A Crowdsensing Intrusion Detection Dataset For Decentralized Federated Learning Models

Chao Feng[1,*], Alberto Huertas Celdran[1,2], Jing Han[1], Heqing Ren[1], Xi Cheng[1], Zien Zeng[1], Lucas Krauter[1], Gérôme Bovet[3], and Burkhard Stiller[1]

[1]Communication Systems Group, Department of Informatics, University of Zurich UZH, CH–8050 Zürich, Switzerland
[2]Department of Information and Communications Engineering, University of Murcia, 30100–Murcia
[3]Cyber-Defence Campus, armasuisse Science & Technology, CH–3602 Thun, Switzerland
[*]cfeng@ifi.uzh.ch

ABSTRACT

This paper introduces a dataset and experimental study for decentralized federated learning (DFL) applied to IoT crowdsensing malware detection. The dataset comprises behavioral records from benign and eight malware families. A total of 21,582,484 original records were collected from system calls, file system activities, resource usage, kernel events, input/output events, and network records. These records were aggregated into 30-second windows, resulting in 342,106 features used for model training and evaluation. Experiments on the DFL platform compare traditional machine learning (ML), centralized federated learning (CFL), and DFL across different node counts, topologies, and data distributions. Results show that DFL maintains competitive performance while preserving data locality, outperforming CFL in most settings. This dataset provides a solid foundation for studying the security of IoT crowdsensing environments.

Background & Summary

The Internet of Things (IoT) has permeated nearly every aspect of the physical world, from smart homes to industrial automation[1]. These devices enable a wide variety of sensing, actuation, and automation applications, but their large-scale deployment also introduces new challenges for monitoring, managing, and securing such heterogeneous and resource-constrained networks[2].

One particularly relevant IoT application scenario is crowdsensing, where a large number of distributed, heterogeneous devices collaboratively collect and contribute measurements or observations about their local environments[3]. Crowdsensing leverages the collective sensing capability of many participants to enable large-scale monitoring without relying on a centralized infrastructure. This paradigm is appealing for applications such as traffic monitoring, environmental sensing, and anomaly detection, especially in privacy-sensitive and bandwidth-constrained settings. However, the distributed and open nature of crowdsensing networks also makes them susceptible to security threats[2].

In particular, crowdsensing devices are often targeted by malware and intrusion attacks, which undermine the availability, integrity, and functionality of the system[1]. Detecting such intrusions and identifying compromised devices has therefore become a critical research challenge, especially in large-scale, dynamic crowdsensing deployments. Machine learning (ML)-based anomaly detection methods have been widely adopted for this task, as they can analyze multidimensional behavioral data from devices to identify deviations indicative of compromise. However, conventional ML pipelines typically assume that all behavioral data can be transmitted to a centralized server for training, which raises privacy concerns and creates vulnerabilities by concentrating data and computation in a single location[4].

Federated learning (FL) has emerged as a privacy-preserving alternative to centralized ML for collaborative intrusion detection[5]. In FL, each device retains its local data and performs on-device training, sharing only model updates with a coordinating server that aggregates them into a global model, called centralized FL (CFL). This architecture mitigates privacy risks while benefiting from distributed knowledge[6]. Nevertheless, the reliance on a central server introduces a single point of failure, potential scalability bottlenecks, and vulnerability to targeted attacks, limitations that are particularly pronounced in large, dynamic crowdsensing scenarios.

To address these issues, decentralized FL (DFL) has been proposed. By removing the client-server hierarchy and adopting peer-to-peer topologies, DFL enables each node to act both as a learner and as an aggregator[4]. This fully decentralized design aligns naturally with the characteristics of crowdsensing, enhancing scalability, robustness, and privacy in collaborative intrusion detection for IoT environments.

However, despite the growing interest in DFL as a promising framework for IoT intrusion detection, the lack of suitable

benchmark datasets has become a key impediment to research progress. Several benchmark datasets have been developed for intrusion detection research, most notably in the context of centralized ML. The NSL-KDD dataset[7] is a widely used successor to the KDD'99 benchmark, providing labeled network connection records. However, it is outdated and does not reflect modern IoT-specific traffic or attack patterns. The CICIDS2017 dataset[8] addresses some of these limitations by offering flow-based network traffic data with a wider variety of attacks, but it still assumes a centralized data collection scenario and lacks device-level heterogeneity. To specifically target IoT environments, the Bot-IoT dataset[9] was proposed, simulating IoT device traffic mixed with botnet attacks in a testbed environment. While Bot-IoT captures some IoT-specific characteristics, its synthetic nature and centrally collected data limit its realism for decentralized learning evaluations. Similarly, the TON_IoT dataset[10] provides telemetry, operating system, and network data from IoT and industrial control systems. However, the setup of TON_IoT involves homogeneous devices and centrally collected data, lacking client-level heterogeneity and device-specific data distributions that are essential for evaluating DFL in realistic IoT settings.

These limitations highlight the need for a realistic, distributed, and well-documented dataset specifically tailored to evaluating DFL frameworks in crowdsensing-based IoT intrusion detection tasks. Such a dataset would enable systematic research on privacy-preserving, scalable, and robust anomaly detection methods under realistic deployment conditions.

To address the lack of datasets and benchmarks for evaluating DFL in crowdsensing-based IoT intrusion detection, a dataset was constructed and validated through experimental evaluation. Therefore, the main contributions of this study are as follows: (1) the behavioral dimensions of IoT devices affected by malware attacks were analyzed, identifying network activity, input/output operations, file system access, resource usage, system calls, and kernel events as key data to monitor. This informed the design of the experimental platform and the selection of features; (2) an experimental setup consisting of six Raspberry Pi 3 and two Raspberry Pi 4 devices connected to the ElectroSense platform was implemented. Data were collected under eight malware attacks and a benign state, yielding 288 hours of monitoring and over 21,582,484 records. The dataset was cleaned, normalized, and processed to ensure suitability for machine learning while reducing overfitting risk; (3) a DFL pipelines was developed within the Nebular framework to demonstrate the dataset's applicability to DFL-based intrusion detection. A multilayer perceptron (MLP) model was used for malware classification. The DFL models were trained and evaluated under multiple DFL topologies, and their performance was measured using standard classification metrics, confirming the dataset's utility for crowdsensing scenarios.

Methods

This section describes the experimental design, data collection procedures, and data processing steps conducted in this work. The design of the experimental platform, the selection of behavioral dimensions to monitor, and the construction of the dataset are detailed to ensure reproducibility. The overall workflow of the proposed dataset construction and validation process is illustrated in Figure 1. It shows the data collection at the device level, the subsequent data processing and feature engineering steps, and the DFL training and aggregation performed collaboratively by IoT devices.

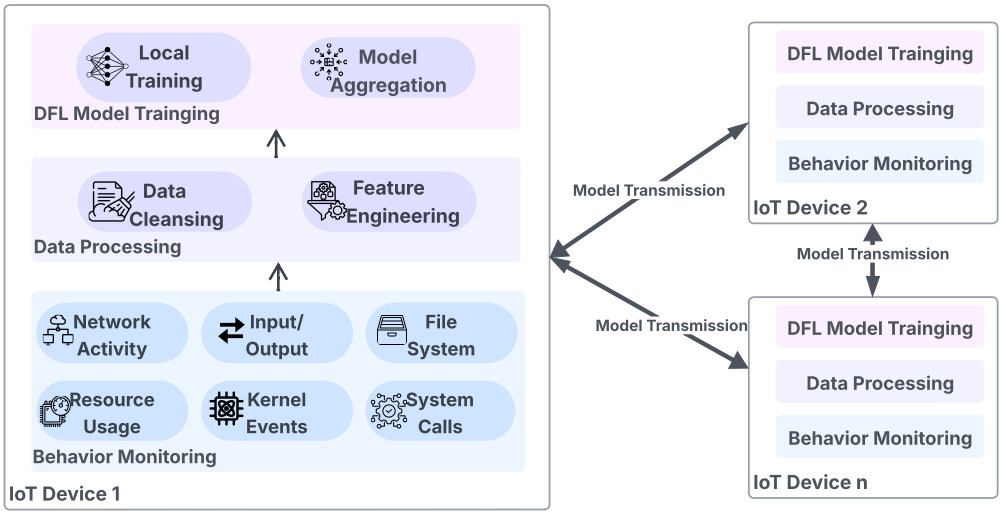


Figure 1. Workflow of the proposed dataset construction

Behavior Selection

The behavior of IoT devices under malware attacks is complex and multifaceted, involving a wide range of runtime activities and system interactions. Monitoring and recording all possible behaviors in their entirety is impractical due to the high overhead and limited relevance of many behaviors to malware detection. Therefore, it is necessary to select a subset of behavioral dimensions that are most effective for identifying malware, induced anomalies.

Table 1. Behavioral dimensions affected by different malware types in IoT environments.

Malware type	Network	I/O	File system	Resource usage	System call	Kernel events
Botnets	Ilavarasan et al.[11] Meidan et al.[12] Koroniotis et al.[13]			Bezerra et al.[14]	de Costa et al.[15] Martinelli et al.[16] Saracino et al.[17]	Ilavarasan et al.[11] Martinelli et al.[16]
Backdoors	Zhang et al.[18] Huertas et al.[1]		Huertas et al.[1]	Huertas et al.[1]	Canzanese et al.[19]	Huertas et al.[1]
Rootkits	Hoglund & Butler[20]	Kruegel et al.[21] Baliga et al.[22]	Nick et al.[23]	Nick et al.[23] Baliga et al.[24]	Nick et al.[23] Carbone et al.[25]	
Ransomware	Kok et al.[26]	Kok et al.[26]	Kok et al.[26]	Kok et al.[26]	Martinelli et al.[16] Kok et al.[26]	Martinelli et al.[16] Kok et al.[26]
Coinminer	Barbhuiya et al.[27] Tanana et al.[28]			Barbhuiya et al.[27] Tanana et al.[28]	Tanana et al.[28]	Huertas et al.[1]

To determine which behavioral dimensions are most relevant for monitoring in IoT malware detection, an analysis was conducted of prior studies on malware categories targeting IoT devices. Table 1 summarizes how different types of malware, botnets, backdoors, rootkits, ransomware, and coinminers, affect various behavioral sources. As shown in the table, different malware families can influence a range of runtime behaviors, including network activity, input/output operations, file system access, resource usage, system calls, and kernel events. Botnets, for example, exhibit distinct patterns in network traffic and system calls, while rootkits manipulate kernel events and file system operations. Ransomware and coinminers are associated with increased resource usage and modifications to system files and kernel-level events.

Based on this analysis, six behavioral dimensions were selected for monitoring: resource usage, kernel events, system calls, network activity, input/output operations, and file system access. Monitoring these dimensions is intended to capture the diverse effects of malware on IoT devices, and to enable the construction of a dataset that supports dynamic anomaly detection and malware classification.

Device Behavior Monitoring

The experimental platform was implemented using eight Raspberry Pi devices, comprising six Raspberry Pi 3 and two Raspberry Pi 4 boards. Each device was equipped with a software-defined radio (SDR) kit, serving as the sensing infrastructure. The devices were configured with either 32GB or 64GB SD cards, and ran on an ARM-based CPU architecture using the ElectroSense sensor image to enable data acquisition and processing.

These eight devices were deployed to collect behavioral data under two conditions: a benign (normal) operational state and eight distinct malware attack scenarios. The selected malware covered five major families, botnet, backdoor, rootkit, ransomware, and coinminer. Specifically, the botnet sample used was Bashlite[29]; the backdoor category included HttpBackdoor[30], Backdoor[31], and TheTick[32]; the rootkit category included Beurk[33] and Bdvl[34]; the ransomware sample was Ransomware-PoC[35]; and the coinminer sample was XMRig[36]. All malware samples were open-source implementations retrieved from their respective public repositories. Custom scripts were created to execute each malware sample continuously on the devices to simulate realistic attack behavior and ensure consistent data collection.

Each data collection session lasted four hours, and each malware scenario was executed separately. In total, 288 hours of behavioral data were recorded. To simulate realistic attacker behavior and maximize the observable impact of each malware sample, dedicated execution scripts were implemented. These scripts initiated infinite loops to generate continuous malicious activity, including file creation, modification, deletion, directory listing, file encryption, and other operations depending on the malware type. This setup ensured a sustained workload on the monitored devices and allowed for consistent data acquisition across all experimental conditions.

Each device monitored its behavior using six modules, collecting data from the following dimensions: resource usage, kernel events, system calls, network activity, input/output operations, and file system access.

Resource Usage

This module monitored device-level resource utilization, including CPU and memory usage, disk utilization, network throughput, page faults, cache misses, and hardware performance counters. Metrics were sampled every 5 seconds using standard system

utilities and hardware counters exposed via `perf`. The collected features provided a high-level view of overall system load and bottlenecks during both benign and malicious operation.

Kernel Events

This module recorded fine-grained kernel-level tracepoints at 5-second intervals, using `perf` to log a predefined set of kernel events indicative of I/O operations, memory management, process scheduling, signal handling, network stack activity, and file system writeback. The monitored events covered a wide range of subsystems, including `block`, `jbd2`, `kmem`, `sched`, `writeback`, `irq`, `net`, `signal`, and others. These events provide a low-level view of device behavior during benign and malicious operation, enabling analysis of how malware affects kernel state transitions and resource usage.

System Calls

This module monitored the sequence of system calls executed by processes running on the device. Sampling occurred every 10 seconds by using `perf`, recording the count and type of system calls observed during the window. The collected data included both user-space initiated calls and kernel-level service calls, enabling analysis of process behavior changes indicative of malware activity.

Network Activity

This module captured TCP and UDP traffic on the `eth0` interface of each device using the Python-based `Scapy` library. For each observed packet, the timestamp, protocol type, source and destination IP addresses, source and destination ports, and packet length were recorded. Data were aggregated in 5-second windows to provide time-resolved network flow characteristics under different scenarios.

Input/output Operations

This module monitored block-level input/output activity and the entropy of modified files. Block activity was recorded using `iostat`, capturing metrics such as reads, writes, and I/O utilization at 5-second intervals. For each file modification event detected via `inotifywait`, the Shannon entropy of the first 100 bytes of the file content was calculated as:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{1}$$

where p_i is the relative frequency of each byte value in the sample. This metric reflects the randomness of file content and is commonly used to detect packed or encrypted malware payloads.

File System

This module logged file system-level operations by recording `perf` events related to `ext4`, `block`, `jbd2`, and writeback subsystems. Events such as file creation, deletion, modification, and journaling activity were tracked. Data were collected continuously and aggregated into 5-second windows to capture fine-grained changes in file system behavior during normal and malicious execution.

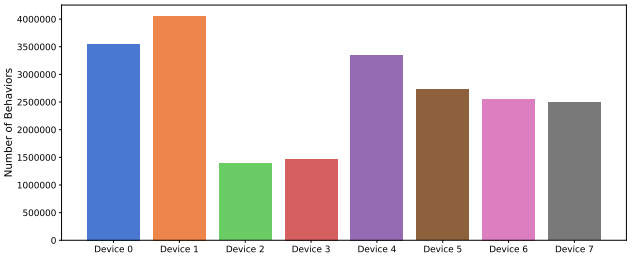
Data Processing

This subsection describes the procedures applied to process the collected raw behavioral data into a structured and clean dataset suitable for training and evaluation. The processing pipeline includes three main steps: cleansing the data to remove irrelevant, missing, or redundant information; analyzing and visualizing the distributions of collected features to understand their characteristics; and transforming the raw monitoring outputs into meaningful statistical features.

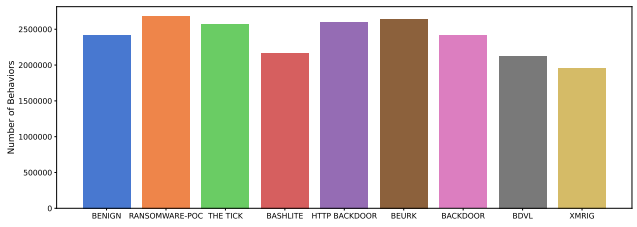
Data Cleansing

After integrating the behavioral data from six monitoring modules across eight IoT devices, a data cleansing step was performed to ensure the quality of the dataset and prepare it for subsequent feature selection and model training. The cleansing process consisted of the following steps:

- **Elimination of Useless Features**: Columns deemed irrelevant for learning tasks, such as `time_mean`, `seconds_mean`, `connectivity_mean`, and `timestamp`, were removed from the dataset.
- **Missing Value Imputation**: Missing entries in the dataset were filled with zeros to maintain a consistent feature space without introducing NaN values.
- **Elimination of Constant Features**: Features with zero variance (i.e., constant across all samples) were identified and removed, as they provide no discriminative power for the learning models.



(a) Number of collected behavioral records per IoT device.



(b) Number of collected behavioral records per malware and benign condition.

Figure 2. Distribution of collected behavioral records across devices and conditions.

Data Records

A total of 21,582,484 behavioral records were collected from eight Raspberry Pi devices equipped with SDR sensors. The dataset is stored in CSV format and organized hierarchically to reflect the experimental configuration. At the top level, data are partitioned by device, with a separate folder corresponding to each of the eight devices. Within each device folder, the data are further divided into subfolders by label, covering one benign condition and eight malware families: Bashlite, HttpBackdoor, Backdoor, TheTick, Beurk, Bdvl, Ransomware-PoC, and XMRig.

Figure 2a illustrates the distribution of collected behavioral records across the eight IoT devices. The data distribution among devices shows that Device 2 and Device 3, which correspond to Raspberry Pi 4 hardware, generated fewer records compared to the Raspberry Pi 3 devices. This is likely due to their lower utilization or specific roles during data collection.

Regarding the distribution of behavior labels, the dataset covers benign activities as well as eight distinct malware, as shown in Figure 2b . The number of samples per category is relatively balanced, ensuring that the dataset is suitable for training and evaluating classification models without significant class imbalance.

The complete dataset is available from the Science Data Bank[37], comprising CSV files organized by device and label, with each file containing preprocessed behavioral records and the corresponding extracted features.

Table 2. Extracted features from network traffic.

Network Feature	Description
`PacketCount`	Number of packets in the window
`TotalLength`	Sum of packet lengths
`AverageLength`	Mean packet length
`MedianLength`	Median packet length
`MinLength`	Minimum packet length
`MaxLength`	Maximum packet length
`VarianceLength`	Variance of packet lengths
`DifferentSourcePorts`	Number of unique source ports
`DifferentDestPorts`	Number of unique destination ports
`TcpPacketCount`	Number of TCP packets
`UdpPacketCount`	Number of UDP packets
`TcpUdpProtocolRatio`	Ratio of TCP to UDP packets
`MeanInterPacketInterval`	Mean time between consecutive packets
`VarianceInterPacketInterval`	Variance of inter-packet intervals
`MinInterPacketInterval`	Minimum inter-packet interval
`MaxInterPacketInterval`	Maximum inter-packet interval
`FirstDerivativeInterPacketInterval`	First derivative of inter-packet intervals
`SecondDerivativeInterPacketInterval`	Second derivative of inter-packet intervals
`AverageBandwidth`	Mean bandwidth consumption
`VarianceBandwidth`	Variance in bandwidth usage
`MinBandwidth`	Minimum bandwidth observed
`MaxBandwidth`	Maximum bandwidth observed
`DifferentDestIPs`	Number of unique destination IPs

Feature Engineering

This section describes the methods used to transform the raw behavioral monitoring data into structured, ML-ready feature representations.

Although some of the monitoring modules already produced encoded metrics (e.g., counts of kernel or resource events per interval), other modules, including input/output operations, network activity, and system calls, required additional processing to extract features. To ensure consistency, all features were aggregated over fixed time windows of 30 seconds.

- **Input/output operations.** When a file was created or modified, its entropy value was calculated from the first 100 bytes of its content to capture randomness indicative of encryption. To quantify suspicious activity over time, the recorded entropy values were aggregated into 30-second windows. Within each window, the number of files with entropy values greater than or equal to 6 was computed as the feature `entropy_file_count`.
- **Network Activity.** Network traffic was captured at the packet level, recording protocol, source and destination IP addresses and ports, and packet lengths. These data were aggregated into 30-second windows. From each window, 23 features were computed, as shown in Table 2. These features characterize both the volume and the structure of network communication.
- **System Calls.** System call traces were collected by monitoring kernel interactions within each 30-second window. The Bag-of-Words (BoW) encoding was applied, producing frequency vectors representing the distribution of system calls.

The feature-engineered dataset contains a total of 687 dimensions, as shown in Table 3. After aggregating and preprocessing all devices, scenarios, and time windows, the final dataset consists of 342,106 feature records. The names of all feature dimensions are provided in the supplementary material (`all_features.txt`) for reference.

Table 3. Feature source counts after merging categories

Input/Output	File System	Kernel Events	Network Records	Resource Usage	System Calls	Total
12	179	80	24	232	160	**687**

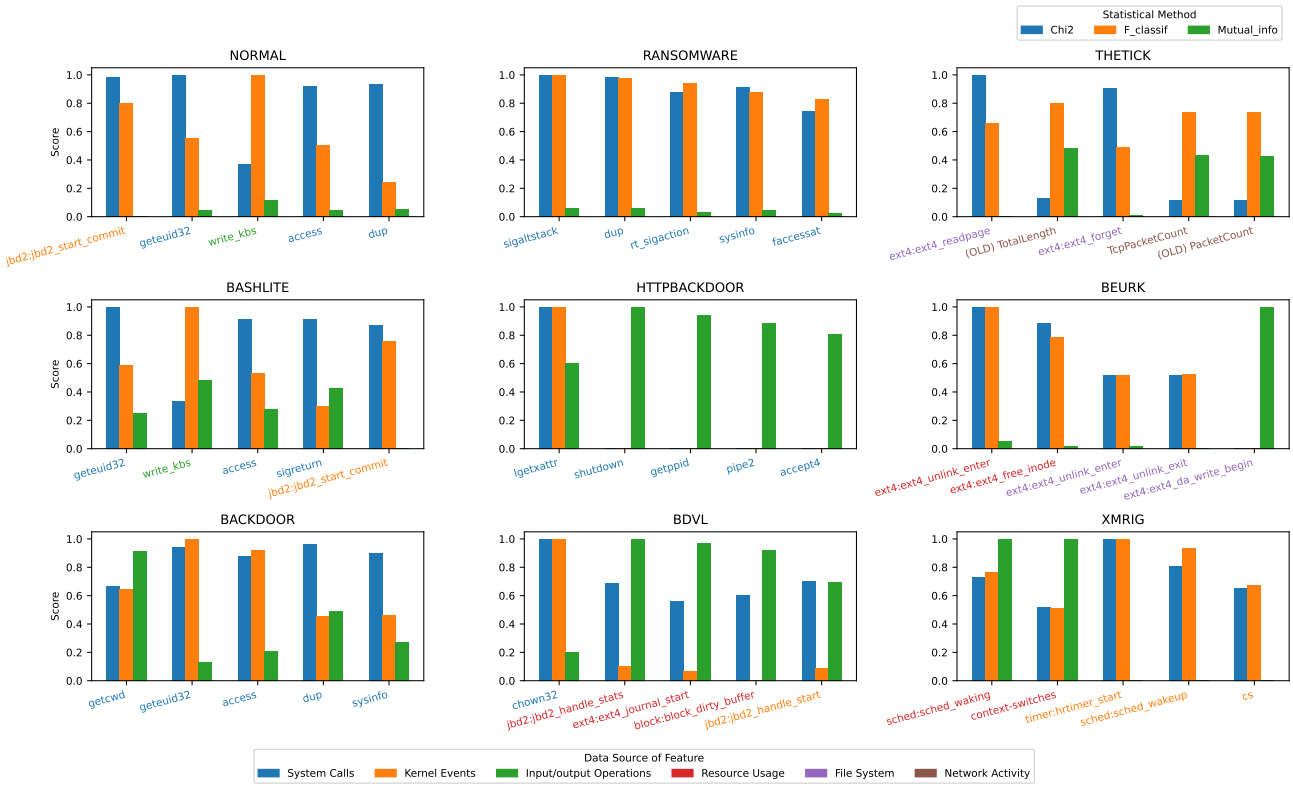


Figure 3. Top-5 feature for each label. Feature names are color-coded by data source.

Technical Validation

This section validates the usability of the proposed dataset for training DFL models for IoT crowdsensing intrusion detection.

Feature Selection

The raw dataset contains 687 dimensions, which is unsuitable for model training on resource-constrained IoT devices due to the high computational and memory overhead. To address this limitation, a feature selection process was carried out to reduce dimensionality while preserving the most informative features. Statistical feature selection was performed to rank features based on their individual relevance to each label. Three widely used statistical tests were employed:

- **Chi-Squared Test (*chi2*):** This test computes the chi-squared statistic between each non-negative feature and the class label, identifying features that are strongly dependent on the target classes.
- **ANOVA F-value (*f_classif*):** The ANOVA F-value measures the ratio of variance between classes to the variance within classes. Features with higher F-values are better at distinguishing between classes.
- **Mutual Information (*mutual_info_classif*):** Mutual information quantifies the dependency between a feature and the target variable using a nonparametric entropy-based estimator. Features with higher scores exhibit stronger association with the labels.

Table 4. Selected 32 features, their behavioral sources, and descriptions, ranked by statistical importance.

Feature	Source	Description
shutdown	System Calls	System call events reflecting process termination.
socket	System Calls	System call for creating or manipulating sockets.
inotify_add_watch	System Calls	Monitoring changes to the file system via inotify.
seconds_RES_data	Resource Usage	Resource utilization over time.
jbd2:jbd2_handle_extend	File System	Journaled file system (ext4) transaction activity.
setgroups32	System Calls	Set process group IDs.
geteuid32	System Calls	Get effective user ID of process.
pipe2	System Calls	Create a pipe for IPC.
ext4:ext4_da_update_reserve_space_RES_data	Resource Usage	Disk space reservation in ext4 file system.
brk	System Calls	Adjust process data segment size.
ext4:ext4_ext_rm_leaf	File System	Remove extent leaf in ext4 file system.
iowritetime	Resource Usage	Time spent writing to I/O devices.
ext4:ext4_ext_remove_space_done	File System	Completed extent space removal in ext4.
recv	System Calls	Receive data from socket.
getegid32	System Calls	Get effective group ID.
iowrite	Resource Usage	Bytes written to I/O devices.
prlimit64	System Calls	Set or get resource limits.
statfs64	System Calls	Get file system statistics.
fchmod	System Calls	Change file permissions.
write_merge	Resource Usage	Merged write operations to I/O.
writeback:sb_clear_inode_writeback_KERN_data	Kernel Events	Kernel-level inode writeback clearing.
util	Input/Output Events	I/O device utilization metrics.
write_kbs	Input/Output Events	Kilobytes written per second to disk.
getsockname	System Calls	Get socket name.
rename	System Calls	Rename file or directory.
block:block_unplug_KERN_data	Kernel Events	Block device queue unplug in kernel.
madvise	System Calls	Advise kernel about memory usage patterns.
armv7_cortex_a15/br_mis_pred/	Resource Usage	CPU branch misprediction count.
setitimer	System Calls	Set timer.
connect	System Calls	Initiate socket connection.
mkdir	System Calls	Create directory.
dup2	System Calls	Duplicate file descriptor.

To investigate the most relevant features for each class, the top-5 features per label were selected based on their average scores across the three statistical tests: *Chi-Squared*, *ANOVA F-value*, and *Mutual Information*, as shown in Figure 3. The sources of the top features differ across attack categories. For normal operation, the top features are primarily from System Calls

and Input/Output Operations. The ransomware sample has all top-5 features originating from System Calls, while the backdoor samples (including HttpBackdoor, Backdoor, and TheTick) show top features mainly from File System and Input/Output Operations. The rootkit samples (Beurk and Bdvl) have top features concentrated in System Calls and Input/Output Operations. In contrast, the coinminer sample (XMRig) includes top features from Resource Usage and Kernel Events.

After statistical selection and deduplication, the 32 most important features were identified and used as the input dimensions for the final model training, as summarized in Table 4. The majority of the selected features originate from System Calls. This indicates that system call patterns are highly discriminative for differentiating between benign and malicious behaviors. In addition, several features are derived from Resource Usage and File System events, suggesting that resource consumption and file system operations also contribute to distinguishing different types of activities.

DFL Model Training and Results

All experiments were conducted on the Nebular platform, which provides a DFL environment with virtualized nodes. The platform supports flexible configurations of datasets, model architectures, data distributions, and network topologies, enabling systematic evaluation of different scenarios.

The evaluation metrics used in all experiments include Accuracy, F1 score, Precision, and Recall, which collectively measure classification performance from multiple perspectives.

As summarized in Table 5, each experiment was run for 10 rounds of federated training, with each round consisting of 3 local epochs per node. Since the task involves malware detection and classification, a nine-class (1 benign + 8 malware) classification model was used, implemented as a three-layer multilayer perceptron (MLP) with a $32 \times 128 \times 9$ architecture. The experiments compared three approaches: traditional ML, CFL, and DFL. For CFL and DFL, FedAvg algorithm was used for model aggregation. In terms of DFL, three network topologies, fully connected, ring, and star, were evaluated, with varying numbers of nodes (4, 8, and 16). In the first experiment, an Independent and identically distributed (IID) data split was used, while the second experiment employed Dirichlet distributions with $\alpha = 1$ to simulate the impact of non-IID data on the DFL model.

Table 5. Experimental configuration overview.

Aspect	Configuration
Training rounds	10
Local epochs per round	3
Model	Three-layer MLP ($32 \times 128 \times 9$)
Approaches compared	ML, CFL, DFL
DFL topologies	Fully connected, Ring, Star
Number of nodes	4, 8, 16
Data distributions	IID; Non-IID simulated with Dirichlet ($\alpha = 1$)

The experimental results are summarized in Table 6. When the data can be centralized, as in the traditional ML) setting, the model achieves the best performance, with accuracy, F1 score, precision, and recall all exceeding 0.94 on the complete dataset. This highlights the advantage of having full access to all data during training.

Table 6. Average performance metrics for different training scenarios and network topologies.

Scenario	Nodes	Accuracy	F1	Precision	Recall
ML	1	0.9447	0.9369	0.9474	0.9447
CFL	4	0.7656 ± 0.16	0.7331 ± 0.18	0.7498 ± 0.18	0.7656 ± 0.16
CFL	8	0.6652 ± 0.19	0.6265 ± 0.22	0.6531 ± 0.22	0.6652 ± 0.19
DFL (Fully)	4	0.8858 ± 0.00	0.8665 ± 0.00	0.8835 ± 0.00	0.8858 ± 0.00
DFL (Fully)	8	0.8318 ± 0.04	0.8064 ± 0.04	0.8342 ± 0.04	0.8318 ± 0.04
DFL (Fully)	16	0.7286 ± 0.09	0.6792 ± 0.10	0.6895 ± 0.10	0.7286 ± 0.09
DFL (Ring)	4	0.8948 ± 0.03	0.8754 ± 0.03	0.8921 ± 0.03	0.8948 ± 0.03
DFL (Ring)	8	0.8204 ± 0.04	0.7901 ± 0.04	0.8138 ± 0.04	0.8204 ± 0.04
DFL (Star)	4	0.9002 ± 0.01	0.8840 ± 0.01	0.8987 ± 0.01	0.9002 ± 0.01
DFL (Star)	8	0.8350 ± 0.05	0.8063 ± 0.05	0.8304 ± 0.05	0.8350 ± 0.05

The DFL approach preserves data privacy by keeping data local to each node, but at the cost of a slight drop in model performance. For example, with four nodes and a fully connected topology, the DFL model achieves an F1 score of

approximately 0.88, which is lower than ML but still competitive. This performance drop is mainly attributed to the reduced amount of data available in each node. Notably, DFL consistently outperforms CFL in most scenarios, indicating that the dataset characteristics and heterogeneity are better suited to the decentralized training paradigm proposed in this work.

In terms of scalability, the results show that increasing the number of nodes negatively impacts model performance. As the number of nodes increases from 4 to 16, the F1 score decreases, reflecting the diminishing amount of data allocated to each node and the increasing challenge of maintaining global consistency.

Regarding the impact of network topology, the results suggest that the choice of topology has only a minor effect on model performance for this dataset. Fully connected, ring, and star topologies yield comparable results at the same node count, indicating that the dataset is not highly sensitive to the communication structure of the DFL network.

In terms of the impact of data distribution on DFL performance, experiment was conducted with an 8-node fully connected DFL setup with Dirichlet α equals to 1. As shown in Table 7, the model's precision dropped to approximately 0.78, resulting in an F1 score of 0.79. This represents a degradation compared to the IID setting.

Dirichlet α	Accuracy	F1	Precision	Recall
1	0.8397	0.7933	0.7822	0.8397

Table 7. DFL Model performance with a fully connected topology with 8 nodes in non-IID setting.

The experiments demonstrate that the collected dataset supports effective training of DFL models. DFL achieves reasonable performance while preserving data locality, and outperforms CFL in most settings. The results also show that increasing the number of nodes or introducing non-IID data distributions reduces model performance, reflecting the data sparsity and heterogeneity introduced by the dataset. Overall, the dataset enables evaluation of privacy-preserving learning approaches under realistic constraints of distribution and scale.

Limitations and Future Work

The current experiments are limited to a specific model architecture, which may not generalize to other types of malware or hardware platforms. The impact of more diverse node capabilities, asynchronous training, and dynamic network conditions was not investigated. Future work includes extending the dataset with additional malware families and benign behaviors, evaluating more complex and heterogeneous models, and incorporating resource-constrained nodes to better reflect real-world IoT environments.

Usage Notes

The dataset is intended for research on DFL and malware detection tasks. Users should be aware that the data distribution is non-IID, and class imbalance exists across certain labels. It is recommended to perform feature normalization and stratified splitting when training models.

Code availability

The scripts for data collection are available at: https://github.com/Cyber-Tracer/MalwareDetectionDataset and the scripts for data processing and model training are provided at: https://github.com/Cyber-Tracer/iot-feature-engineering.

References

1. Huertas, A. *et al.* Intelligent and behavioral-based detection of malware in iot spectrum sensors. *Int. J. Inf. Secur.* **22**, DOI: 10.1007/s10207-022-00602-w (2022).
2. Celdrán, A. H. *et al.* Privacy-preserving and syscall-based intrusion detection system for iot spectrum sensors affected by data falsification attacks. *IEEE Internet Things J.* (2022).
3. association, E. Electrosense - collaborative spectrum monitoring (2016). Accessed 2024-01-10.
4. Beltrán, E. T. M. *et al.* Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Commun. Surv. & Tutorials* (2023).
5. Nguyen, T. D. *et al.* Dïot: A federated self-learning anomaly detection system for iot. In *2019 IEEE 39th International conference on distributed computing systems (ICDCS)*, 756–767 (IEEE, 2019).

6. Rey, V., Sánchez, P. M. S., Celdrán, A. H. & Bovet, G. Federated learning for malware detection in iot devices. *Comput. Networks* **204**, 108693 (2022).

7. Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A. A. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, 1–6 (Ieee, 2009).

8. Sharafaldin, I., Lashkari, A. H., Ghorbani, A. A. *et al.* Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **1**, 108–116 (2018).

9. Koroniotis, N., Moustafa, N., Sitnikova, E. & Turnbull, B. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Futur. Gener. Comput. Syst.* **100**, 779–796 (2019).

10. Alsaedi, A., Moustafa, N., Tari, Z., Mahmood, A. & Anwar, A. Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems. *Ieee Access* **8**, 165130–165150 (2020).

11. Ilavarasan, E. & Muthumanickam, K. A survey on host-based botnet identification. In *2012 International Conference on Radar, Communication and Computing (ICRCC)*, 166–170, DOI: 10.1109/ICRCC.2012.6450569 (2012).

12. Meidan, Y. *et al.* N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Comput.* **17**, 12–22 (2018).

13. Koroniotis, N., Moustafa, N., Sitnikova, E. & Turnbull, B. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Futur. Gener. Comput. Syst.* **100**, 779–796, DOI: https://doi.org/10.1016/j.future.2019.05.041 (2019).

14. Bezerra, V. H., da Costa, V. G. T., Junior, S. B., Miani, R. S. & Zarpelão, B. B. Iotds: A one-class classification approach to detect botnets in internet of things devices. *Sensors (Basel, Switzerland)* **19** (2019).

15. da Costa, V. G. T., Barbon, S., Miani, R. S., Rodrigues, J. J. P. C. & Zarpelão, B. B. Detecting mobile botnets through machine learning and system calls analysis. In *2017 IEEE International Conference on Communications (ICC)*, 1–6, DOI: 10.1109/ICC.2017.7997390 (2017).

16. Martinelli, F., Mercaldo, F. & Saracino, A. Bridemaid: An hybrid tool for accurate detection of android malware. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 899–901, DOI: 10.1145/3052973.3055156 (2017).

17. Saracino, A., Sgandurra, D., Dini, G. & Martinelli, F. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable Secur. Comput.* **15**, 83–97, DOI: 10.1109/TDSC.2016.2536605 (2018).

18. Zhang, Y. & Paxson, V. Detecting backdoors. In *9th USENIX Security Symposium (USENIX Security 2000)*, 1–15 (2000).

19. Canzanese, R., Mancoridis, S. & Kam, M. System call-based detection of malicious processes. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, 119–124, DOI: 10.1109/QRS.2015.26 (2015).

20. Hoglund, G. & Butler, J. *Rootkits: subverting the Windows kernel.* (Addison-Wesley Professional, 2005).

21. Kruegel, C., Robertson, W. & Vigna, G. Detecting kernel-level rootkits through binary analysis. In *20th Annual Computer Security Applications Conference*, 91–100, DOI: 10.1109/CSAC.2004.19 (2004).

22. Baliga, A., Ganapathy, V. & Iftode, L. Automatic inference and enforcement of kernel data structure invariants. In *2008 Annual Computer Security Applications Conference (ACSAC)*, 77–86, DOI: 10.1109/ACSAC.2008.29 (2008).

23. Nick L. Petroni, J., Fraser, T., Molina, J. & Arbaugh, W. A. Copilot—a coprocessor-based kernel runtime integrity monitor. In *13th USENIX Security Symposium (USENIX Security 04)* (USENIX Association, San Diego, CA, 2004).

24. Baliga, A., Ganapathy, V. & Iftode, L. Detecting kernel-level rootkits using data structure invariants. *IEEE Transactions on Dependable Secur. Comput.* **8**, 670–684, DOI: 10.1109/TDSC.2010.38 (2011).

25. Carbone, M. *et al.* Mapping kernel objects to enable systematic integrity checking. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, 555–565, DOI: 10.1145/1653662.1653729 (Association for Computing Machinery, New York, NY, USA, 2009).

26. Kok, S., Abdullah, A., Jhanjhi, N. & Supramaniam, M. Ransomware, threat and detection techniques: A review. *Int. J. Comput. Sci. Netw. Secur.* **19**, 136 (2019).

27. Barbhuiya, S., Papazachos, Z., Kilpatrick, P. & Nikolopoulos, D. S. Rads: Real-time anomaly detection system for cloud data centres (2018). 1811.04481.

28. Tanana, D. Behavior-based detection of cryptojacking malware. In *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, 0543–0545, DOI: 10.1109/USBEREIT48449.2020.9117732 (2020).

29. Bashlite. Bashlite botnet source code. https://github.com/hammerzeit/BASHLITE. Accessed: 2025-07-14.

30. HttpBackdoor. Httpbackdoor source code. https://github.com/SkryptKiddie/httpBackdoor. Accessed: 2025-07-14.

31. Backdoor. Backdoor source code. https://github.com/jakoritarleite/backdoor. Accessed: 2025-07-14.

32. TheTick. Thetick backdoor source code. https://github.com/nccgroup/thetick. Accessed: 2025-07-14.

33. Beurk. Beurk rootkit source code. https://github.com/unix-thrust/beurk. Accessed: 2025-07-14.

34. Bdvl. Bdvl rootkit source code. https://github.com/Error996/bdvl. Accessed: 2025-07-14.

35. Ransomware-PoC. Ransomware-poc source code. https://github.com/jimmy-ly00/Ransomware-PoC. Accessed: 2025-07-14.

36. XMRig. Xmrig miner source code. https://github.com/xmrig/xmrig. Accessed: 2025-07-14.

37. Feng, C. *et al.* IoT intrusion detection dataset for decentralized federated learning, DOI: 10.57760/sciencedb.25380 (2025).

Acknowledgements

This work was supported by the Swiss Federal Office for Defense Procurement (armasuisse) under the CyberDFL project (CYD-C-2020003) and by the University of Zürich, UZH.

Author contributions statement

G.B. conceived and designed the study. J.H., H.Q., X.C., Z.Z., and L.K. performed the experiments and data analysis. C.F. and A.H.C. drafted the manuscript with contributions from all authors. B.S. provided overall supervision. All authors reviewed and approved the final manuscript.

Ethics declarations

Competing interests

The authors declare no competing interests.