4th IFAC Symposium on Telematics Applications TA 2016.
- [5] CSIRO's Data61. 2018. StellarGraph Machine Learning Library. <https://github.com/stellargraph/stellargraph>.
- [6] Tim Farnham, Simon Jones, Adnan Aijaz, Yichao Jin, Ioannis Mavromatis, Usman Raza, Anthony Portelli, Aleksandar Stanoev, and Mahesh Sooriyabandara. 2021. UMBRELLA Collaborative Robotics Testbed and IoT Platform. In *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*. 1–7. <https://doi.org/10.1109/CCNC49032.2021.9369615>
- [7] FrankSpierings. 2021. Linux Container Escapes and Hardening. <https://gist.github.com/FrankSpierings/5c79523ba693aaa38bc963083f48456c#escaping>. Accessed: 2021-09-06.
- [8] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (San Francisco, California, USA) (KDD '16)*. Association for Computing Machinery, New York, NY, USA, 855–864. <https://doi.org/10.1145/2939672.2939754>
- [9] BRIL Toshiba Europe Ltd. 2021. UMBRELLA Node. <https://www.umbrellaiot.com/what-is-umbrella/umbrella-node/>. Accessed: 2021-09-06.
- [10] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 239, Article 2 (March 2014).
- [11] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V.N. Venkatakrisnan. 2019. POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (London, United Kingdom) (CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1795–1812. <https://doi.org/10.1145/3319535.3363217>
- [12] NVIDIA. 2021. Jetson Nano Developer Kit. <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>. Accessed: 2021-09-06.
- [13] James Pope and Francesco Raimondo. 2021. Container Escape Detection for Edge Devices. <https://github.com/jpope8/container-escape-dataset>. Accessed: 2021-09-10.
- [14] RaspberryPi. 2021. Compute Module 3+. <https://www.raspberrypi.org/products/compute-module-3-plus/>. Accessed: 2021-09-06.
- [15] RedHat. 2021. Defining Audit Rules. [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/security\\_guide/sec-defining\\_audit\\_rules\\_and\\_controls](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-defining_audit_rules_and_controls).
- [16] Chin-Wei Tien, Tse-Yung Huang, Chia-Wei Tien, Ting-Chun Huang, and Sy-Yen Kuo. 2019. KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches. *Engineering Reports* 1, 5 (2019), e12080. <https://doi.org/10.1002/eng2.12080> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/eng2.12080>
- [17] Felix Wilhelm. 2019. Understanding Docker container escapes. <https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/>. Accessed: 2021-09-10, [https://twitter.com/\\_fel1x](https://twitter.com/_fel1x).
- [18] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. 2014. Internet of Things for Smart Cities. *IEEE Internet of Things Journal* 1, 1 (2014), 22–32. <https://doi.org/10.1109/JIOT.2014.2306328>