electronics

MDPI

Article

Generation of a Multi-Class IoT Malware Dataset for Cybersecurity

Mazdak Maghanaki [1,*], Soraya Keramati [2], F. Frank Chen [1] and Mohammad Shahin [3]

[1] Department of Mechanical, Aerospace, and Industrial Engineering, The University of Texas at San Antonio, San Antonio, TX 78249, USA
[2] Department of Mineral, Metallurgical, and Materials Engineering, Université Laval, Québec City, QC G1V 0A6, Canada
[3] Department of Industrial and Systems Engineering, University of Tennessee, Knoxville, TN 37996, USA
* Correspondence: mazdak.mohammadimaghanaki@utsa.edu

Abstract

This study introduces a modular, behaviorally curated malware dataset suite consisting of eight independent sets, each specifically designed to represent a single malware class: Trojan, Mirai (botnet), ransomware, rootkit, worm, spyware, keylogger, and virus. In contrast to earlier approaches that aggregate all malware into large, monolithic collections, this work emphasizes the selection of features unique to each malware type. Feature selection was guided by established domain knowledge and detailed behavioral telemetry obtained through sandbox execution and a subsequent report analysis on the AnyRun platform. The datasets were compiled from two primary sources: (i) the AnyRun platform, which hosts more than two million samples and provides controlled, instrumented sandbox execution for malware, and (ii) publicly available GitHub repositories. To ensure data integrity and prevent cross-contamination of behavioral logs, each sample was executed in complete isolation, allowing for the precise capture of both static attributes and dynamic runtime behavior. Feature construction was informed by operational signatures characteristic of each malware category, ensuring that the datasets accurately represent the tactics, techniques, and procedures distinguishing one class from another. This targeted design enabled the identification of subtle but significant behavioral markers that are frequently overlooked in aggregated datasets. Each dataset was balanced to include benign, suspicious, and malicious samples, thereby supporting the training and evaluation of machine learning models while minimizing bias from disproportionate class representation. Across the full suite, 10,000 samples and 171 carefully curated features were included. This constitutes one of the first dataset collections intentionally developed to capture the behavioral diversity of multiple malware categories within the context of Internet of Things (IoT) security, representing a deliberate effort to bridge the gap between generalized malware corpora and class-specific behavioral modeling.

Keywords: Malware Dataset Generation; malware detection; sandbox; Industrial IoT; machine learning; cybersecurity

check for updates

Academic Editors: Beibei Li, Peican Zhu, Jiwei Tian and Yafei Song

Received: 18 September 2025
Revised: 23 October 2025
Accepted: 24 October 2025
Published: 27 October 2025

Citation: Maghanaki, M.; Keramati, S.; Chen, F.F.; Shahin, M. Generation of a Multi-Class IoT Malware Dataset for Cybersecurity. *Electronics* **2025**, *14*, 4196. https://doi.org/10.3390/electronics14214196

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Malware detection remains one of the fundamental challenges in contemporary cybersecurity, since the rapid expansion in both the scale and sophistication of cyber threats continues to accelerate at an unprecedented pace [1]. The development of reliable and fine-grained detection systems is not merely advantageous but is also indispensable for

the maintenance of secure and resilient digital infrastructures, particularly as the attack surface of modern computing environments expands across enterprise networks, consumer devices, and the ever-growing Internet of Things. In recent years, machine learning has emerged as a particularly powerful paradigm for identifying malicious software, primarily through the analysis of behavioral patterns and statistical features that are extracted during program execution [2]. The effectiveness of these models is determined not only by the choice of algorithm but also, more critically, by the quality, structure, and contextual specificity of the datasets employed for training and evaluation [3]. Although an enormous volume of malware samples is readily available through public and private feeds, there exists a pronounced shortage of datasets that are explicitly designed to capture the distinctive operational characteristics of specific malware families. Many existing datasets, for example, are generalized in structure and aggregate samples from a wide array of malware categories into a single, monolithic collection. While such aggregation may appear convenient from the standpoint of data acquisition and scalability, it fails, consequently, to acknowledge that each malware type is characterized by unique operational signatures manifested through differences in system modification patterns, registry edits, network communication behaviors, and execution flows, which are precisely the elements most crucial for accurate and interpretable classification [4]. As a result, models trained on these broad, undifferentiated datasets frequently struggle to distinguish the subtle, yet operationally significant, behavioral nuances that separate one category from another [5,6]. For this reason, designing datasets that are built around individual categories offers a practical advantage. Behavioral traits that are unique to a given malware family can be captured in a way that strengthens both the accuracy of detection and the clarity of the model's output [7]. Classification is not straightforward, because many modern threats blur the lines between categories by including multiple functions in a single sample. A single program may, for instance, include features for remote access, data theft, and file encryption all at once [8]. In practice, classification often follows the malware's primary goal rather than its internal structure. To illustrate this, if the main outcome of an attack is the encryption of user files to demand payment, the program is labeled as ransomware, even when some of its modules resemble those of a remote access Trojan [9]. Figure 1 outlines the eight categories of malware that dominate the current cybersecurity landscape and are relevant to Internet of Things environments. These include Trojan, Mirai (botnet), ransomware, rootkit, worm, spyware, keylogger, and virus. The categories were chosen because they appear frequently in both enterprise and consumer systems, and because they are consistently highlighted in academic research as well as in professional threat intelligence reporting [10–12].

In this study, each malware type was considered as a separate modeling problem, since treating them in a combined way often hides the very differences that make classification possible in the first place. Features were chosen independently for each dataset, relying on domain knowledge, previously published studies, and on what are already known to be the behavioral signatures of the different malware families [13–16]. For example, in the case of botnets, network flow metrics such as packet counts, inter-arrival times, and destination ports were prioritized, because these values are repeatedly associated with command-and-control communication (C3) in the literature [17–19]. Ransomware, by contrast, required a different strategy: here, both network telemetry and blockchain-related variables were included, since modern extortion campaigns are not only technical but also economic in nature, and the financial transactions behind them form part of the attack chain that must be captured [20,21]. All malware samples were executed in isolation inside the secure, cloud-based AnyRun sandbox [22]. Running the samples in this way prevented contamination from other processes and ensured that the behaviors observed could be

traced directly to the sample under study. AnyRun provides a very wide range of runtime attributes, but only those that have real diagnostic or discriminative value were retained, because including everything would only add noise without improving the quality of the datasets. The suite consists of eight modular datasets, one for each malware type, and every dataset contains malicious, benign, and suspicious samples that were processed under controlled conditions. The attributes in each case were aligned with what the literature describes as the core operational traits of the corresponding malware family. Therefore, the datasets do not simply repeat generic indicators but instead reflect the actual differences that matter for classification. Unlike large aggregated collections, which blur distinctions and reduce interpretability, this modular framework makes it easier to see why a model makes the predictions it does and to capture the small but important behavioral details that separate one category from another. As a result, the design provides a practical foundation for applying machine learning to malware detection in real environments, especially in the increasingly exposed domain of Internet of Things devices.

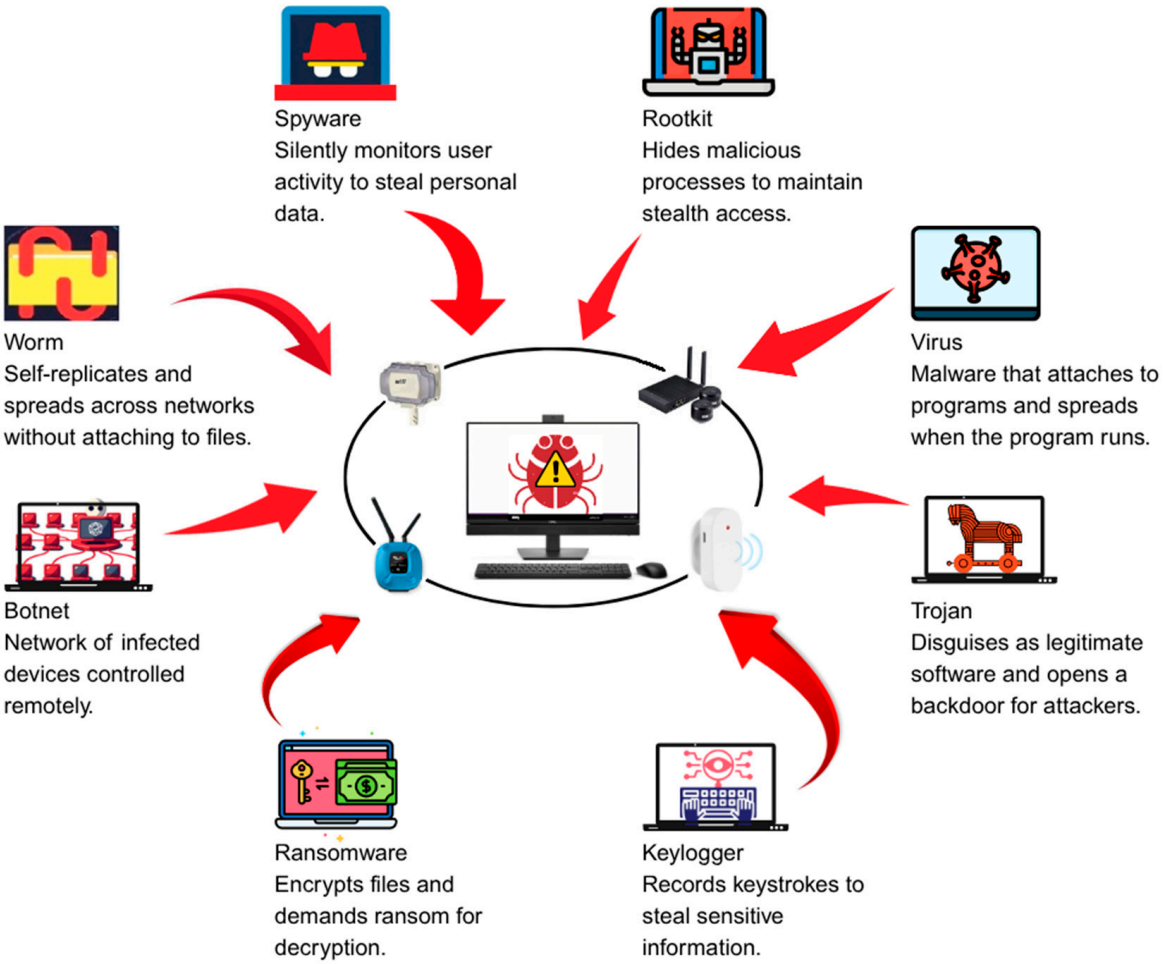


Figure 1. Eight major categories of malware frequently observed in IoT environments.

2. Literature Review

Creating reliable and representative datasets for malware detection remains one of the most persistent challenges in cybersecurity research. A growing number of studies have pointed out that many widely used public datasets suffer from structural issues that directly undermine their accuracy, reproducibility, and, ultimately, their usefulness in practice. In other words, even though these datasets are widely cited, their flaws create real obstacles for building detection systems that can keep up with the diversity and rapid

evolution of malware seen in real-world environments [23–25]. One issue that comes up again and again in the literature is the inconsistency of malware labels. Detection systems that depend on antivirus engines inherit the problem that different engines often classify the very same sample in very different ways. These discrepancies make it difficult to trust datasets that are assembled through automated aggregation, especially when the labels are taken as the ground truth without further verification [26]. In modern IoT-driven manufacturing, cloud space must be taken more seriously when investigating cyberattacks, as it has become the core layer linking devices, data, and services. The growing reliance on interconnected cloud infrastructures increases exposure to risks, and understanding how these environments enable service interaction is essential for addressing data breaches, service hijacking, and unauthorized access in industrial systems [27]. Industry 4.0 brings together advanced technologies like cybersecurity and IoT devices, both of which can be applied in healthcare to improve reliability, data integrity, and patient monitoring. When combined with innovations such as computer vision and lean healthcare principles, these tools can support earlier diagnosis, safer data handling, and more efficient clinical workflows [28,29].

Without a clear and standardized framework for labeling, machine learning models risk learning unreliable correlations, or worse, overfitting to mislabeled examples, which severely reduces their reliability once they are deployed in real systems [30]. Another recurring problem is the presence of duplicates or near-duplicates. These can arise either from deliberate code changes designed to evade detection or simply from poor duplication checks during dataset compilation. Redundancy of this kind can distort both the training and testing phases. For instance, in unsupervised learning setups, duplicate entries can artificially inflate accuracy by exposing the model to nearly identical inputs more than once [31]. Earlier studies have also been criticized for collapsing all malware into a single undifferentiated category. While this may simplify dataset construction, it reduces the granularity of the models trained on them and, as a result, their effectiveness [32,33]. Some research has shown that transforming executables into image-like representations and applying hybrid vision transformer approaches can improve detection accuracy by extracting spatial features across different families [34,35]. However, when datasets merge all malicious activity into one broad category, they fail to account for the very different behavioral strategies employed by distinct families. This leads to reduced performance both in family-level classification and in behavioral analysis [36,37]. More recent work has emphasized the need for context-aware classification, noting that ransomware, spyware, and Trojans each have operational traits and evolutionary patterns that deserve to be modeled separately [38]. Long-term monitoring of families has further shown that focusing on category-specific behavior provides valuable insight into the evolution of threats and supports more resilient detection strategies [10,39].

Recent work in dataset design has shifted toward behavior-focused approaches, as exemplified by the TON-IoT and CICIoT2023 datasets, which capture dynamic runtime activity through IoT telemetry, operating system audit logs, and network traffic collected during live attack simulations [40,41]. These methods focus on runtime signals—such as process creation, system call sequences, registry edits, file manipulation, and network activity—that static analysis alone cannot capture. By concentrating on features with demonstrated diagnostic value, such behavior-focused datasets offer a more faithful view of how malware behaves in practice [42]. For example, one study tested a hybrid ensemble that stacked vision transformers with convolutional networks, combining hierarchical spatial features with uncertainty modeling. This approach improved performance across diverse families and demonstrated that hybrid designs can address the variability of malware more effectively than single-model systems [43]. Several benchmark datasets

have been especially influential in the field. One widely cited example offered labeled Windows executables along with tools for feature extraction and baseline models, which made reproducibility possible for many follow-up studies [44]. Another influential line of work involved converting binaries into grayscale images and applying computer vision techniques. In these cases, convolutional networks were able to detect structural patterns in byte sequences, and when tuned in deep learning frameworks, these grayscale image datasets—like Malimg and Microsoft Malware—produced near-perfect classification results [45]. Such findings highlight how visual and entropy-based patterns can capture family-specific differences effectively. There have also been efforts to improve the semantic and temporal aspects of malware labeling. For example, one Android dataset collected applications over four years from Google Play, recording not only the APK files but also metadata such as reviews, download counts, and descriptive text. Labeling was achieved using a mix of VirusTotal scans and long-term observation of whether the apps were eventually removed. This created a high-confidence dataset that covered dozens of families and, importantly, reflected real-world lifecycle patterns. It solved two persistent problems at once: the outdated nature of static repositories and the overly narrow focus of datasets built only from file content [46]. In IoT security research, stochastic datasets that capture the random and evolving nature of device behavior can play a key role in improving forecasting and AI-based threat prediction. Statistical forecasting methods and machine learning models can analyze these variable patterns to identify subtle trends, adapt to unpredictable activity, and enhance decision-making in dynamic, data-driven IoT environments [47]. Another major step forward came with the MOTIF dataset, which contained thousands of samples across more than four hundred families and aligned them carefully with intelligence reports. One of its strongest contributions was the inclusion of alias mappings, which made family labels consistent across the many naming conventions used by different vendors. Evaluations with MOTIF showed just how inaccurate majority-vote heuristics and tools like AVClass can be, exposing the noise that comes from inconsistent labeling. Through its scale and semantic clarity, MOTIF set a new bar for dataset quality, showing that large resources could still be rigorous if carefully curated [48].

A literature review of more than forty studies further highlighted the state of dataset usage, identifying thirty-seven datasets, forty-seven frameworks, and six targeted computing platforms. This revealed a striking lack of transparency in many works, with common shortcomings including vague feature extraction descriptions and poorly explained labeling schemes. The review emphasized the urgent need for standardized reporting practices in dataset design [49]. Together, these findings reinforce the trend toward modular, behavior-driven datasets. Researchers have been moving away from large but undifferentiated repositories and toward smaller, structured collections built around operational behavior [50]. This shift places relevance and interpretability above sheer sample size and has enabled more accurate and family-specific model training [51].

Many publicly available datasets are outdated, inconsistently labeled, or rely too heavily on vendor heuristics. Others adopt a uniform structure that ignores the diversity of malware strategies. These flaws result in models that overfit to noise, fail to generalize, or look strong on paper but collapse under real-world testing [52,53]. Ambiguity in family classification adds another layer of uncertainty, reducing both interpretability and utility [54]. A comparative overview of the benchmark malware datasets and their principal contributions is presented in Table 1, which consolidates the studies discussed throughout the literature review to improve clarity and reader accessibility.

The framework presented in this study addresses these issues by introducing a modular dataset design built around clarity, efficiency, and behavioral relevance. By separating malware into multiple datasets, each tied to a specific type and aligned with its oper-

ational profile, the approach enables deliberate feature selection, lowers training costs, and supports tailored detection strategies. The dataset suite thus represents a step forward for malware research, offering a foundation that prioritizes precision, scalability, and real-world applicability.

Table 1. Comparative summary of malware datasets and key contributions.

Dataset	Domain	Key Features or Innovations	Limitations	Reference(s)
TON-IoT	IoT and Edge Security	Captures dynamic telemetry, audit logs, and network traffic from live IoT attack simulations. Enables runtime behavior analysis.	Limited diversity of IoT device types and attack scenarios.	[40]
CICIoT2023	IoT-based Threat Detection	Integrates system calls, file manipulations, and network traces for behavior-driven classification. Supports hybrid learning models.	Lacks standardized labeling protocol and fine-grained class distinctions.	[41]
MOTIF	Windows Malware Families	Includes 400+ malware families with alias mapping across vendors; aligns samples with intelligence reports; improves labeling consistency.	High storage and processing requirements; limited coverage of newer threats.	[48]
Microsoft Malware	Executable Image Datasets	Converts binaries to grayscale images for CNN-based classification; captures structural and entropy-based patterns.	Overfitting risk due to visual redundancy; lacks dynamic behavior context.	[45]
AVClass	Automated Malware Labeling	Provides tools for large-scale family labeling using AV consensus and heuristics.	Inconsistent results due to vendor disagreement; heuristic noise.	[30]
Malsource	Malware Code Reuse Study	Analyzes code reuse and complexity across malware families; measures long-term evolution.	Static-only analysis; does not capture runtime behaviors.	[39]
OmniDroid	Hybrid Feature Fusion	Combines static and dynamic features for Android malware detection using ensemble models.	Data imbalance and potential outdated samples.	[52]
KronoDroid	Android Time-based Dataset	Captures temporal evolution of Android malware features to enhance chronological modeling.	Limited to Android; not representative of cross-platform threats.	[53]
Hybrid Transformers	Behavior Modeling	Combines hierarchical spatial and semantic features for improved detection accuracy and interpretability.	High computational cost; needs large-scale data.	[34,35,43]
Android Lifecycle Dataset	Mobile App Evolution Tracking	Collects apps and metadata over 4 years from Google Play, tracking removals and version histories.	Focused on app store dynamics, less on deep system behavior.	[46]
Sisyfos	Modular Malware Analysis	Provides modular framework integrating analysis tools and datasets for extendable experimentation.	Depends on dataset input quality; limited documentation on features.	[51]
Systematic Review	Meta-analysis of 40+ Studies	Identifies 37 datasets and 47 frameworks; highlights lack transparency and need for standardized reporting.	No experimental data; review-only.	[49]

3. Methodology

To build a modular and behaviorally separated malware dataset suite, a two-source acquisition process was used. Data came primarily from Anyrun, which is an interactive malware sandbox that operates in a controlled virtual environment, and from open repositories available on GitHub [22,55–62]. The reason for this two-way approach was simple: not all malware families are distributed evenly across platforms. Some types, like Trojans or ransomware, are found in large numbers in sandbox archives, while others, such as worms and traditional viruses, are poorly represented there and needed to be supplemented from outside repositories. Without combining both sources, the dataset would have been skewed and incomplete.

Most of the samples were collected from Anyrun, which maintains more than two million files labeled as malicious, suspicious, or benign. One advantage of Anyrun is that it does not just store executables but also allows them to be executed in a live virtual environment. This produces detailed behavioral information, including file changes, process trees, system calls, registry edits, and network activity. For each malware type, representative samples were selected and run inside the sandbox. Executions were carried out one by one, in isolation, so that the behavioral traces belonged only to the sample being tested and were not contaminated by others running at the same time.

After execution, the reports generated by Anyrun were exported in plain text format. These logs included structured details of processes, files, registry activity, and network communication. From these raw reports, attributes were extracted according to what was known to be useful for each malware type. For example, ransomware was examined for signs of file encryption, ransom note creation, or attacker communication channels. Mirai samples were chosen based on evidence of scanning, propagation, and distributed denial-of-service functions. The goal was to focus only on features that had diagnostic value, rather than collecting every possible attribute, which may increase noise and make the resulting models harder to interpret.

Because viruses and worms were not well covered in Anyrun, additional samples were taken from GitHub repositories containing executable binaries. These collections were especially useful for expanding the range of worm and virus behaviors, including replication strategies in file-infecting viruses and code modification in worms. After execution and feature extraction, the attributes for each family were stored in CSV format. Every sample was labeled as malicious, suspicious, or benign, based on the classification provided by its source. At this stage, the datasets were manually cleaned: corrupted entries, missing fields, or incomplete runs were removed to preserve integrity.

This process differs from earlier efforts in two main ways. First, domain knowledge was directly applied during curation, so features were not blindly collected but chosen with reference to what is known about how each malware family behaves. Second, the approach deliberately combined the richness of sandbox telemetry with external sources to cover underrepresented types. The result is a dataset suite that is modular, internally consistent, and aligned with the behavioral expectations of eight malware categories. Such a structure not only improves interpretability but also reduces unnecessary computational load and, most importantly, makes the datasets more relevant for use in real-world detection systems in Internet of Things contexts. By grounding dataset design in both technical rigor and contextual awareness, this effort creates a foundation that can be readily extended to new malware classes as IoT threats continue to evolve.

3.1. Sandboxing

Sandboxing provides a safe environment to execute and observe malware, making it essential in IoT settings where infections can spread rapidly and behavioral analysis must be performed without risking real devices [63]. At the basic level, a sandbox is nothing more than an isolated and instrumented environment—either virtual or physical—where suspicious files can be executed without putting the real host at risk. This separation makes it possible to watch what the file actually does in real time, and to record things like process creation, registry changes, network calls, file system activity, and sometimes even memory usage. The key point is that all of this can be observed without interference from the outside and without contaminating the broader system [64]. In practice, sandboxing usually involves two complementary layers: static analysis and dynamic execution. Static analysis is the stage where the file is inspected without running it [65]. Analysts often look at headers, metadata, hashes, and embedded strings, or break down the portable executable structure

to obtain a first impression and to compare against known signatures [66]. Dynamic analysis begins once the file is actually run inside the sandbox [67]. At that point, every activity, like registry edits, process trees, and network traffic, is logged. The strength of combining static inspection with dynamic execution is that it helps catch malware that uses tricks like obfuscation, evasion, or delayed execution [68]. Because threat actors are leaning on these methods more and more, the ability of sandboxes to record accurate behavior has become essential. Another role of sandboxing is in finding zero-day threats, cases where no known signature exists [69]. Sandboxes can also feed real-time threat intelligence because they generate fresh behavioral traces as the malware runs [70]. Old signature-based detection alone is not fast enough to keep up with polymorphic or metamorphic malware, so behavioral profiling in sandboxes is now a core element of modern SOC operations, endpoint detection and response, and threat intel platforms [71,72].

In this study, Anyrun was the tool chosen to handle both sandboxing and malware intelligence. Anyrun is cloud-based, it allows for live execution, and it is interactive. It supports different Windows versions and many file types. Unlike traditional batch sandboxes, it lets the user interact directly with the virtual machine, mouse clicks, keyboard input, and file moves, which makes it harder for malware to hide by pretending to be dormant [22]. For dataset generation, Anyrun offers two main advantages. The first is its public malware database, which holds over two million labeled samples and grows daily. Each record includes rich metadata: hashes, execution traces, process trees, dropped files, mutexes, registry edits, network logs, and so on. This makes it possible to pick and separate samples for specific families. The second advantage is the text report export. These reports provide structured summaries of everything that happened, like system events, network flows, and threat indicators. Also, they can be parsed directly into features. This simplifies the work of turning raw behavior into usable data.

In this study, samples were taken from eight categories: Trojan, ransomware, Mirai, rootkit, spyware, keylogger, worm, and virus. Each sample was either run in Anyrun or retrieved from its database, and the behavioral data were taken from the text reports. The goal was to capture typical patterns that matched what is already known about each family. Every execution was run in isolation to avoid contamination. This combination of execution, labeling, and telemetry in one system removed many of the old barriers to dataset construction, such as the need to build and maintain a private sandbox from scratch.

A session was started by logging into Anyrun and uploading the sample. Depending on the requirements, either Simple Mode was used, where a file is uploaded and run in a default virtual machine, or Pro Mode was chosen, which allows for control over network routing, environment parameters, and other options. Once the run started, static data such as hashes and metadata were recorded, along with dynamic data such as process activity, registry changes, file system edits, and network communications. When the run ended, the reports were exported and parsed into datasets containing both static and behavioral features. The overall procedure is shown in Figure 2.

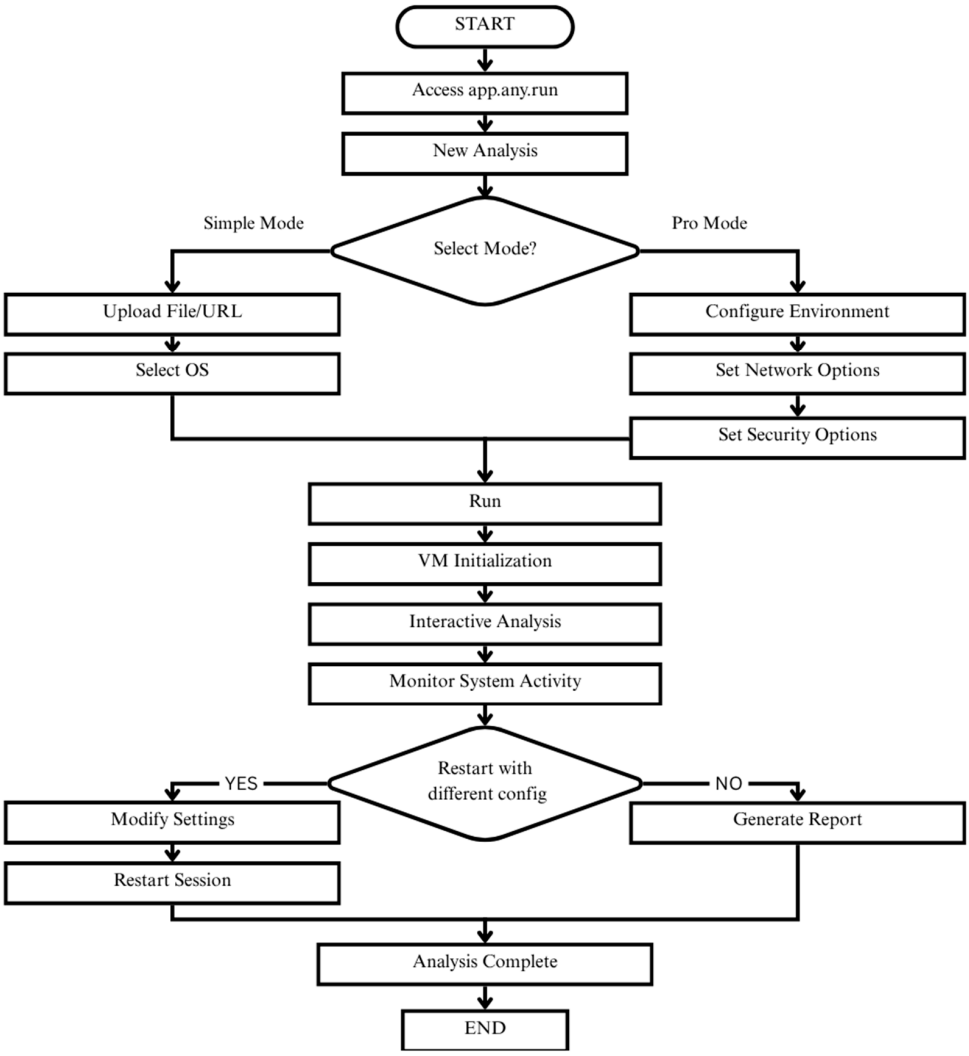


Figure 2. Workflow diagram depicting the sandbox-based malware execution and analysis process using the Anyrun platform.

3.2. Data

This study employed two sources for data acquisition: the AnyRun platform and publicly available repositories hosted on GitHub [22,55–62]. Together, these sources provided a diverse collection of executable files that were classified as malware, suspicious, or benign. The strength of AnyRun lies in the fact that it records system-level activity in detail, including process hierarchies, file system modifications, API calls, and network communications. In practice, these traces capture the actual behavior of a program and therefore reflect the operational characteristics of each sample in a way that static inspection alone cannot. The platform also supports multiple operating systems and integrates analytical features such as YARA rule matching and links to the MITRE ATT&CK framework, which makes it a useful foundation for constructing a dataset that is not only broad but also behaviorally rich [22]. Each file in the study was placed into one of three operational categories—malicious, suspicious, or benign. Every sample was executed individually inside the sandbox so that its behavior could be recorded without contamination from other processes, and after execution a detailed analysis report was generated. These reports provided both static properties and dynamic behavioral indicators, from which attributes were extracted. Not every attribute was retained; only those with clear diagnostic value were kept, while irrelevant metadata or noisy variables were excluded. This step was

important for interpretability and also reduced the computational overhead that would have come from unnecessarily large feature sets.

The final dataset suite consists of eight separate datasets, each aligned with one malware category. Each dataset was standardized to 1250 samples. This number was chosen as a balance: large enough to allow for meaningful analysis and model training, yet still practical for execution and handling within the sandboxing framework. Suspicious samples were included selectively, particularly when their behavior was ambiguous or when they partially matched known signatures. The reasoning here was that real-world operations often involve uncertainty, and including such cases makes the dataset more realistic and closer to the conditions faced by security analysts. All datasets were exported in comma-separated values (CSV) format, since CSV files are straightforward to parse and are compatible with widely used data science environments such as Python and R. This structured, tabular representation ensures that the datasets can be integrated directly into machine learning workflows and supports efficient analysis at scale. Each dataset was curated with an emphasis on clarity of behavior, diversity across categories, and relevance to the kinds of threats currently documented in practice. The composition of each malware dataset, including sample counts, is summarized in Table 2.

Table 2. Sample counts for each malware dataset.

Dataset	Malware Samples	Suspicious Samples	Benign Samples	Total
Mirai	350	150	750	1250
Trojan	400	200	650	1250
Ransomware	370	230	650	1250
Rootkit	320	180	750	1250
Worm	350	200	700	1250
Spyware	300	300	650	1250
Keylogger	300	300	650	1250
Virus	300	300	650	1250

Feature relevance was evaluated using a single decision tree to compute Gini-based importance scores. The model followed default *scikit-learn* parameters (criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, random_state=42), which allows the algorithm grow fully so that all informative splits were considered. The unpruned structure avoids bias and guarantees the attributes contribute to impurity reduction analysis. Highly correlated features ($|\rho| \geq 0.90$) were removed through correlation filtering, and domain-specific variables were normalized to a 0–1 range using min–max scaling to reduce variance, mitigate bias toward high-cardinality variables, and maintain reproducibility across all datasets, avoiding overfitting to a particular classifier.

The complete dataset comprises 10,000 executable files, representing a wide range of behavioral profiles, execution outcomes, and levels of classification complexity. The entire data generation process is illustrated in Figure 3, which traces the workflow from the initial acquisition of samples through to the final construction of the dataset suite. Executable files obtained from AnyRun and GitHub were first categorized as malicious, suspicious, or benign. Each file was subsequently executed within the AnyRun sandbox environment, where both dynamic system behaviors and static characteristics were recorded. Following execution, relevant features were extracted and then subjected to a rigorous filtering process to retain only those with clear diagnostic value. These refined features formed the basis for building structured, category-specific datasets that reflect the distinctive behavioral signatures associated with each malware type. The resulting datasets provide a high-quality

foundation for subsequent classification tasks and for detailed behavioral analyses aimed at advancing malware detection research.

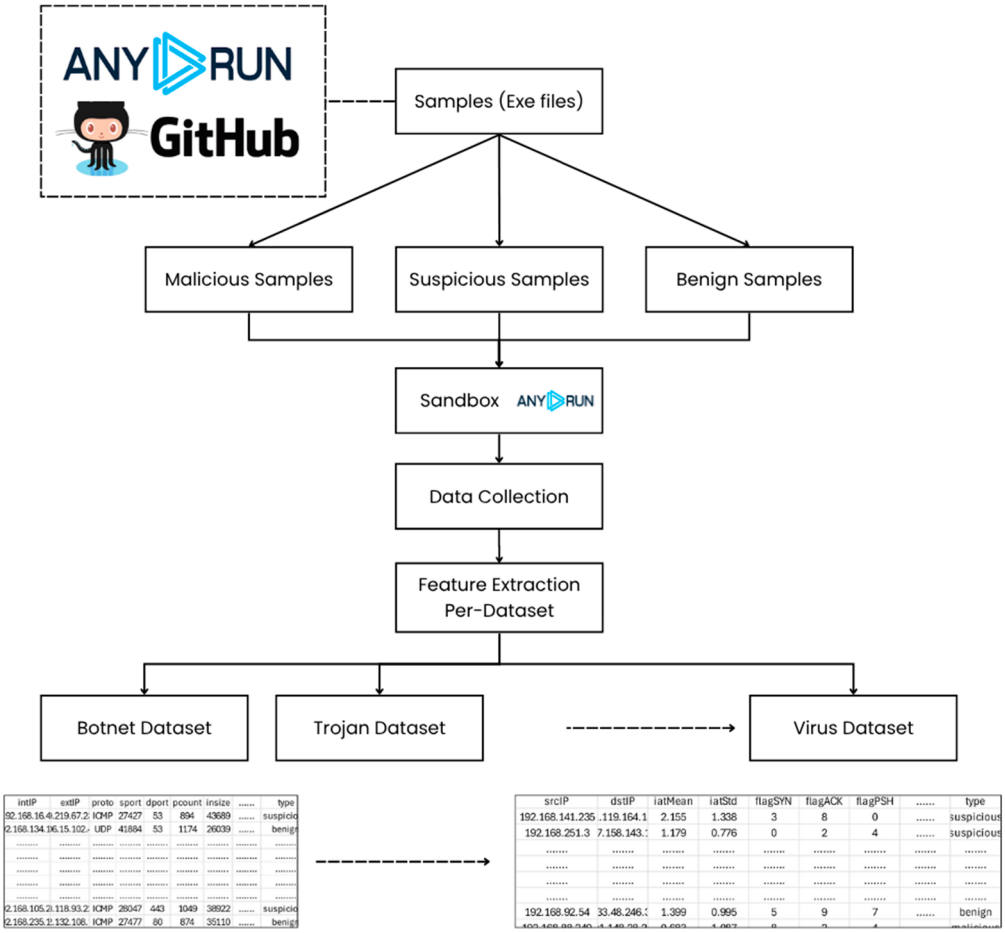


Figure 3. Workflow for constructing malware-specific datasets.

3.2.1. Trojan Dataset

Trojan is a kind of malware that misleads users as to its true intent by disguising itself as a normal program [73]. They generally spread by some form of social engineering. Many tend to contact one or more Command and Control (C2) servers across the Internet and await instruction [74]. Since individual Trojans typically use a specific set of ports for this communication, it can be relatively simple to detect them [75]. Moreover, other malware could potentially take over the Trojan, using it as a proxy for malicious action. Because of this, when building a dataset that is targeting Trojans, the features cannot be merely surface-level; they have to represent the operational elements that demonstrate the difference between what the program claims to be and what it actually does when it is run [76]. In the case of this dataset, features were pulled from multiple layers of the system. Network-level activity was considered, along with host-level events, file and registry changes, and API-level activity. The thought behind this is that Trojans have no single mode of operation and they tend to leave a trail in multiple places at once [77]. Figure 4 shows the end-to-end workflow employed, beginning with raw executable samples that were executed in the Anyrun sandbox. The sandbox produced behavioral telemetry that was then parsed and filtered so that only attributes of immediate Trojan activity importance were retained. The result was a dataset organized into five categories of features: network indicators, temporal features, TCP flag usage, process-level features, and API or registry activity. From these categories, 27 final attributes were established. This number offers

a balance between completeness and specificity. Features were presented to capture the variability in Trojan behavior, without introducing extraneous noise. The outcome is a dataset that highlights the characteristic set of signals, network traces on one hand and host- or process-level activity on the other, that together define the Trojan family.

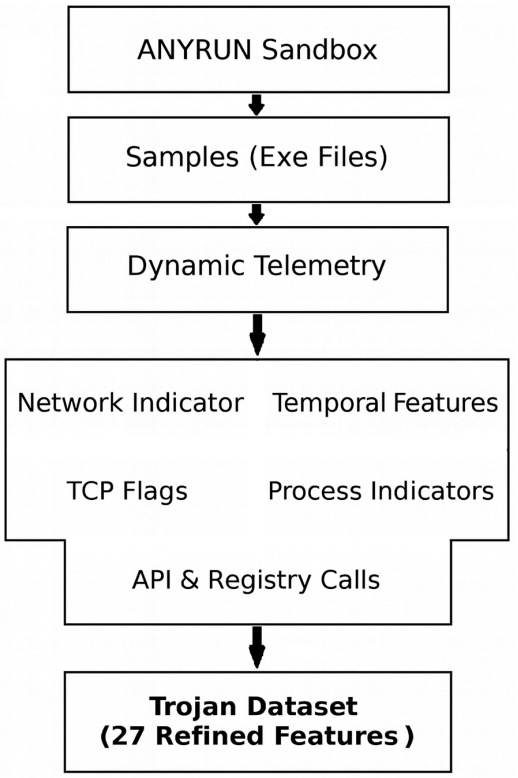


Figure 4. Trojan dataset generation pipeline from AnyRun sandbox telemetry.

From the network side, Trojans usually depend on command-and-control communication to stay active on the system and to move data out once it has been stolen [74]. For that reason, features such as source and destination IP addresses and ports were included, since they help capture where the traffic is going and where it is coming from. Other attributes, such as flow duration, the total number of forward and backward packets, and the byte rate of a flow, were used to reflect both the intensity and the direction of traffic, which in practice can uncover hidden exfiltration channels that are otherwise hard to notice. Timing features, for example, the mean inter-arrival time of flows, also provide useful evidence, because periodic beaconing is a very common sign of C2 traffic. Flag-level indicators were kept in the dataset, since abnormal session starts or unusual persistence in connections are often observed when Trojans maintain backdoors [78]. Put together, these network indicators capture both the stealthy traffic and the persistent patterns that mark Trojan communication.

On the host side, execution-level signals add another important dimension. Process identifiers (PIDs) and integrity levels were included because they show when a process escalates beyond normal user privileges. Exit codes also help, since they can indicate abnormal termination after a payload runs. TCP window features such as initial forward window bytes and simple counts of forwarded data packets were added too, because many Trojans show oddities in payload delivery when sessions are being established. Host-level details help distinguish malicious processes from the kinds of processes started by normal applications [79].

Beyond processes, filesystem and persistence behaviors are key for Trojans, since they often try to maintain a foothold. WRITE permissions and installation-related features map

to common behaviors such as dropping new binaries, creating autorun registry entries, or editing keys for persistence. Features that capture the manipulation of other processes, like terminating security tools or killing competitors, were also included. The point here is that Trojans often want to eliminate obstacles and remain active on the machine, so these persistence signals reflect that strategy [80]. At a deeper level, API features were used to capture what the Trojan actually does once it is running. For example, calls to the Runtime exec function suggest attempts to spawn hidden processes or run outside binaries. Dynamically loaded libraries and class loaders point to the Trojan's ability to insert new payloads even after the first stage is executed [81]. Cryptographic calls, such as Cipher's doFinal, show the encryption or decryption of exfiltrated data. The DefaultHttpClient send function maps directly to HTTP-based C2 traffic, which is still a preferred channel for many Trojans. API calls for system or package enumeration highlight the fact that Trojans often survey the environment before deciding what to do next. Taken together, these API-level interactions provide a code-level view of Trojan operations and make it possible to distinguish them from benign executables.

Finally, a feature correlation analysis was applied to the thirty features that were initially selected. A correlation matrix was used to check redundancy and interdependence. This step revealed that FlowDuration and FlowBytes/s were very strongly correlated with FwdPkts, so they added little new information. A third attribute, ReadSMS, showed no correlation with anything else and, more importantly, did not make sense in the PC Trojan context. To reduce duplication and avoid irrelevant features, these three were removed. The refined dataset therefore retained twenty-seven features, each one contributing something unique. Figure 5 shows the refined correlation matrix with only these twenty-seven features. The effect of this refinement step was to reduce noise, improve clarity, and make sure that the final Trojan dataset represents Trojan behavior instead of carrying unnecessary or misleading information.

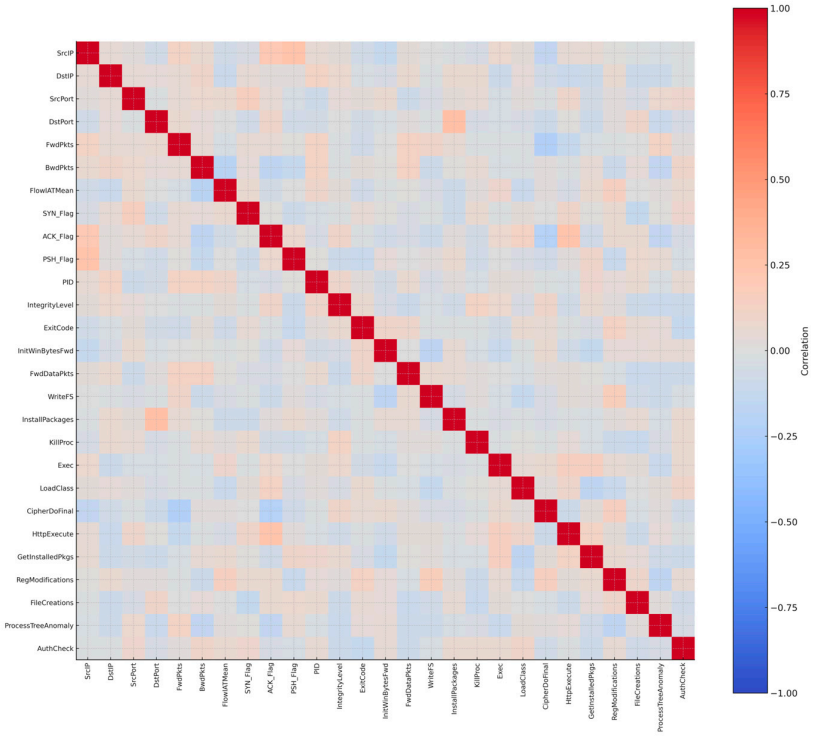


Figure 5. Correlation matrix of the 27 refined Trojan features.

3.2.2. Mirai Dataset (Botnet)

Mirai turns networked devices running Linux into remotely controlled bots that can be used as part of a botnet in large-scale network attacks. It primarily targets online consumer devices such as IP cameras and home routers [82]. How it spreads is easy enough but extremely effective in practice. It searches for devices left with default weak or unchanged credentials, takes them over, and then integrates those machines into a large-scale botnet [83]. Once established, the network can be used to create distributed denial-of-service (DDoS) attacks potent enough to crash servers and networks [84]. What is different here is that Mirai does not operate like a Trojan. A Trojan would normally try to camouflage itself by pretending to be legitimate software, but Mirai does not even try to camouflage. Instead, it reveals itself through constant scanning, rapid exploitation of exposed devices, and heavy network traffic that persists even once the device is under its control [85]. Mirai draws more on persistence and more on numbers, and less on stealth; this renders it vastly disruptive [86].

To capture these operational signatures in the dataset, twenty-five features were selected. These features came from several categories: network flows, packet statistics, temporal characteristics, and TCP-level indicators. The aim was to make sure both phases of Mirai's activity were represented—the propagation stage, where it spreads from device to device, and the attack stage, where infected devices are used for DDoS traffic. On the network side, source and destination IP addresses and ports were included. These show where the scanning traffic originates and where it is headed. Mirai infections typically generate very wide scans across the internet, and the unusual spread of source and destination ports is a clear marker of this behavior [87].

Flow statistics were also essential. Attributes such as flow duration, total forward packets, total backward packets, and the corresponding byte counts were included to measure both the direction and the overall volume of traffic. Because Mirai mixes very small scanning packets with much larger attack payloads, packet length statistics were added as well. These measures help reveal the irregular and burst-like nature of Mirai traffic, which stands out compared to the more uniform communication patterns of normal IoT devices [88].

Timing features play an equally important role. The mean, maximum, and standard deviation of flow inter-arrival times highlight the timing regularities associated with scripted scanning and coordinated attack traffic. Flow bytes per second and packets per second capture throughput levels that are typical of sustained flooding. Active and idle times reveal how infected devices cycle between scanning for new victims and taking part in attacks.

TCP-level indicators were also emphasized because Mirai frequently uses SYN floods and related amplification methods. Counts of SYN, ACK, FIN, and RST flags were included since they provide a direct measure of abnormal connection attempts and unusual termination behavior. Another useful feature was the ratio of downlink to uplink traffic, which captures the asymmetry often seen in DDoS traffic, where outbound flows dominate. Aggregate indicators such as average packet size, segment sizes in both directions, and packet length variance were selected to reflect the diversity of Mirai's packet composition. This combination highlights the way Mirai traffic alternates between short reconnaissance probes and larger bursts of data.

Taken together, these twenty-five features provide a well-rounded profile of Mirai's behavior. They balance fine-grained packet analysis with higher-level flow and session statistics, and they were validated using a correlation matrix. In contrast to the Trojan dataset, where a few features were found to be redundant, the Mirai dataset showed that all twenty-five attributes contributed unique information. None of them had an excessive

correlation with each other, and none were irrelevant. In other words, every feature added something useful for detection. The final set of twenty-five features is summarized in the correlation matrix shown in Figure 6.

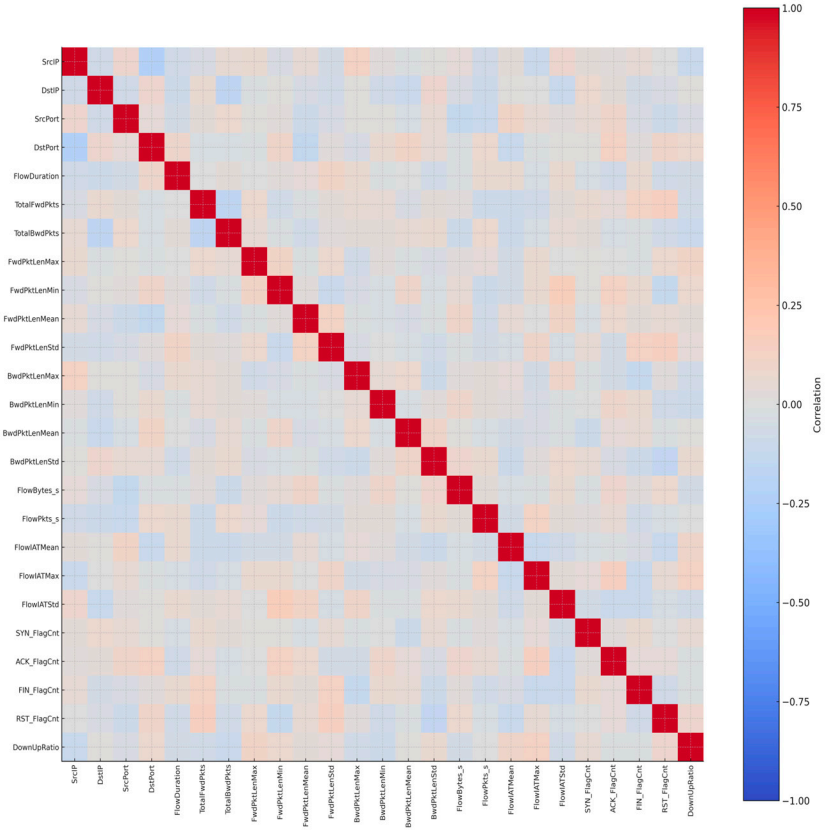


Figure 6. Correlation matrix of the 25 Mirai features.

To examine how the Trojan and Mirai datasets relate to one another, a comparative analysis of feature overlap was carried out. As shown in Figure 7, there are seven features that appear in both datasets: SrcIP, DstIP, SrcPort, DstPort, FlowIATMean, FlowIATMax, and FlowIATStd. These are basic network flow indicators, and their presence in both datasets is not surprising since communication endpoints and timing dynamics are fundamental for almost any type of malware traffic. The important point here is that since this small set of common features forms a kind of core, the two datasets diverge considerably beyond it.

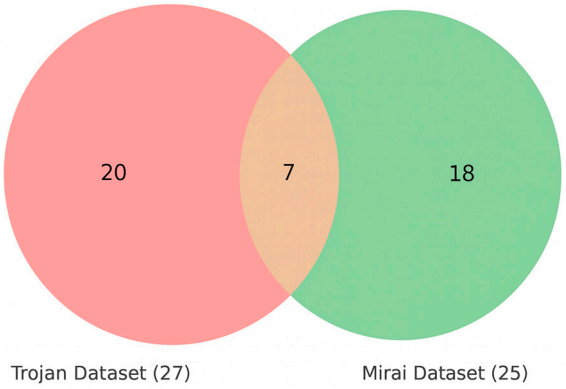


Figure 7. Feature overlap between Trojan and Mirai datasets.

The Trojan dataset leans heavily toward host-level and API-related attributes—things like process identifiers, registry edits, and runtime execution calls. These reflect the way Trojans often operate by escalating privileges locally, maintaining persistence, and executing payloads in a covert manner. The Mirai dataset, on the other hand, looks very different. It emphasizes packet-level statistics, counts of TCP flags, and throughput measures, all of which point to its focus on rapid scanning and high-volume distributed denial-of-service traffic.

This contrast shows why malware-specific feature engineering is necessary. If analysis were based only on generic network indicators, much of the unique behavior that distinguishes Trojans from Mirai would be missed. In other words, the overlap highlights the common ground, but the divergence demonstrates why tailored feature sets are required to capture the true signatures of different malware families.

3.2.3. Ransomware Dataset

Ransomware has become a major focus in cybersecurity due to the rapid escalation of attacks and the emergence of new variants designed to evade traditional antivirus and anti-malware defenses [20]. Although it is a comparatively recent form of malware, it has quickly gained popularity among cybercriminals because of its effectiveness and the direct financial incentives it offers. The core objective of ransomware is to deny victims access to their own resources, either by locking the operating system or by encrypting files that hold personal or business value, such as images, spreadsheets, and presentations [89]. The economic and operational impact is significant, as the devices that have been hijacked usually provide critical services, and downtime is loss directly translatable to money [90]. Unlike Mirai, which induces unavailability through enormous scanning and distributed denial-of-service traffic, ransomware directly interferes with the integrity and availability of a victim's data [91]. It also frequently deposits forensic artifacts at the system level as well as at the network level in the form of encryption processes, registry changes, and unusual process behavior [1].

To develop a dataset that correctly captures these operational signatures, thirty-five features were selected. They span several layers of activity: process runs, file system operations, registry modifications, packet streams, and network utilization. Together, they form a behavioral fingerprint that spans the necessary steps of ransomware activity—from encryption to persistence to communication with the attacker. The process of creating the dataset is summarized in Figure 8. Labeled ransomware instances were executed under dynamic sandbox environments and runtime properties were collected systematically.

For ease of administration, the attributes were divided into four broad categories: (i) network indicators, (ii) packet and timing statistics, (iii) process execution attributes, and (iv) file/registry activity with cryptographic API calls. This resulted in the thirty-five feature final ransomware dataset, which collectively describes both the destructive and stealthy features of ransomware behavior. At the network level, attributes such as source and destination IP addresses and ports were included to trace connections between infected hosts and attacker-controlled servers [92]. Flow duration, total forward and backward packets, and their associated byte counts were retained since they capture both the volume and the direction of ransomware traffic, particularly during key exchange or data exfiltration attempts. Flow bytes per second and packets per second were added to quantify throughput, while average packet size and packet length variance reflected the mix of small negotiation packets and larger bursts of file-transfer-like activity. Timing-based variables, including the mean, maximum, and standard deviation of inter-arrival times, were also chosen because they help reveal the irregular traffic caused when encryption routines run at the same time as command-and-control communication. On the host execution

side, process identifiers, integrity levels, and exit codes were prioritized. These features expose ransomware's tendency to escalate privileges, terminate processes abnormally, and repeatedly invoke encryption routines. Active and idle time statistics were also kept since they capture abnormal bursts of CPU-intensive activity followed by periods of inactivity, a cycle that commonly appears during batch file encryption. Initialization features, such as initial forward window bytes and active data packets sent, were preserved to track TCP session setup, which often deviates from the patterns seen in benign applications.

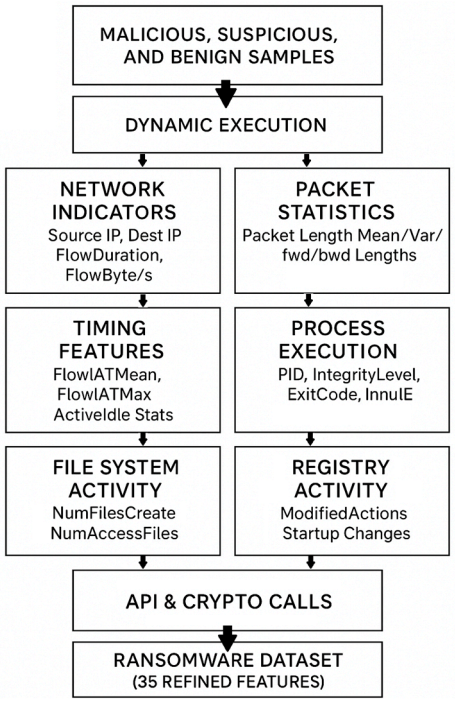


Figure 8. Construction workflow of the ransomware dataset.

File system and persistence indicators were equally important. Counts of file creations, access operations, and modifications were included to reflect the large-scale file activity typical of encryption campaigns. Registry-related attributes, such as write operations and system modifications, were retained to capture persistence techniques, where ransomware alters startup behavior or disables recovery utilities [21]. The point here is that these structural traces differentiate ransomware from other categories of malware that do not manipulate the file system as aggressively or as systematically.

Encryption and execution behaviors were represented by API-level features. Runtime execution calls, dynamic class loading, and cryptographic operations such as Cipher's doFinal were included to provide direct evidence of ransomware's use of built-in encryption libraries. These go beyond high-level file interactions and show the low-level mechanisms of rapid file transformation. Additional features tied to system alerts and control were also included, reflecting ransomware's ability to disable user input or interfere with graphical interfaces, which prevents victims from halting the encryption process mid-execution [93].

The final set of thirty-five features was validated with a correlation matrix to check for redundancy. Unlike the Trojan dataset, where overlapping attributes had to be removed, the ransomware dataset did not show evidence of highly correlated or irrelevant variables. Each attribute formed meaningful relationships with others without duplication, confirming its value for profiling ransomware behavior. Figure 9 presents the correlation matrix, which illustrates the complementary nature of the chosen features. The refinement process demonstrated that the dataset preserves the multi-layered profile of ransomware—from

network anomalies to file system events and cryptographic traces—providing a reliable basis for both detection and analysis.

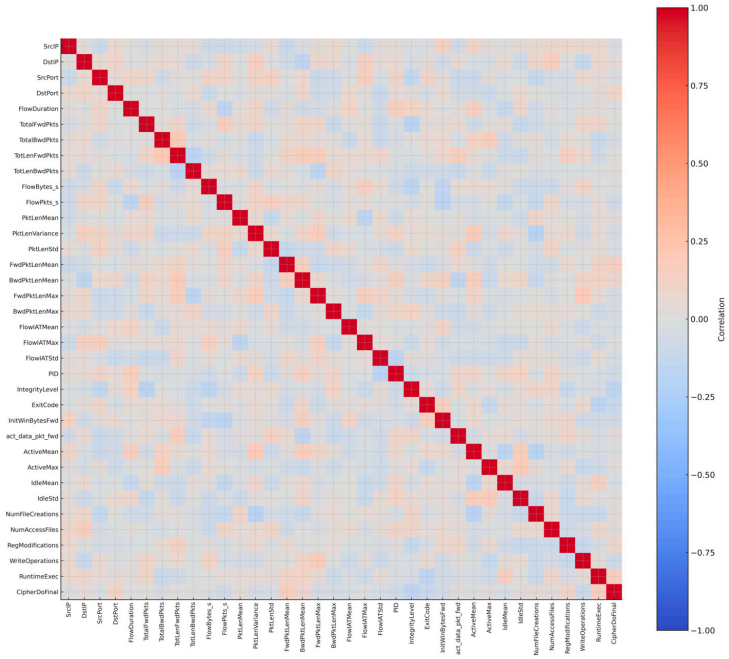


Figure 9. Correlation matrix of the 35 ransomware features.

3.2.4. Rootkit Dataset

A rootkit refers to a set of malicious software components developed to grant unauthorized users privileged access to a computer system or restricted areas of its software. At the same time, it commonly conceals its own presence, or the presence of other malicious programs, to avoid detection [94]. Modern rootkits are not primarily used to gain elevated access; instead, their role is to conceal another software payload by providing stealth capabilities [95]. They are generally classified as malware because the hidden payloads they accompany are themselves malicious [96]. Rootkits use a range of techniques to gain control over a system, and the specific method often depends on the type of rootkit involved [97]. One of the most common approaches is the exploitation of security vulnerabilities to achieve hidden privilege escalation.

In order to capture this kind of behavior in a dataset, twenty-two features were selected [98]. These include signals of kernel manipulation, hidden process activity, illegitimate privilege escalation, and persistence methods. Taken together, they map the structural stealth strategies that make rootkits so dangerous: the ability to keep running quietly in the background, to interfere with system-level functions, and to evade not just casual observation by the user but also many standard security tools [99].

At the process and execution layer, attributes such as PID, IntegrityLevel, and ExitCode were retained because they help identify processes that are running with elevated or otherwise abnormal privileges and that terminate unexpectedly. This is important since rootkits often hide their activity by embedding themselves within legitimate system processes, and execution-related features can reveal those irregularities [98]. MinSegSizeFwd and ActDataFwd were also included, since rootkits frequently rely on process injection or process hollowing, which leaves irregular segmentation patterns in process-linked communications. In other words, process-level attributes make it possible to observe the ways rootkits exploit existing system processes to remain hidden [98]. Registry and persistence manipulation were central features in the dataset. Attributes such as RegMods, WriteOps, and DelPkgs were included to capture the ways rootkits alter registry keys and remove ar-

tifacts left behind by detection or monitoring tools. SuperUserAccess and BindAdmin were prioritized because rootkits often take advantage of administrative privileges to ensure they maintain a persistent foothold on the system. Similarly, AlertWindow and DisableKeyguard reflect the ability of rootkits to override system defenses or tamper with user-facing security controls, actions that allow them to remain active even in hardened environments. Kernel- and API-level hooks were also emphasized. RuntimeExec, SysLoad, and DexLoadClass were selected because they capture the dynamic loading of concealed modules or the injection of malicious code into system memory. These behaviors are significant since rootkits depend on dynamic linking and runtime execution calls to remain concealed from standard inspection tools [100]. InjectEvents and CrossUserInteract were kept as well, as they indicate the capability of rootkits to manipulate user sessions and input streams, often sidestepping normal permission boundaries. System service-related features were incorporated to highlight how rootkits tamper with monitoring and maintenance processes. DevicePower, KillProcs, and UpdateStats were chosen because rootkits frequently disable system logging, terminate security services, or falsify statistics about system resource usage. ReadLogs was added for a similar reason, since rootkits often clear or modify log entries to erase traces of their presence. Finally, MasterClear and Reboot were retained because they serve as indicators of destructive actions, where a rootkit wipes data or forces a restart as part of its concealment or persistence strategy [98]. These twenty-two features describe the structural stealth and system-subverting activity that define rootkits, in contrast to the outward scanning behavior seen in botnets or the encryption activity typical of ransomware. To verify the quality of the feature set, a correlation matrix was applied. This analysis confirmed that none of the features were redundant or irrelevant; instead, each one provided complementary insight into different aspects of rootkit behavior—whether hidden process execution, registry manipulation, kernel hooking, or service abuse. Figure 10 shows the resulting correlation matrix, illustrating that the selected features together offer a comprehensive behavioral profile of rootkits.

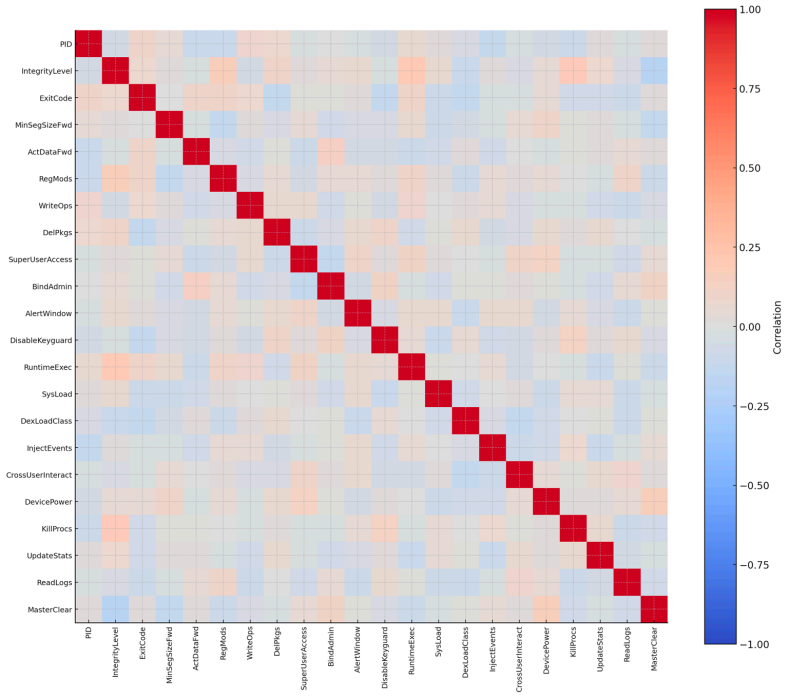


Figure 10. Correlation matrix of the 22 rootkit features.

3.2.5. Worm Dataset

Worms are self-replicating malware that use system or network vulnerabilities to multiply and propagate automatically from device to device, usually without user interaction [101]. Their signature behaviors include aggressive vulnerable target scanning [102], brute-force entry points, and establishing many concurrent outbound connections [103]. Unlike ransomware or Trojans, whose objectives are payload execution, bulk replication and lateral movement are the goals of worms [104]. Although worm epidemics were the norm in the early 2000s, contemporary worm binaries remain uncommon since operating system development, patch cycles, and the advent of even more stealthy fileless attack techniques have reduced their circulation. Most worms are designed to erase themselves or alter, and very few enduring samples exist to analyze. On the other hand, monetized households like Trojans or ransomware are still utilized and repackaged by actors for financial reasons, which explains their greater availability in sandbox repositories [105].

To overcome the scarcity of worm samples, this dataset was constructed using a hybrid sourcing approach. The majority of binaries were collected from the AnyRun sandbox archive, supplemented by curated collections of historical and open-source worm implementations hosted on GitHub [22,55–62]. These repositories were critical for diversifying coverage of propagation strategies, ranging from older SMB-exploiting worms to more modern network-scanning variants. The potential of image-based datasets as a supplement to numerical datasets has been investigated by researchers [106], as this approach can greatly help address the insufficiency of malware data—particularly for types such as worms and keyloggers, which are not available in large numbers. However, this study, aiming for a simpler pipeline design, focuses solely on numerical dataset generation.

Four independent repositories were incorporated, ensuring that the dataset captured both legacy and current worm tactics. The final dataset consists of 1250 total samples, including 350 confirmed worm executables, 200 suspicious files, and 700 benign samples. Suspicious files displayed partial propagation-like behavior, such as initiating port scans or broadcasting to multiple IP addresses, without conclusive evidence of replication. Benign programs were selected from multi-threaded network services and software updaters, which generate superficially similar traffic patterns but are non-malicious.

Feature selection focused on capturing the burst-driven, fan-out communication patterns that define worm propagation. Core network metrics such as SrcBytes, DstBytes, SrcPkts, and DstPkts quantify session asymmetry, reflecting the heavy outbound bias typical of scanning worms. Attributes such as ScanFanOut, RepeatDstIPs, and UniquePorts measure the diversity of targets and ports accessed within short intervals, directly highlighting the wide attack surface probed during propagation. To quantify connection reliability, FailedConnRate, ScanToConnectRatio, and ConcurrentConns were included, exposing brute-force behaviors where multiple failed attempts precede a successful compromise [107].

Temporal dynamics were equally important for profiling worms. Features such as FlowIATMean, FlowIATStd, and BurstInterval reveal the sudden bursts of outbound packets associated with automated scanning. Jitter, RTTVar, and InterFlowGap further capture irregular delays and repeated connection attempts thatdistinguish worms from steady client-server traffic. To reflect protocol abuse and evasion, MultiProtoSwitch, SYNRate, and RSTDensity were selected, since worms often switch between protocols or generate excessive flag activity in attempts to bypass intrusion detection systems [108].

Entropy-based features provided another lens focusing on worm activity. DstEntropy measures the randomness of destination IPs, highlighting the indiscriminate nature of scanning. PortEntropy quantifies the diversity of ports accessed, distinguishing worms from benign applications that repeatedly use a fixed service port. PayloadEntropy was also

included, as polymorphic worms frequently randomize payloads to avoid signature-based detection. Together, these attributes quantify the unpredictability and randomness that worms use to frustrate static detection methods.

Finally, throughput-related attributes such as TransRate and AggBytesRate capture the bandwidth consumed during replication bursts. Worms often push large volumes of packets in short periods, producing distinguishable throughput signatures [105]. By combining connection intensity, temporal irregularity, entropy measures, and brute-force indicators, the selected 25 features create a dataset capable of differentiating worms from benign and other malware families. To ensure that the feature set was both representative and non-redundant, a correlation matrix was applied to the selected attributes. Figure 11 presents the correlation matrix, showing that the chosen attributes collectively capture the diversity of worm propagation strategies.

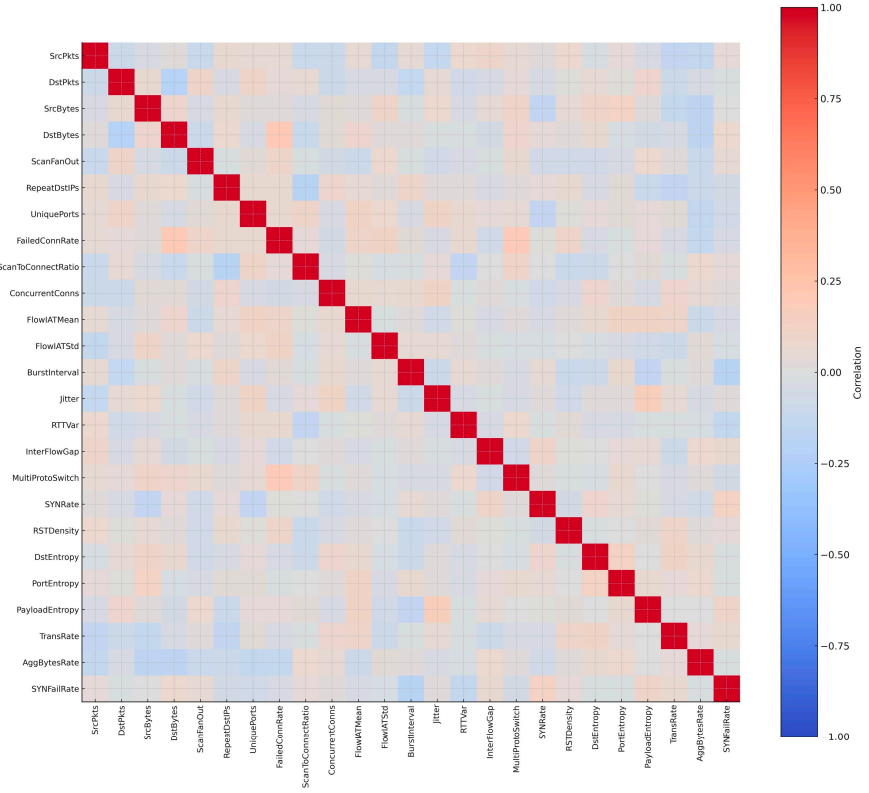


Figure 11. Correlation matrix of the 25 worm features.

To better understand the degree of commonality across malware categories, a cross-dataset comparison of feature usage was performed. As shown in Figure 12, certain attributes recur across multiple families, reflecting baseline system or network behaviors that are widely exploited. For example, core network timing metrics such as FlowIATMean, FlowIATStd, and FlowIATMax are present in the Trojan, Mirai, ransomware, and worm datasets, highlighting their universal value in capturing traffic irregularities across different propagation and execution strategies. Similarly, host-level indicators including PID, IntegrityLevel, and ExitCode appear in several datasets, underscoring the role of abnormal process behavior as a consistent signal of compromise. System manipulation variables such as RegMods and RuntimeExec are also shared by multiple families, representing persistence and dynamic execution behaviors that cut across malware types. It is clear that each dataset also incorporates distinct, family-specific attributes. Mirai emphasizes packet-level counts and flag densities associated with scanning and DDoS, while ransomware introduces cryptographic and file-access features. Rootkits are dominated by stealth-oriented

attributes such as registry subversion, log manipulation, and API hooking. Worms, by contrast, prioritize fan-out scanning, entropy measures, and aggressive outbound connection rates. This balance of shared and unique features demonstrates both the common foundation of malicious behavior and the family-specific strategies that necessitate tailored feature engineering.

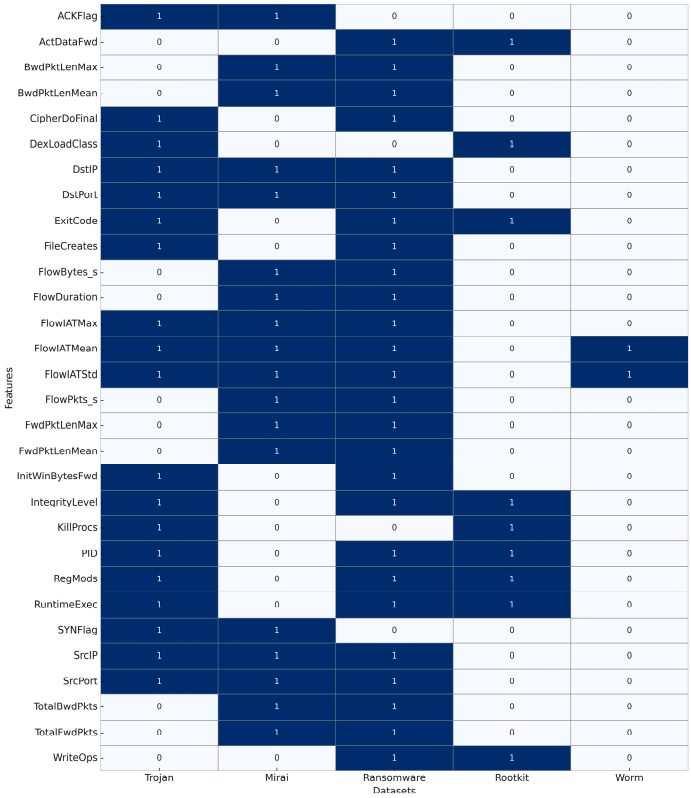


Figure 12. Heatmap of feature overlap across five malware datasets.

3.2.6. Spyware Dataset

Spyware refers to malicious software designed to collect information about an individual or organization and transmit it to a third party, typically in ways that compromise user privacy, weaken device security, or cause other forms of harm [109]. Similar behaviors can also be found in other types of malware and, in some cases, even in legitimate software. For example, websites may engage in tracking practices that resemble spyware activity, and hardware devices can also be affected [110]. Spyware is often linked to advertising, sharing many of the same privacy and security concerns. However, because these behaviors are widespread and can sometimes serve non-malicious purposes, drawing a precise boundary around what qualifies as spyware remains a challenging task [111]. Given that spyware is often run in user space rather than installing at the kernel level, its indicators of compromise tend to be more evident at the application level hooks, stealthy process behavior, and unusual network traffic [112]. IoT cybersecurity overlaps with fields like customer analytics, as both rely on understanding behavioral context to extract meaningful insights. Whether analyzing human communication patterns or monitoring the actions of connected devices, both domains require intelligent systems that can interpret subtle signals, adapt to variability, and generate reliable insights for prediction, defense, and informed decision-making [113,114].

The spyware dataset consists of 1250 total samples: 300 confirmed spyware executables, 300 suspicious programs, and 750 benign samples. Suspicious files demonstrated partial evidence of spyware-like behavior, such as initiating covert network beacons or

attempting screen capture, without establishing full persistence. Benign programs were carefully selected from legitimate utilities such as cloud synchronization tools, messaging applications, and screen-sharing software, which produce superficially similar behaviors but are non-malicious. Figure 13 illustrates the operational flow of a spyware program, highlighting its initialization, surveillance modules, and covert communication. The architecture demonstrates how spyware captures user activity, disguises its persistence through system modifications, and funnels data through obfuscated network channels to external payload servers.

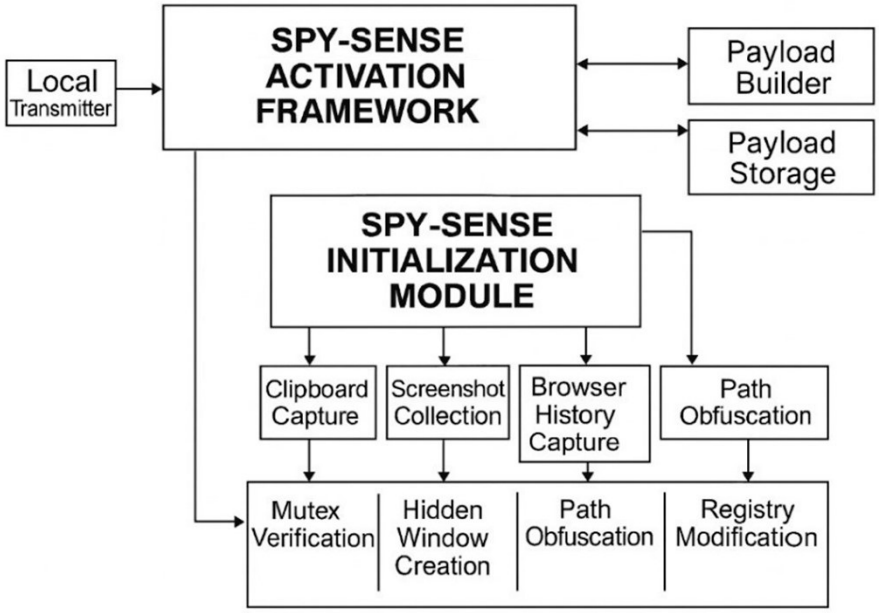


Figure 13. Spyware architecture.

Feature selection emphasized attributes that capture both surveillance actions and network-based stealth channels. From the surveillance perspective, variables such as ClipboardAccess, ScreenshotTrigger, BrowserHistoryExport, KeyloggerFlag, and WindowFocusEvents were included to reflect unauthorized interception of user data. These features highlight the programmatic harvesting of visible and hidden user interactions. Complementary API-related features such as GetAsyncKeyState, SetWinHook, GetForegroundWin, and ShellExecCall capture lower-level function calls frequently invoked by spyware for keystroke logging, active window monitoring, and covert process launching.

The dataset distinguishes spyware from other families by including a large set of network activity features, reflecting its reliance on exfiltration. Attributes such as HTTPBeaconRate, DNSLeakEvents, POSTRequestFreq, ExfilPktRate, and DataUploadSize were selected to detect silent data transfer channels. Features like C2ConnCount, KeepAliveBeacon, and ProxyBypassAttempts measure stealthy command-and-control traffic patterns. To detect obfuscation during transmission, PayloadEncFlag, HeaderAnomalyRate, and PktTimingJitter were introduced, highlighting evasive encryption or randomized traffic bursts.

Persistence and concealment indicators were also included. Features such as MutexActivity, HiddenWindowRate, InstallPathObf, and RegPersistenceKey capture spyware's tendency to mask itself in legitimate system folders and maintain automatic startup execution. Additional behavioral indicators such as ParentProcMismatch, NonInteractiveThreadRate, and IdleSessionExfilRate highlight adaptive evasion tactics where spyware hijacks legitimate processes, spawns hidden threads, or waits for idle periods to transmit stolen data [115].

In total, 39 features were refined to provide a comprehensive representation of spyware operations. The dataset integrates network telemetry (beaconing, exfiltration, and traffic anomalies) with user surveillance events (clipboard, screenshots, keystrokes) and stealth techniques (mutexes, registry persistence, hidden execution). As shown in Figure 14, the analysis confirmed that all variables contributed unique behavioral information, with no excessive redundancy or isolation. This validation step ensures that the dataset captures the subtle but critical signals of spyware activity without introducing noise into the model training process.

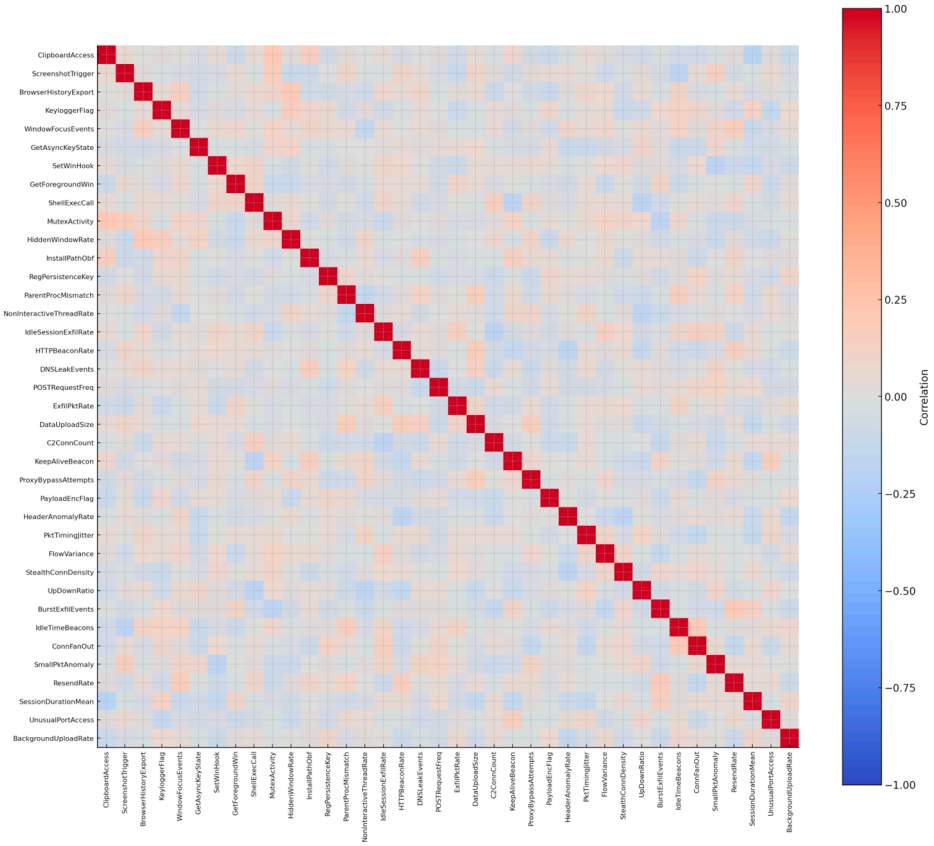


Figure 14. Correlation matrix of 39 spyware features.

3.2.7. Keylogger Dataset

Keyloggers are tools that capture and record the sequence of keystrokes entered on a keyboard, usually in a hidden manner so that the user remains unaware of the monitoring activity [116]. The recorded information can later be retrieved and reviewed by the operator of the logging system [117]. Keystroke logging may be implemented through either hardware devices or software applications. Although certain keylogging programs are legally distributed—for example, to allow employers to supervise workplace computer usage—the technology is more commonly linked to malicious purposes, such as stealing login credentials and other sensitive data [118]. At the same time, keyloggers are sometimes employed in non-criminal contexts, including parental monitoring, classroom supervision, or law enforcement investigations into unlawful computer activity. Beyond security-related applications, keystroke logging has also been used in research settings, such as analyzing keystroke dynamics or studying patterns of human–computer interaction. A variety of approaches exist, ranging from dedicated hardware loggers and software-based systems to more advanced methods like acoustic analysis of typing sounds [119,120].

The keylogger dataset introduced in this study consists of 1250 executable files: 300 confirmed keylogger binaries, 300 suspicious files, and 650 benign programs. Sus-

picious files showed partial traits of keylogging behavior, such as API hook initialization or abnormal keyboard event monitoring, without evidence of persistent logging. Benign programs were selected from legitimate applications such as word processors, text editors, and input testing tools, which naturally interact with keystrokes but lack covert collection or transmission functions [121]. This balanced composition ensures reliable training and validation while reflecting realistic threat distributions. Figure 15 illustrates the flow of a typical keylogger attack. The diagram shows how user input is intercepted at the system level, logged in hidden storage, and transmitted through stealthy outbound connections to an external attacker, highlighting the lifecycle from keystroke interception to data theft.

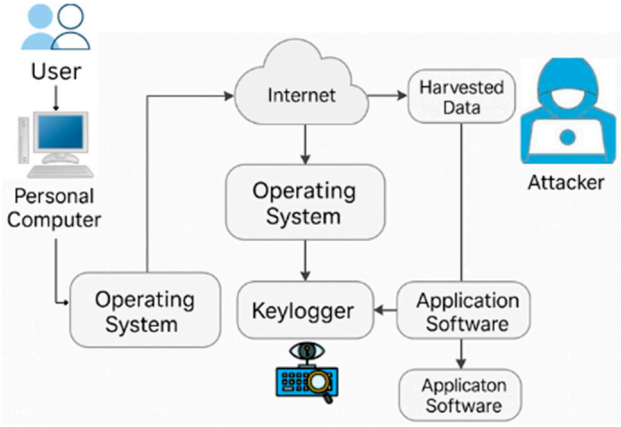


Figure 15. Keylogger attack flow.

Feature selection emphasized the mechanisms by which keyloggers intercept, record, and exfiltrate user input. Core attributes such as KeyboardHookInit, HookDLLLoad, InjectedProcFlag, ThreadInjectionMode, and HookDuration capture how malicious programs attach themselves to system input streams and processes. Behavioral indicators such as KeyStrokeBufferSize, HiddenLogFilePath, and OutputFileWriteRate reflect how captured data is stored locally, often using concealed directories or irregular write operations. To capture covert exfiltration, LogUploadRate, StealthConnFlag, and BeaconInterval were included, representing the silent transmission of logged data to external servers.

Timing- and rhythm-based features were also incorporated, as they are essential for distinguishing keyloggers from benign input-heavy applications [122]. Attributes such as InterKeyDelayMean, InterKeyDelayVar, KeyFrequencyDeviation, and TypingEntropy reveal anomalies in typing rhythm that arise when keystrokes are intercepted by a logging layer. Features such as ActiveAppContext, FocusWindowShiftRate, and UIThreadMismatch further highlight mismatches between expected user activity and background monitoring, exposing instances where keyloggers track keystrokes across multiple processes or hidden windows.

Additional system-level indicators were chosen to capture persistence and evasive behavior. RegPersistenceKey, MutexCheckFlag, and ParentProcMismatch detect common concealment methods such as registry-based startup entries, mutexes preventing duplicate execution, and anomalous process hierarchies. Indicators such as LowVisibilityThreads, IdleTimeCaptureRate, and NonInteractiveThreadRatio capture the use of hidden or idle execution contexts to avoid detection by monitoring tools.

In total, 26 refined features were selected to represent the behavioral signature of keyloggers (Figure 16). This dataset differs from traditional spyware datasets by focusing specifically on the keyboard input capture pipeline—from hook initialization, to keystroke timing analysis, to hidden storage and eventual exfiltration. By isolating these behaviors, the dataset enables the fine-grained detection and classification of keylogger activity, which

is often lost in broader spyware groupings. This level of granularity is especially critical in Internet of Things environments, where devices typically lack robust input monitoring or antivirus protections, making them prime targets for persistent surveillance.

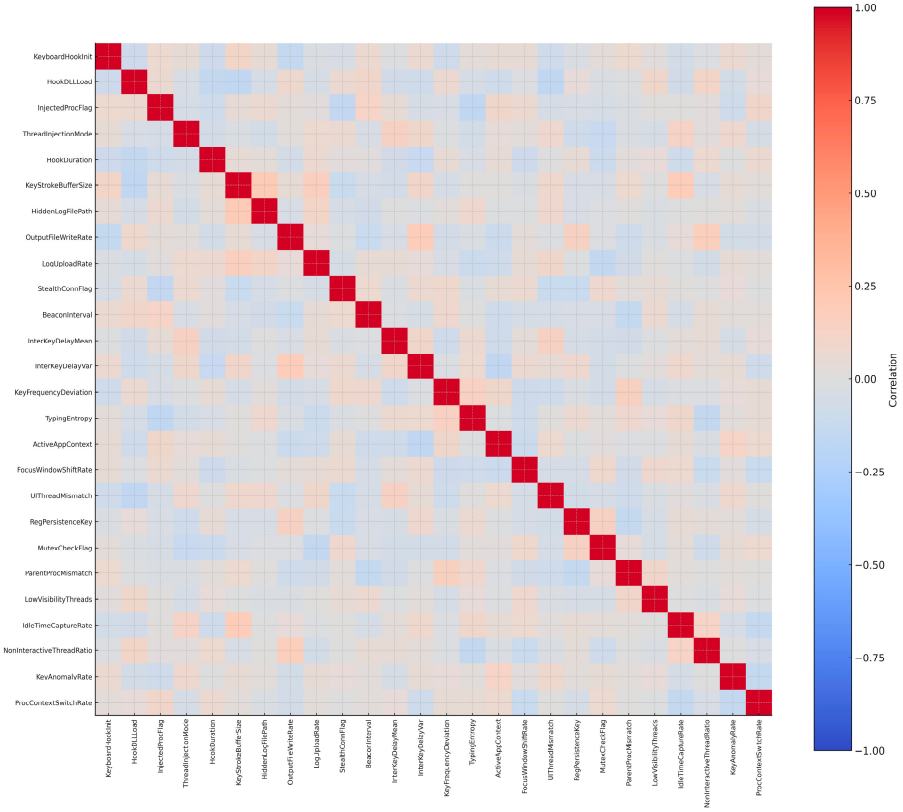


Figure 16. Correlation matrix of 26 keylogger features.

3.2.8. Virus Dataset

Computer viruses are self-replicating code segments that find their way into host files or processes and spread by changing executables or typical system resources [123]. Unlike worms, which scan networks for vulnerable entry points, viruses depend on a host program and typically require some form of user invocation to become active [74]. When triggered, they can modify or overwrite files, modify boot records, or interfere with normal system processes [124]. Modern virus families frequently employ polymorphic code and stealth techniques, making the older signature-based detection increasingly ineffective and necessitating profiling methods that emphasize runtime behavior [125].

The virus dataset constructed in this study contains 1250 samples: 400 confirmed malicious viruses, 200 suspicious binaries, and 650 benign executables. Malicious samples were collected from AnyRun sandbox telemetry and curated repositories on GitHub [22,55–62], providing both contemporary and legacy variants. Suspicious samples were drawn exclusively from AnyRun, reflecting executables that displayed partial viral traits such as abnormal file modifications or replication-like behavior, but without full confirmation of payload activity. Benign programs were selected from installers, utility software, and system tools that may mimic some structural characteristics of viruses but do not contain self-replicating code. This sampling strategy ensured that all binaries were executed and analyzed under uniform sandbox conditions, capturing consistent telemetry while also presenting challenging edge cases for classification.

Feature selection was guided by the defining behaviors of viruses: host infection, replication, persistence, and evasion. Indicators of file infection include FileOverwriteEvents, InjectedSectionsCount, and EXEHeaderAnomalies, which reveal tampering with executable

structures and unexpected code injection into legitimate binaries [126]. Registry-related variables such as RegistryHiveModFlag highlight unauthorized persistence mechanisms, while AutorunEntryInsertion and WriteToProgramFilesDir capture viral strategies to reinitiate upon reboot or spread through shared system directories.

Replication signatures were represented by features like DuplicateProcessForking, ExecutablePayloadDrop, and SpawnLoopSignature, each reflecting viral attempts to duplicate processes or drop infected executables for lateral spread. Structural irregularities such as ExecutionChainDepth and ParentPIDMismatch were included to differentiate viruses from benign installers that spawn multiple processes in predictable hierarchies. To account for polymorphic and evasive behaviors, features such as PESectionShuffling, EntropyVariance, and SelfChecksumBypass were integrated. These reflect common obfuscation strategies designed to bypass both static inspection and runtime anomaly detection.

By isolating viruses into their own dataset and collecting detailed telemetry specific to replication and infection, this study addresses a limitation in many public datasets that group viruses alongside Trojans or worms. The applications of deep learning models are extensively covered in the literature [127] and this virus-specific dataset provides a benchmark for evaluating deep learning classifiers that must capture file tampering, replication loops, and stealthy persistence, offering a foundation for the specialized detection of polymorphic and self-replicating malware. Figure 17 illustrates the pipeline for constructing the virus dataset, outlining the sourcing of binaries, sandbox execution, and extraction of infection- and replication-specific features.

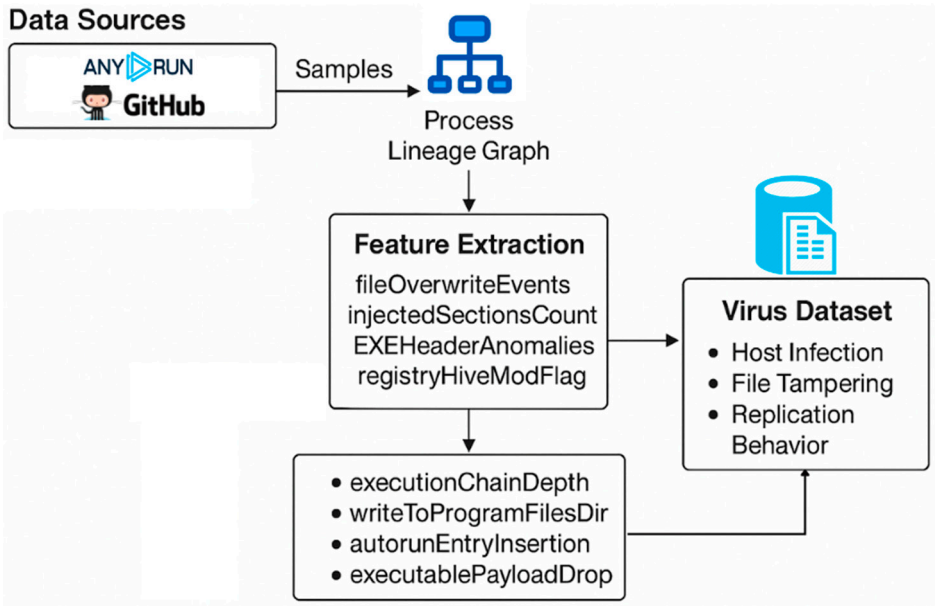


Figure 17. Virus dataset pipeline.

4. Results and Discussion

Each malware-specific dataset was built with a deliberate mix of malicious, suspicious, and benign samples, and the target size was set at 1250 samples for each family. The reasoning here was to strike a balance, providing enough data to support statistical robustness, while still being small enough to preserve interpretability at the class level. Figure 18 illustrates the class distribution of all samples in each of the eight malware datasets. The point here is that, in real detection work, uncertainty is common, and suspicious samples represent those edge cases where activity looks abnormal but does not provide absolute proof of a malicious payload. Suspicious files are often discarded in traditional binary-labeled datasets, but in this work they were retained to strengthen model robustness. By including

samples that only trigger partial feature activation, the datasets force classifiers to learn from ambiguous situations. In practice, this design choice makes detection models better prepared to handle the uncertainty and incomplete information that analysts regularly face in real-world cybersecurity environments.

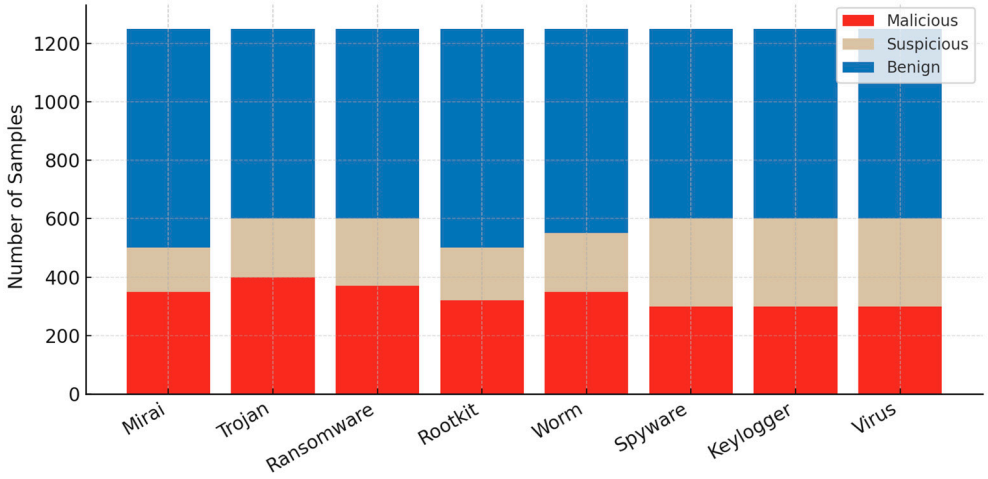


Figure 18. Class distribution of malicious, suspicious, and benign samples in each of the eight malware datasets.

Across the eight constructed malware datasets, a total of 213 features were identified, covering network-level indicators, process execution attributes, file and registry manipulations, timing statistics, API calls, and cryptographic or obfuscation markers. A comparative analysis of feature distribution revealed that 42 features were repeated across two or more datasets, while the remaining 171 features were globally unique across all eight datasets. This balance highlights the dual nature of the feature design: a shared baseline of malicious activity present across multiple malware types, alongside distinct attributes that capture the specialized behaviors of each category.

The overlap between datasets underscores common strategies that malware families employ. For instance, the Trojan and Mirai datasets share seven features—source and destination IP addresses, source and destination ports, and flow inter-arrival time mean, maximum, and standard deviation—reflecting their mutual reliance on network communication and timing anomalies as key indicators. Trojan, Mirai, ransomware, and worm datasets all include the trio of FlowIATMean, FlowIATMax, and FlowIATStd, making these timing-based variables universal markers of abnormal traffic. Host-level indicators such as PID, Integrity Level, and Exit Code recur in several datasets, capturing the abnormal process behaviors that characterize many malware families. Registry manipulation and execution flags such as RegMods and RuntimeExec are shared between Trojan, rootkit, and ransomware datasets, aligning with their reliance on persistence and dynamic code injection. Spyware and keylogger datasets show strong intersection, sharing features such as GetAsyncKeyState, SetWinHook, GetForegroundWin, and ShellExecCall, along with persistence markers like RegPersistenceKey and Mutex-related attributes, reflecting their shared emphasis on user surveillance and stealth. Virus datasets overlap with rootkit and ransomware through features like RegistryHiveModFlag, AutorunEntryInsertion, ParentPIDMismatch, and EntropyVariance, capturing infection persistence and polymorphic evasion strategies.

Alongside these overlaps, the 171 globally unique features provided each dataset with a distinct behavioral fingerprint. Trojan retained features exclusive to runtime execution calls and encrypted communication, while Mirai contributed attributes highlighting TCP flag anomalies and DDoS throughput. Ransomware contained unique features such as

cryptographic API calls and large-scale file modification indicators. Rootkit provided stealth-specific features, including DisableKeyguard, MasterClear, and log manipulation. Worm included propagation-oriented attributes such as ScanFanOut, PortEntropy, and PayloadEntropy. Spyware offered surveillance- and exfiltration-specific indicators such as ClipboardAccess, ScreenshotTrigger, and HTTPBeaconRate. Keylogger retained attributes such as HookDLLLoad, TypingEntropy, and HiddenLogFilePath, while virus contributed features reflecting file infection and polymorphism, including EXEHeaderAnomalies, PE-SectionShuffling, and SelfChecksumBypass.

This distribution demonstrates that while 42 features provide a shared baseline across multiple families, the majority of features are unique, ensuring that each dataset maintains its own behavioral identity. The result is a framework that captures both the commonalities of malicious behavior and the specialized strategies that define each malware type, enabling detection models to recognize universal indicators of compromise while also distinguishing between family-specific tactics. The comparative feature analysis across datasets revealed several important patterns of overlap. As shown in Figure 19, the heatmap of feature overlap highlights both expected and family-specific intersections. Notably, the Mirai botnet and ransomware datasets share a meaningful number of features. This similarity arises because both families generate distinct and high-volume network traffic during execution: Mirai through distributed denial-of-service floods and aggressive scanning, and ransomware through simultaneous file encryption operations and outbound command-and-control communications. In both cases, attributes such as flow duration, packet length statistics, and throughput measures capture the irregular traffic surges that distinguish malicious sessions from benign baselines.

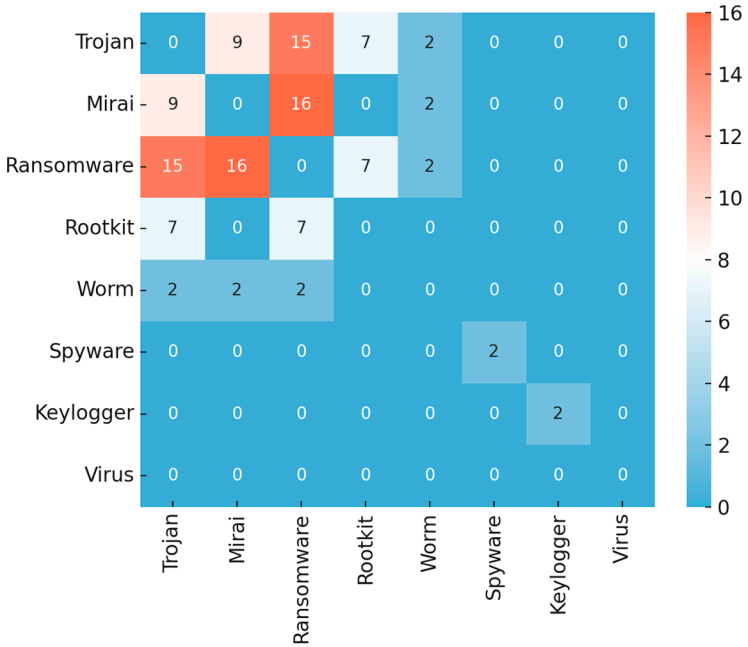


Figure 19. Feature overlap across malware datasets.

The overlap demonstrates that while these malware families have very different objectives—service disruption in the case of Mirai and data extortion for ransomware—they rely on parallel patterns of network abuse that can be captured with shared telemetry. At the same time, the heatmap confirms that other intersections, such as between Trojan and rootkit or spyware and keylogger, reflect host-level and persistence-related similarities rather than purely network-driven ones. This balanced distribution of overlaps illustrates the dual role of common signals and family-specific behaviors in building robust detection datasets.

To consolidate the feature importance analysis across all eight datasets, a global ranking was produced using Random Forest Gini scores. The results, presented in Figure 20, highlight a clear stratification between features that act as universal signals of malicious activity and those that are unique to specific families. At the top of the ranking, attributes such as Fwd Avg Bytes/Bulk, FlowID, SrcIP, and DstIP dominate, reflecting the central role of flow volume and endpoint information in distinguishing malware traffic from benign sessions. These features consistently reduce impurity across diverse malware categories because abnormal packet aggregation, unusual flow identifiers, and irregular communication endpoints are common hallmarks of malicious behavior.

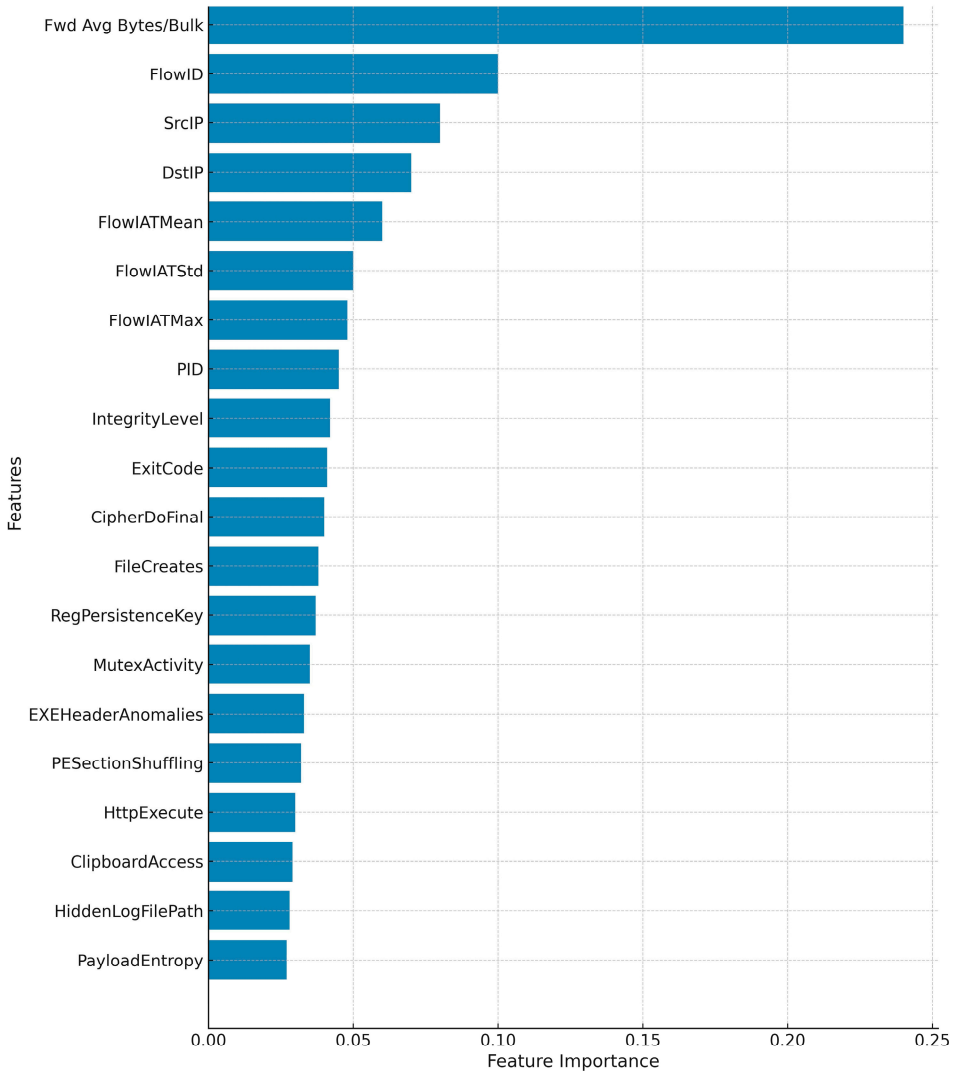


Figure 20. Feature importance across all datasets.

Timing-based indicators such as FlowIATMean, FlowIATStd, and FlowIATMax also rank prominently, confirming that irregular inter-arrival patterns are universal across families including Trojan, Mirai, ransomware, and worm. Likewise, process-level variables such as PID, IntegrityLevel, and ExitCode remain critical for capturing abnormal execution and privilege escalation events, providing cross-family discriminative strength.

In addition to these repeated features, the figure underscores the contribution of family-specific attributes. CipherDoFinal and FileCreates increase in importance within ransomware, while RegPersistenceKey and MutexActivity emerge as leading spyware and keylogger indicators. Similarly, EXEHeaderAnomalies and PESectionShuffling rank highly in virus detection, reflecting polymorphic code obfuscation strategies. This combination

of repeated and unique features demonstrates how detection models benefit from both universal and specialized signals: universal indicators provide robustness across malware categories, while unique attributes allow models to fine-tune predictions at the family level.

By incorporating suspicious samples alongside malicious and benign, the datasets simulate real-world forensic complexity and improve the external validity of any classifier trained on them. Moreover, the dataset generation process, grounded in sandbox telemetry from AnyRun, enables fine-grained feature extraction rooted in dynamic behavioral signatures rather than static file properties. The diagnostic precision of this approach is evident in the feature overlap matrix: the high dimensional sparsity and minimal redundancy across classes demonstrate a successful execution of the methodology's design goals, which are modularity, behavior-specificity, and diagnostic parsimony. The feature orthogonality further allows each dataset to support targeted classification models rather than forcing a monolithic malware-versus-benign dichotomy.

This modular architecture can be exploited for transfer learning across malware families or ensemble approaches that assign specialized classifiers to each malware class based on its unique feature space. Finally, by leveraging both cloud-scale sandbox telemetry and domain-specific feature curation, the study establishes a high-fidelity foundation for training behaviorally grounded detection algorithms. These datasets transcend signature-based limitations and are optimized for learning the nuanced operational footprints of diverse malware categories, offering a pathway toward more interpretable, resilient, and operationally aligned threat detection systems.

This study represents a significant advancement in the field of behavioral malware analysis. It is not only uniquely modular, comprising eight finely curated, malware-type-specific datasets, but also practically engineered for real-world applicability. By grounding feature selection in sandbox-executed behaviors and tailoring each dataset to the operational logic of its respective malware class, this resource bridges the gap between academic research and field-ready threat detection. Unlike monolithic datasets that rely on coarse-grained labels or static disassembly features, the present collection supports nuanced behavioral classification and can be readily deployed in modern machine learning pipelines. Its architecture is particularly well-suited to cybersecurity professionals working in IoT environments, where device-level visibility is limited and signature-based detection often fails. The lightweight, behaviorally disaggregated format of these datasets makes them ideally suited for rapid integration into endpoint detection systems, threat hunting platforms, and SOC-level triage tools.

Furthermore, the feature space has been delicately tailored to emphasize parsimony over brute dimensionality, ensuring both interpretability and computational efficiency—two qualities that are rarely achieved simultaneously. Whether used for academic modeling, operational detection, or adversarial resilience testing, this dataset suite offers a robust and highly actionable foundation. Its clarity, diagnostic precision, and modular design make it a powerful and unprecedented resource for anyone seeking to advance malware detection at scale.

5. Limitations

Although the dataset suite developed in this study offers a behaviorally grounded and modular foundation for malware detection, it is not without limitations. The construction pipeline used here is methodologically rigorous, but it is also time-consuming. Every sample requires manual execution, telemetry extraction, and feature curation. In practice, this slows expansion and limits scalability, particularly when larger collections of malware are needed. In operational environments, where threat intelligence has to be produced quickly, delays of this kind can become an obstacle. Automating feature extraction and

labeling from sandbox sources such as Anyrun would likely require a combination of lightweight telemetry parsers, real-time execution filters, and classifiers capable of handling raw data directly.

Another issue is the ambiguity that comes with malware classification itself. Categories like Trojan, worm, ransomware, or spyware are not always cleanly separated. Modern threats often blend features from more than one family. A ransomware strain that includes a backdoor module, for instance, also behaves like a Trojan, and in some cases even like a worm. This overlap complicates the definition of boundaries and makes feature selection less straightforward. In this study, features were chosen based on domain knowledge, but there is still some degree of subjectivity in that process. Relevant variables may be excluded unintentionally, or their importance misjudged. Subtle patterns, such as timing irregularities or code injection sequences, are especially difficult to capture without risking overfitting or bloating the feature set unnecessarily.

A further limitation comes from the evolving nature of malware itself. As attackers continue to develop new obfuscation and evasion techniques, some of the features selected here may lose diagnostic value. This means that periodic review and adjustment of the feature set will be necessary to keep the datasets relevant. Despite these challenges, the methodology remains reliable. The primary constraint is one of scale, not accuracy. The datasets produced are carefully curated, behaviorally meaningful, and suitable for both classical machine learning and modern artificial intelligence models. Future work should therefore concentrate on pipeline efficiency, the automation of behavioral labeling, and the more precise definition of class boundaries so that the complexity of real-world malware can be better represented.

6. Conclusions and Future Work

Malware threats continue to grow in both frequency and sophistication, and the consequences are especially serious for IoT networks. Traditional defenses and existing datasets have not kept pace with this rapid change. Most widely used resources are still based on static signatures or broad, generic labels, which rarely provide the detail needed for class-specific and interpretable detection models. This study was undertaken as an effort to address current limitations in the field of malware dataset design. Whereas much of the existing work has emphasized the accumulation of large quantities of samples, often at the expense of interpretability and diagnostic precision, the present approach focused on the construction of smaller but more carefully curated datasets. The guiding principle was not quantity, but quality—tailored, behaviorally grounded datasets designed to capture the distinctive attributes of specific malware families.

The primary objective was to design and select features with direct diagnostic value, ensuring that each dataset included attributes capable of distinguishing a given malware type. This aim was supported by a comprehensive literature review of each family and by the integration of current knowledge regarding malware operations. An additional objective was to evaluate the relationships among selected features, retaining only those that contributed independent diagnostic information while eliminating redundant or non-informative variables. Through this refinement process, the resulting datasets achieved both parsimony and precision.

To enhance usability, all datasets were labeled, facilitating their integration into future research and detection efforts. Importantly, each malware type was treated as an independent dataset rather than combined into a monolithic collection. This represents a deliberate departure from the prevailing trend in public datasets, which often merge all samples into a single binary classification of “malicious” versus “benign” or else present users with large numbers of irrelevant attributes. Such approaches obscure the behavioral diversity of

malware and limit the development of family-specific detection strategies. By contrast, the modular structure presented here allows researchers to study malware types in isolation, acknowledging that each family interacts with systems differently and must therefore be detected through distinct operational signatures.

Another key achievement of this work was the inclusion of suspicious files alongside malicious and benign samples. As suspicious files often exhibit hybrid behaviors, containing elements of both benign and malicious activity eliminates the ambiguity frequently encountered in real-world cybersecurity operations. By exposing models to these edge cases, the datasets encourage the development of classifiers that are more sensitive to subtle variations and better equipped to handle threats that combine attributes from multiple malware families.

The findings confirm that behaviorally informed and rigorously curated datasets yield greater diagnostic value, clearer interpretability, and stronger applicability to real-world detection tasks. By addressing both the structural shortcomings of existing resources and the practical challenges of malware diversity, this work lays a foundation for more robust detection models and provides a path forward for future research in IoT malware defense.

Looking forward, the dataset suite provides the foundation for building eight machine learning models, one for each malware type, tuned to maximize detection accuracy in their respective categories. Beyond static evaluation, the intention is to extend this work into a real-time detection framework for IoT devices, capable of classifying threats as they appear. The long-term vision is to scale these datasets into a broader intrusion detection system, one that can dynamically respond to behavioral anomalies across different devices and evolving threat types.

Author Contributions: M.M. conceived the study, developed the methodology, analyzed the results, and wrote and revised the manuscript. S.K. performed sample selection, sandbox execution, data collection, and dataset construction. M.S. contributed to the literature review and provided overall guidance. F.F.C. provided institutional resources. All authors have read and agreed to the published version of the manuscript.

Funding: The Lutcher Brown Distinguished Chair Professorship Fund of the University of Texas at San Antonio.

Data Availability Statement: The datasets are available from the first author, Mazdak Maghanaki, upon reasonable request.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Aslan, O.; Samet, R. A Comprehensive Review on Malware Detection Approaches. *IEEE Access* **2020**, *8*, 6249–6271. [CrossRef]
2. Mimura, M.; Ito, R. Applying NLP techniques to malware detection in a practical environment. *Int. J. Inf. Secur.* **2022**, *21*, 279–291. [CrossRef]
3. Borah, P.; Bhattacharyya, D.; Kalita, J. Malware Dataset Generation and Evaluation. In Proceedings of the 2020 IEEE 4th Conference on Information & Communication Technology (CICT), Chennai, India, 3–5 December 2020; pp. 1–6. [CrossRef]
4. Ronen, R.; Radu, M.; Feuerstein, C.; Yom-Tov, E.; Ahmadi, M. Microsoft Malware Classification Challenge. *arXiv* **2018**, arXiv:1802.10135. [CrossRef]
5. Anderson, H.S.; Roth, P. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *arXiv* **2018**, arXiv:1804.04637. [CrossRef]
6. Narudin, F.A.; Feizollah, A.; Anuar, N.B.; Gani, A. Evaluation of machine learning classifiers for mobile malware detection. *Soft Comput.* **2016**, *20*, 343–357. [CrossRef]
7. Zhang, H.; Luo, S.; Zhang, Y.; Pan, L. An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis. *IEEE Access* **2019**, *7*, 69246–69256. [CrossRef]

8. Magaña, E.; Sesma, I.; Morató, D.; Izal, M. Remote access protocols for Desktop-as-a-Service solutions. *PLoS ONE* **2019**, *14*, e0207512. [CrossRef]
9. Kramer, S.; Bradfield, J.C. A general definition of malware. *J. Comput. Virol.* **2010**, *6*, 105–114. [CrossRef]
10. Alenezi, M.N.; Alabdulrazzaq, H.; Alshaher, A.A.; Alkharang, M.M. Evolution of malware threats and techniques: A review. *Int. J. Commun. Netw. Inf. Secur.* **2020**, *12*, 326–337. [CrossRef]
11. Gandotra, E.; Bansal, D.; Sofat, S. Malware Analysis and Classification: A Survey. *J. Inf. Secur.* **2014**, *5*, 56–64. [CrossRef]
12. Eder-Neuhauser, P.; Zseby, T.; Fabini, J. Malware propagation in smart grid networks: Metrics, simulation and comparison of three malware types. *J. Comput. Virol. Hacking Tech.* **2019**, *15*, 109–125. [CrossRef]
13. Pachhala, N.; Jothilakshmi, S.; Battula, B.P. A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques. In Proceedings of the 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 7–9 October 2021; pp. 1207–1214. [CrossRef]
14. Pirscoveanu, R.S.; Hansen, S.S.; Larsen, T.M.T.; Stevanovic, M.; Pedersen, J.M.; Czech, A. Analysis of Malware behavior: Type classification using machine learning. In Proceedings of the 2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), London, UK, 8–9 June 2015; pp. 1–7. [CrossRef]
15. Chandy, J. Review on malware, types, and its analysis. *IJRASET* **2022**, *10*, 386–390. [CrossRef]
16. Dube, T.; Raines, R.; Peterson, G.; Bauer, K.; Grimaila, M.; Rogers, S. Malware Type Recognition and Cyber Situational Awareness. In Proceedings of the 2010 IEEE Second International Conference on Social Computing, Minneapolis, MN, USA, 20–22 August 2010; pp. 938–943. [CrossRef]
17. García, S.; Grill, M.; Stiborek, J.; Zunino, A. An empirical comparison of botnet detection methods. *Comput. Secur.* **2014**, *45*, 100–123. [CrossRef]
18. Strayer, W.T.; Lapsely, D.; Walsh, R.; Livadas, C. Botnet detection based on network behavior. In *Botnet Detection: Countering the Largest Security Threat*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 1–24.
19. Karim, A.; Salleh, R.B.; Shiraz, M.; Shah, S.A.A.; Awan, I.; Anuar, N.B. Botnet detection techniques: Review, future trends, and issues. *J. Zhejiang Univ. Sci. C* **2014**, *15*, 943–983. [CrossRef]
20. Kok, S.; Abdullah, A.; Jhanjhi, N.; Supramaniam, M. Ransomware, threat and detection techniques: A review. *Int. J. Comput. Sci. Netw. Secur.* **2019**, *19*, 136.
21. Sgandurra, D.; Muñoz-González, L.; Mohsen, R.; Lupu, E.C. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. *arXiv* **2016**, arXiv:1609.03020. [CrossRef]
22. ANY.RUN Interactive Malware Analysis Sandbox. Available online: https://any.run (accessed on 20 March 2025).
23. Lashkari, A.H.; Kadir, A.F.A.; Taheri, L.; Ghorbani, A.A. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In Proceedings of the 2018 International Carnahan conference on security technology (ICCST), Montreal, QC, Canada, 22–25 October 2018; pp. 1–7.
24. Li, P.; Liu, L.; Gao, D.; Reiter, M.K. On challenges in evaluating malware clustering. In *International Workshop on Recent Advances in Intrusion Detection*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 238–255.
25. Li, Y.; Xiong, K.; Chin, T.; Hu, C. A machine learning framework for domain generation algorithm-based malware detection. *IEEE Access* **2019**, *7*, 32765–32782. [CrossRef]
26. Mahdavifar, S.; Alhadidi, D.; Ghorbani, A.A. Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *J. Netw. Syst. Manag.* **2022**, *30*, 22. [CrossRef]
27. Maghanaki, M.; Chen, F.F.; Shahin, M.; Hosseinzadeh, A.; Bouzary, H. A Novel Transformer-Based Model for Comprehensive Text-Aware Service Composition in Cloud-Based Manufacturing. In *Intelligent Production and Industry 5.0 with Human Touch, Resilience, and Circular Economy*; Šormaz, D.N., Bidanda, B., Alhawari, O., Geng, Z., Eds.; Lecture Notes in Production Engineering; Springer Nature: Cham, Switzerland, 2025; pp. 313–321. [CrossRef]
28. Gupta, A.; Singh, A. A comprehensive survey on cyber-physical systems towards healthcare 4.0. *SN Comput. Sci.* **2023**, *4*, 199. [CrossRef]
29. Maghanaki, M.; Shahin, M.; Chen, F.F.; Hosseinzadeh, A. Improving Early Diagnosis: The Intersection of Lean Healthcare and Computer Vision in Cancer Detection. In *Proceedings of the Second International Conference on Advances in Computing Research (ACR'24), Madrid, Spain, 3–5 June 2024*; Daimi, K., Al Sadoon, A., Eds.; Lecture Notes in Networks and Systems; Springer Nature: Cham, Switzerland, 2024; Volume 956, pp. 404–413. [CrossRef]
30. Sebastián, M.; Rivera, R.; Kotzias, P.; Caballero, J. Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 230–253.
31. Ahmed, Y.A.; Koçer, B.; Huda, S.; Al-Rimy, B.A.S.; Hassan, M.M. A system call refinement-based enhanced Minimum Redundancy Maximum Relevance method for ransomware early detection. *J. Netw. Comput. Appl.* **2020**, *167*, 102753. [CrossRef]
32. Kattamuri, S.J.; Penmatsa, R.K.V.; Chakravarty, S.; Madabathula, V.S.P. Swarm Optimization and Machine Learning Applied to PE Malware Detection towards Cyber Threat Intelligence. *Electronics* **2023**, *12*, 342. [CrossRef]
33. Oliveira, A. Malware Analysis Datasets: Top-1000 PE Imports. *IEEE DataPort* **2019**, *10*, 21227. [CrossRef]

34. Alshomrani, M.; Albeshri, A.; Alsulami, A.A.; Alturki, B. An Explainable Hybrid CNN–Transformer Architecture for Visual Malware Classification. *Sensors* **2025**, *25*, 4581. [CrossRef]
35. Obidiagha, C.C.; Rahouti, M.; Hayajneh, T. DeepImageDroid: A Hybrid Framework Leveraging Visual Transformers and Convolutional Neural Networks for Robust Android Malware Detection. *IEEE Access* **2024**, *12*, 156285–156306. [CrossRef]
36. Chen, Q.; Bridges, R.A. Automated behavioral analysis of malware: A case study of wannacry ransomware. In Proceedings of the 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 18–21 December 2017; pp. 454–460.
37. Hansen, S.S.; Larsen, T.M.T.; Stevanovic, M.; Pedersen, J.M. An approach for detection and family classification of malware based on behavioral analysis. In Proceedings of the 2016 International Conference on Computing, Networking and Communications (ICNC), Kauai, HI, USA, 15–18 February 2016; pp. 1–5.
38. Narayanan, A.; Chandramohan, M.; Chen, L.; Liu, Y. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Trans. Emerg. Top. Comput. Intell.* **2017**, *1*, 157–175. [CrossRef]
39. Calleja, A.; Tapiador, J.; Caballero, J. The malsource dataset: Quantifying complexity and code reuse in malware development. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 3175–3190. [CrossRef]
40. Moustafa, N. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustain. Cities Soc.* **2021**, *72*, 102994. [CrossRef]
41. Hosseinzadeh, A.; Shahin, M.; Chen, F.F.; Maghanaki, M.; Tseng, T.-L.; Rashidifar, R. Using Applied Machine Learning to Detect Cyber-Security Threats in Industrial IoT Devices. In *Flexible Automation and Intelligent Manufacturing: Manufacturing Innovation and Preparedness for the Changing World Order*; Wang, Y.-C., Chan, S.H., Wang, Z.-H., Eds.; Lecture Notes in Mechanical Engineering; Springer Nature: Cham, Switzerland, 2024; pp. 22–30. [CrossRef]
42. Raghuraman, C.; Suresh, S.; Shivshankar, S.; Chapaneri, R. Static and dynamic malware analysis using machine learning. In *First International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2019, Jaipur, India, 29–30 March 2019*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 793–806.
43. Saranya, G.; Kumaran, K.; Misra, A.; Mishra, M. Context-Aware Malware Scoring: A Multi-Model Ensemble Approach using Structured and Unstructured Data. In Proceedings of the 2025 6th International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), Tirunelveli, India, 17–19 June 2025; pp. 53–59. [CrossRef]
44. Perdisci, R.; Lanzi, A.; Lee, W. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC), Anaheim, CA, USA, 8–12 December 2008; pp. 301–310.
45. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.D.B.; Wang, Y.; Iqbal, F. Malware Classification with Deep Convolutional Neural Networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018; pp. 1–5. [CrossRef]
46. Ghaleb, T.; Abduljalil, O.; Hassan, S. CI/CD Configuration Practices in Open-Source Android Apps: An Empirical Study. *ACM Trans. Softw. Eng. Methodol.* **2025**, 3736758. [CrossRef]
47. Shahin, M.; Chen, F.F.; Maghanaki, M.; Firouzranjbar, S.; Hosseinzadeh, A. Evaluating the fidelity of statistical forecasting and predictive intelligence by utilizing a stochastic dataset. *Int. J. Adv. Manuf. Technol.* **2024**, *138*, 193–223. [CrossRef]
48. Joyce, R.J.; Amlani, D.; Nicholas, C.; Raff, E. MOTIF: A Malware Reference Dataset with Ground Truth Family Labels. *Comput. Secur.* **2023**, *124*, 102921. [CrossRef]
49. Berrios, S.; Leiva, D.; Olivares, B.; Allende-Cid, H.; Hermosilla, P. Systematic Review: Malware Detection and Classification in Cybersecurity. *Appl. Sci.* **2025**, *15*, 7747. [CrossRef]
50. Paranthaman, R.; Thuraisingham, B. Malware Collection and Analysis. In Proceedings of the 2017 IEEE International Conference on Information Reuse and Integration (IRI), San Diego, CA, USA, 4–6 August 2017; pp. 26–31. [CrossRef]
51. Serpanos, D.; Michalopoulos, P.; Xenos, G.; Ieronymakis, V. Sisyfos: A Modular and Extendable Open Malware Analysis Platform. *Appl. Sci.* **2021**, *11*, 2980. [CrossRef]
52. Martín, A.; Lara-Cabrera, R.; Camacho, D. Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset. *Inf. Fusion* **2019**, *52*, 128–142. [CrossRef]
53. Guerra-Manzanares, A.; Bahsi, H.; Nõmm, S. KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization. *Comput. Secur.* **2021**, *110*, 102399. [CrossRef]
54. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9–11 March 2016; pp. 183–194. [CrossRef]
55. Pyran1. Malware Collection Repository. Available online: https://github.com/Pyran1/MalwareCollection (accessed on 12 July 2024).
56. bediger4000. Network-Worm-Simulator. [Online]. Available online: https://github.com/bediger4000/network-worm-simulator (accessed on 12 July 2024).

57. Rshipp. Awesome-Malware-Analysis. [Online]. Available online: https://github.com/rshipp/awesome-malware-analysis (accessed on 12 July 2024).
58. Ytisf. theZoo. [Online]. Available online: https://github.com/ytisf/theZoo (accessed on 12 July 2024).
59. Cryptwareapps. Malware-Database. [Online]. Available online: https://github.com/cryptwareapps/Malware-Database (accessed on 12 July 2024).
60. Abdulkadir-Gungor. JPGtoMalware. [Online]. Available online: https://github.com/abdulkadir-gungor/JPGtoMalware (accessed on 12 July 2024).
61. Ghost-crypto-exe. 2018-Malware-Repository. [Online]. Available online: https://github.com/Ghost-crypto-exe/2018-Malware-Repository (accessed on 12 July 2024).
62. MalDev101. Loveware. [Online]. Available online: https://github.com/MalDev101/Loveware (accessed on 12 July 2024).
63. Denham, B.; Thompson, D.R. Ransomware and Malware Sandboxing. In Proceedings of the 2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 26–29 October 2022; pp. 173–179. [CrossRef]
64. Vasilescu, M.; Gheorghe, L.; Tapus, N. Practical malware analysis based on sandboxing. In Proceedings of the 2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference, Chisinau, Moldova, 11–13 September 2014; pp. 1–6. [CrossRef]
65. Dewald, A.; Holz, T.; Freiling, F.C. ADSandbox: Sandboxing JavaScript to fight malicious websites. In Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; pp. 1859–1864. [CrossRef]
66. Griffin, K.; Schneider, S.; Hu, X.; Chiueh, T. Automatic Generation of String Signatures for Malware Detection. In *Recent Advances in Intrusion Detection*; Kirda, E., Jha, S., Balzarotti, D., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5758, pp. 101–120. [CrossRef]
67. Spreitzenbarth, M.; Schreck, T.; Echtler, F.; Arp, D.; Hoffmann, J. Mobile-Sandbox: Combining static and dynamic analysis with machine-learning techniques. *Int. J. Inf. Secur.* **2015**, *14*, 141–153. [CrossRef]
68. Le, H.-V.; Ngo, Q.-D. V-Sandbox for Dynamic Analysis IoT Botnet. *IEEE Access* **2020**, *8*, 145768–145786. [CrossRef]
69. Alhaidari, F.; Shaib, N.A.; Alsafi, M.; Alharbi, H.; Alawami, M.; Aljindan, R.; Zagrouba, R. ZeVigilante: Detecting Zero-Day Malware Using Machine Learning and Sandboxing Analysis Techniques. *Comput. Intell. Neurosci.* **2022**, *2022*, 1615528. [CrossRef]
70. Keim, Y.; Mohapatra, A.K. Cyber threat intelligence framework using advanced malware forensics. *Int. J. Inf. Technol.* **2022**, *14*, 521–530. [CrossRef]
71. Drew, J.; Moore, T.; Hahsler, M. Polymorphic Malware Detection Using Sequence Classification Methods. In Proceedings of the 2016 IEEE Security and Privacy Workshops (SPW), San Jose, CA, USA, 22–26 May 2016; pp. 81–87. [CrossRef]
72. Oh, C.; Ha, J.; Roh, H. A Survey on TLS-Encrypted Malware Network Traffic Analysis Applicable to Security Operations Centers. *Appl. Sci.* **2021**, *12*, 155. [CrossRef]
73. Bowles, S.; Hernandez-Castro, J. The first 10 years of the Trojan Horse defence. *Comput. Fraud Secur.* **2015**, *2015*, 5–13. [CrossRef]
74. Hughes, L.A.; DeLone, G.J. Viruses, Worms, and Trojan Horses: Serious Crimes, Nuisance, or Both? *Soc. Sci. Comput. Rev.* **2007**, *25*, 78–98. [CrossRef]
75. Kiltz, S.; Lang, A.; Dittmann, J. Malware: Specialized Trojan Horse. In *Cyber Warfare and Cyber Terrorism*; Janczewski, L., Colarik, A., Eds.; IGI Global: Hershey, PA, USA, 2007; pp. 154–160. [CrossRef]
76. Wang, R.; Zhang, G.; Liu, S.; Chen, P.-Y.; Xiong, J.; Wang, M. Practical Detection of Trojan Neural Networks: Data-Limited and Data-Free Cases. In *Computer Vision–ECCV 2020*; Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2020; Volume 12368, pp. 222–238. [CrossRef]
77. Gao, Y.; Kim, Y.; Doan, B.G.; Zhang, Z.; Zhang, G.; Nepal, S.; Kim, H. Design and evaluation of a multi-domain trojan detection method on deep neural networks. *IEEE Trans. Dependable Secure Comput.* **2021**, *19*, 2349–2364. [CrossRef]
78. Fields, G.; Samragh, M.; Javaheripi, M.; Koushanfar, F.; Javidi, T. Trojan signatures in DNN weights. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada, 11–17 October 2021; pp. 12–20.
79. Chen, W.; Song, D.; Li, B. Trojdiff: Trojan attacks on diffusion models with diverse targets. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 17–24 June 2023; pp. 4035–4044.
80. Ullah, S.; Ahmad, T.; Buriro, A.; Zara, N.; Saha, S. TrojanDetector: A Multi-Layer Hybrid Approach for Trojan Detection in Android Applications. *Appl. Sci.* **2022**, *12*, 10755. [CrossRef]
81. Hussain, A.; Rabin, M.R.I.; Alipour, M.A. On Trojan Signatures in Large Language Models of Code. *arXiv* **2024**, arXiv:2402.16896. [CrossRef]
82. Antonakakis, M.; April, T.; Bailey, M.; Bernhard, M.; Bursztein, E.; Cochran, J.; Zhou, Y. Understanding the mirai botnet. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 1093–1110.
83. Margolis, J.; Oh, T.T.; Jadhav, S.; Kim, Y.H.; Kim, J.N. An In-Depth Analysis of the Mirai Botnet. In Proceedings of the 2017 International Conference on Software Security and Assurance (ICSSA), Altoona, PA, USA, 24–25 July 2017; pp. 6–12. [CrossRef]

84. Kambourakis, G.; Kolias, C.; Stavrou, A. The Mirai botnet and the IoT Zombie Armies. In Proceedings of the MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM), Baltimore, MD, USA, 23–25 October 2017; pp. 267–272. [CrossRef]
85. Zhang, X.; Upton, O.; Beebe, N.L.; Choo, K.-K.R. IoT Botnet Forensics: A Comprehensive Digital Forensic Case Study on Mirai Botnet Servers. *Forensic Sci. Int. Digit. Investig.* **2020**, *32*, 300926. [CrossRef]
86. Ahmed, Z.; Danish, S.M.; Qureshi, H.K.; Lestas, M. Protecting IoTs from Mirai Botnet Attacks Using Blockchains. In Proceedings of the 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Limassol, Cyprus, 11–13 September 2019; pp. 1–6. [CrossRef]
87. Jerkins, J.A. Motivating a market or regulatory solution to IoT insecurity with the Mirai botnet code. In Proceedings of the 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 9–11 January 2017; pp. 1–5. [CrossRef]
88. Eustis, A.G. The Mirai Botnet and the Importance of IoT Device Security. In *Proceedings of the 16th International Conference on Information Technology-New Generations (ITNG 2019), Las Vegas, NV, USA, 1–3 April 2019*; Latifi, S., Ed.; Advances in Intelligent Systems and Computing; Springer International Publishing: Cham, Switzerland, 2019; Volume 800, pp. 85–89. [CrossRef]
89. Bae, S.I.; Lee, G.B.; Im, E.G. Ransomware detection using machine learning algorithms. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5422. [CrossRef]
90. Urooj, U.; Al-rimy, B.A.S.; Zainal, A.; Ghaleb, F.A.; Rassam, M.A. Ransomware Detection Using the Dynamic Analysis and Machine Learning: A Survey and Research Directions. *Appl. Sci.* **2021**, *12*, 172. [CrossRef]
91. Alraizza, A.; Algarni, A. Ransomware Detection Using Machine Learning: A Survey. *Big Data Cogn. Comput.* **2023**, *7*, 143. [CrossRef]
92. Khammas, B.M. Ransomware Detection using Random Forest Technique. *ICT Express* **2020**, *6*, 325–331. [CrossRef]
93. Kapoor, A.; Gupta, A.; Gupta, R.; Tanwar, S.; Sharma, G.; Davidson, I.E. Ransomware Detection, Avoidance, and Mitigation Scheme: A Review and Future Directions. *Sustainability* **2021**, *14*, 8. [CrossRef]
94. Riley, R.; Jiang, X.; Xu, D. Multi-aspect profiling of kernel rootkit behavior. In Proceedings of the 4th ACM European Conference on Computer Systems, Nuremberg, Germany, 1–3 April 2009; pp. 47–60. [CrossRef]
95. Joy, J.; John, A.; Joy, J. Rootkit Detection Mechanism: A Survey. In *Advances in Parallel Distributed Computing*; Nagamalai, D., Renault, E., Dhanuskodi, M., Eds.; Communications in Computer and Information Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 203, pp. 366–374. [CrossRef]
96. Liu, L.; Yin, Z.; Shen, Y.; Lin, H.; Wang, H. Research and Design of Rootkit Detection Method. *Phys. Procedia* **2012**, *33*, 852–857. [CrossRef]
97. Shahin, M.; Maghanaki, M.; Hosseinzadeh, A.; Chen, F.F. Advancing Network Security in Industrial IoT: A Deep Dive into AI-Enabled Intrusion Detection Systems. *Adv. Eng. Inform.* **2024**, *62*, 102685. [CrossRef]
98. Arnold, T.M. *A Comparative Analysis of Rootkit Detection Techniques*; University of Houston-Clear Lake: Houston, TX, USA, 2011.
99. Stühn, J.; Hilgert, J.-N.; Lambertz, M. The Hidden Threat: Analysis of Linux Rootkit Techniques and Limitations of Current Detection Tools. *Digit. Threats Res. Pract.* **2024**, *5*, 28. [CrossRef]
100. Kumar, S.S.; Stephen, S.; Rumysia, M.S. Rootkit Detection using Deep Learning: A Comprehensive Survey. In Proceedings of the 2024 10th International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, India, 12–14 April 2024; pp. 365–370. [CrossRef]
101. Yao, Y.; Sheng, C.; Fu, Q.; Liu, H.; Wang, D. A propagation model with defensive measures for PLC-PC worms in industrial networks. *Appl. Math. Model.* **2019**, *69*, 696–713. [CrossRef]
102. Weaver, N.; Paxson, V.; Staniford, S.; Cunningham, R. A taxonomy of computer worms. In Proceedings of the 2003 ACM Workshop on Rapid Malcode, Washington, DC, USA, 27 October 2003; pp. 11–18. [CrossRef]
103. Moskovitch, R.; Elovici, Y.; Rokach, L. Detection of unknown computer worms based on behavioral classification of the host. *Comput. Stat. Data Anal.* **2008**, *52*, 4544–4566. [CrossRef]
104. Stopel, D.; Moskovitch, R.; Boger, Z.; Shahar, Y.; Elovici, Y. Using artificial neural networks to detect unknown computer worms. *Neural Comput. Appl.* **2009**, *18*, 663–674. [CrossRef]
105. Cole, R.J. Computer Worms, Detection, and Defense. In *Encyclopedia of Information Ethics and Security*; Quigley, M., Ed.; IGI Global: Hershey, PA, USA, 2007; pp. 89–95. [CrossRef]
106. Moskovitch, R.; Nissim, N.; Stopel, D.; Feher, C.; Englert, R.; Elovici, Y. Improving the Detection of Unknown Computer Worms Activity Using Active Learning. In *KI 2007: Advances in Artificial Intelligence*; Hertzberg, J., Beetz, M., Englert, R., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4667, pp. 489–493. [CrossRef]
107. Moskovitch, R.; Gus, I.; Pluderman, S.; Stopel, D.; Glezer, C.; Shahar, Y.; Elovici, Y. Detection of Unknown Computer Worms Activity Based on Computer Behavior using Data Mining. In Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications, Honolulu, HI, USA, 1–5 April 2007; pp. 169–177. [CrossRef]
108. Qabalin, M.K.; Naser, M.; Alkasassbeh, M. Android Spyware Detection Using Machine Learning: A Novel Dataset. *Sensors* **2022**, *22*, 5765. [CrossRef]

109. Wang, T.-Y.; Horng, S.-J.; Su, M.-Y.; Wu, C.-H.; Wang, P.-C.; Su, W.-Z. A Surveillance Spyware Detection System Based on Data Mining Methods. In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 3236–3241. [CrossRef]
110. Mallikarajunan, K.M.E.N.; Preethi, S.R.; Selvalakshmi, S.; Nithish, N. Detection of Spyware in Software Using Virtual Environment. In Proceedings of the 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 23–25 April 2019; pp. 1138–1142. [CrossRef]
111. Shahzad, R.K.; Haider, S.I.; Lavesson, N. Detection of Spyware by Mining Executable Files. In Proceedings of the 2010 International Conference on Availability, Reliability and Security, Krakow, Poland, 15–18 February 2010; pp. 295–302. [CrossRef]
112. Shahin, M.; Maghanaki, M.; Chen, F.F.; Hosseinzadeh, A.; Rashidifar, R. Text Mining via ChatGPT to Extract Voice of Customer Insights from Twitter Conversational Interactions Dataset. In *Flexible Automation and Intelligent Manufacturing: Manufacturing Innovation and Preparedness for the Changing World Order*; Wang, Y.-C., Chan, S.H., Wang, Z.-H., Eds.; Lecture Notes in Mechanical Engineering; Springer Nature: Cham, Switzerland, 2024; pp. 261–268. [CrossRef]
113. Shahin, M.; Chen, F.F.; Maghanaki, M.; Hosseinzadeh, A. Adapting the GPT engine for proactive customer insight extraction in product development. *Manuf. Lett.* **2024**, *41*, 1376–1385. [CrossRef]
114. Abualhaj, M.M.; Al-Shamayleh, A.S.; Munther, A.; Alkhatib, S.N.; Hiari, M.O.; Anbar, M. Enhancing spyware detection by utilizing decision trees with hyperparameter optimization. *Bull. Electr. Eng. Inform.* **2024**, *13*, 3653–3662. [CrossRef]
115. Wajahat, A.; Imran, A.; Latif, J.; Nazir, A.; Bilal, A. A Novel Approach of Unprivileged Keylogger Detection. In Proceedings of the 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 30–31 January 2019; pp. 1–6. [CrossRef]
116. Singh, A.; Choudhary, P.; Singh, A.K.; Tyagi, D.K. Keylogger Detection and Prevention. *J. Phys. Conf. Ser.* **2021**, *2007*, 012005. [CrossRef]
117. Simms, S.; Maxwell, M.; Johnson, S.; Rrushi, J. Keylogger Detection Using a Decoy Keyboard. In *Data and Applications Security and Privacy XXXI*; Livraga, G., Zhu, S., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2017; Volume 10359, pp. 433–452. [CrossRef]
118. Gunalakshmii, S.; Ezhumalai, P. Mobile keylogger detection using machine learning technique. In Proceedings of the IEEE International Conference on Computer Communication and Systems ICCCS14, Chennai, India, 20–21 February 2014; pp. 51–56. [CrossRef]
119. Solairaj, A.; Prabanand, S.C.; Mathalairaj, J.; Prathap, C.; Vignesh, L.S. Keyloggers software detection techniques. In Proceedings of the 2016 10th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 7–8 January 2016; pp. 1–6. [CrossRef]
120. Elelegwu, D.; Chen, L.; Ji, Y.; Kim, J. A Novel Approach to Detecting and Mitigating Keyloggers. In Proceedings of the SoutheastCon 2024, Atlanta, GA, USA, 21–24 March 2024; pp. 1583–1590. [CrossRef]
121. Balakrishnan, Y.; Renjith, P.N. An analysis on Keylogger Attack and Detection based on Machine Learning. In Proceedings of the 2023 International Conference on Artificial Intelligence and Knowledge Discovery in Concurrent Engineering (ICECONF), Chennai, India, 5–7 January 2023; pp. 1–8. [CrossRef]
122. Cohen, F. Computer viruses. *Comput. Secur.* **1987**, *6*, 22–35. [CrossRef]
123. Balthrop, J.; Forrest, S.; Newman, M.E.J.; Williamson, M.M. Technological Networks and the Spread of Computer Viruses. *Science* **2004**, *304*, 527–529. [CrossRef]
124. Adleman, L.M. An Abstract Theory of Computer Viruses. In *Advances in Cryptology—CRYPTO' 88*; Goldwasser, S., Ed.; Lecture Notes in Computer Science; Springer: New York, NY, USA, 1990; Volume 403, pp. 354–374. [CrossRef]
125. Spafford, E.H. Computer Viruses as Artificial Life. *Artif. Life* **1994**, *1*, 249–265. [CrossRef]
126. Yu, B.; Fang, Y.; Yang, Q.; Tang, Y.; Liu, L. A survey of malware behavior description and analysis. *Front. Inf. Technol. Electron. Eng.* **2018**, *19*, 583–603. [CrossRef]
127. Kim, D.-W.; Shin, G.-Y.; Han, M.-M. Analysis of feature importance and interpretation for malware classification. *Comput. Mater. Contin.* **2020**, *65*, 1891–1904. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.