

Article

Linux IoT Malware Variant Classification Using Binary Lifting and Opcode Entropy

Jayanthi Ramamoorthy ^{*,†}, Khushi Gupta [†], Narasimha K. Shashidhar and Cihan Varol 

Department of Computer Science, Sam Houston State University, Huntsville, TX 77340, USA;
kxg095@shsu.edu (K.G.); nks001@shsu.edu (N.K.S.); cxv007@shsu.edu (C.V.)

* Correspondence: jxr153@shsu.edu

[†] These authors contributed equally to this work.

Abstract: Binary function analysis is fundamental in understanding the behavior and genealogy of malware. The detection, classification, and analysis of Linux IoT malware and its variants present significant challenges due to the wide range of architectures supported by the Linux IoT platform. This study concentrates on static analysis using binary lifting techniques to extract and analyze Intermediate Representation (IR) opcode sequences. We introduce a set of statistical entropy-based features derived from these IR opcode sequences, establishing a practical and straightforward methodology for machine learning classification models. By exclusively analyzing function metadata and opcode entropy, our architecture-agnostic approach not only efficiently detects malware but also classifies its variants with a high degree of accuracy, achieving an F1 score of 97%. The proposed approach offers a robust alternative for enhancing malware detection and variant identification frameworks for IoT devices.

Keywords: ELF static analysis; binary lifting; opcode sequence analysis; machine learning; malware detection; malware classification



Citation: Ramamoorthy, J.; Gupta, K.; Shashidhar, N.K.; Varol, C. Linux IoT Malware Variant Classification Using Binary Lifting and Opcode Entropy. *Electronics* **2024**, *13*, 2381. <https://doi.org/10.3390/electronics13122381>

Academic Editors: Abdussalam Elhanashi and Pierpaolo Dini

Received: 29 May 2024
Revised: 12 June 2024
Accepted: 15 June 2024
Published: 18 June 2024



Copyright: © 2024 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

The rapid expansion of the Fourth Industrial Revolution and the Internet of Things (IoT) is transforming cyber–physical systems on an unprecedented scale. By 2030, it is projected that 75% of all devices will be IoT devices [1]. This surge will significantly influence various domains, including transportation, healthcare, and energy management. However, the complexity of hardware and software design, along with inadequate security features, makes IoT devices increasingly vulnerable to cyber-attacks [2].

According to the Zscaler ThreatLabz Enterprise IoT and OT Threat Report of 2023, there was a 400% increase in IoT malware attacks [3]. With the rapid adoption of IoT technologies in the industry, there will be an endless rise in these attacks. In September 2016, variants of the Linux Mirai malware were responsible for 1.1 Tbps DDoS attacks directed at the Dyn Domain Name System (DNS) provider [4]. In 2017, Linux/Brickerbot, a botnet similar to Mirai, infected more than 10 million IoT devices around the world [5].

Linux operating systems dominate the landscape of IoT platforms, as indicated by Antony et al. [6]. However, this widespread adoption has caused a corresponding increase in malware targeting Linux-based systems. AV-ATLAS's report [7] underscores this concern, revealing a 50% surge in new Linux malware within a single year. This alarming trend is attributed to Linux's prevalence in IoT devices, scalable cloud infrastructures, and the growing adoption of containerized applications. As Linux operating systems remain integral to numerous digital ecosystems, mitigating these security threats remains a critical challenge.

Malware analysis is the process of examining and understanding malicious binaries (malware) to determine their behavior, purpose, and potential impact. This process involves a range of techniques and tools to analyze different aspects of malware, such as its code,

behavior, network traffic, and interactions with the system. Malware analysis can be broadly classified into two categories: Static analysis and Dynamic analysis. Static analysis involves analyzing the code and structure of malware without execution, which can be performed using features extracted from API calls, strings, byte n-grams, opcodes, etc., whereas dynamic analysis involves executing the malware in a controlled environment to discern its behavior and potential impact.

Opcodes (operation codes) are the fundamental components of machine language instructions in a binary file. They specify the exact operations that the CPU must perform. In the context of a malware binary, opcodes represent the low-level instructions that the malware executes to achieve its malicious objectives. Opcodes have been previously used in static malware detection [8,9]. However, when using opcodes as the features, one of the main challenges in detecting and classifying IoT malware is the heterogeneous device architectures [10,11]. Opcodes can perform equivalent functions across different architectures but due to the diversity of instruction sets of the malware binaries it is not feasible to use a consistent analysis methodology to detect and classify malware binaries across different underlying architectures.

In this paper, we detect and classify Linux IoT ARM malware using the static analysis of disassembled opcode sequences extracted from the functions of the binaries. Due to the diverse range of architectures of the binaries, we use binary lifting technique to extract and analyze the Intermediate Representation (IR) of the opcode. Furthermore, we conduct statistical analysis on the IR opcode to derive a set of features such as the entropy. These features are subsequently used to train various machine learning models, such as Logistic Regression, Support Vector Classification (SVC), Random Forest, and Multi-Layer Perceptron (MLP) Neural Network.

The main contributions of this paper are as follows:

- **Architecture-agnostic methodology:** We propose an architecture-agnostic approach that relies on Intermediate Representation (IR) opcode instructions along with opcode entropy features to detect and categorize IoT ARM malware variants using Binary lifting. Binary lifting involves translating Instruction Set Architecture (ISA) assembly code which vary significantly between architectures, into a high-level Intermediate Representation. This process standardizes different opcode sets into a consistent IR format. Our method focuses specifically on opcode instructions and omits operands and register details to provide an abstract view of function behavior.
- **Statistical IR opcode entropy feature set:** We introduce a statistical feature set related to the entropy of Intermediate Representation (IR) opcodes. This feature set leverages the variability within opcode sequences to enhance the accuracy of malware detection and classification.
- **Comprehensive function analysis dataset:** We have developed a dataset that encompasses function metadata, the function IR opcode sequence, and statistical IR opcode entropy features for each ELF malware and benign binary. This dataset is derived from the raw IoT malware binary dataset and is structured to support machine learning classification models.

2. Preliminaries

In this section, we present an overview of ELF files, binary lifting techniques, and key characteristics of the binary files in our dataset, such as whether they are stripped of symbols and whether they are statically or dynamically linked. This discussion aims to provide a comprehensive understanding of the binaries in our dataset and elucidate the process of extracting the opcode Intermediate Representation (IR) from them.

2.1. ELF Files and Binary Lifting

An Executable and Linkable Format (ELF) file is a widely used standardized file format for executables, object code, shared libraries, and core dumps in Unix-like operating systems. Designed to be flexible and extensible, the ELF format supports various processor

architectures and is the default binary format for many architectures. An ELF file is structured into distinct parts, including the ELF header, program headers, and section headers as shown in Figure 1.

- **ELF header:** Contains metadata about the file, such as its type, architecture, and entry point for execution.
- **Program header:** Describe segments that the operating system loads into memory, such as executable code and data segments.
- **Section headers:** Delineate various sections of the file used during linking and relocation, including sections for code (‘.text’), initialized data (‘.data’), and uninitialized data (‘.bss’).

An ELF file, is a standard binary file commonly generated in various architectures, including x86, ARM, MIPS, and others are composed of a sequence of bytes. When the ELF file is disassembled, an assembly code is generated. It is made up of opcode (commands of a specific architecture) and the operand (parameters used in an operation). This assembly code delineates the low-level operations that the CPU executes, detailing each instruction’s specific role in the program’s functionality. Each opcode corresponds to a specific operation, such as arithmetic calculations (addition, subtraction), data movement (loading and storing data), control flow changes (jumps, calls, conditional branches), and system calls (interacting with the operating system). These operations are encoded in a format that the CPU can directly execute.

Malware is a series of malicious behaviors, and opcodes have been used in prior works to identify malware and its attack behaviours. However, the analysis of ELF binaries can be complex due to the diverse instruction set architectures (ISAs) that define unique opcode specifications for each architecture. This diversity necessitates distinct analyses for each architecture.

Binary lifting is a sophisticated technique used to translate low-level ISA assembly code into high-level intermediate representations (IR). This process is essential for abstracting and standardizing assembly code that originates from different architectures and opcode sets. By converting diverse assembly instructions into a uniform IR format, binary lifting enables a more streamlined and consistent analysis across various platforms [12].

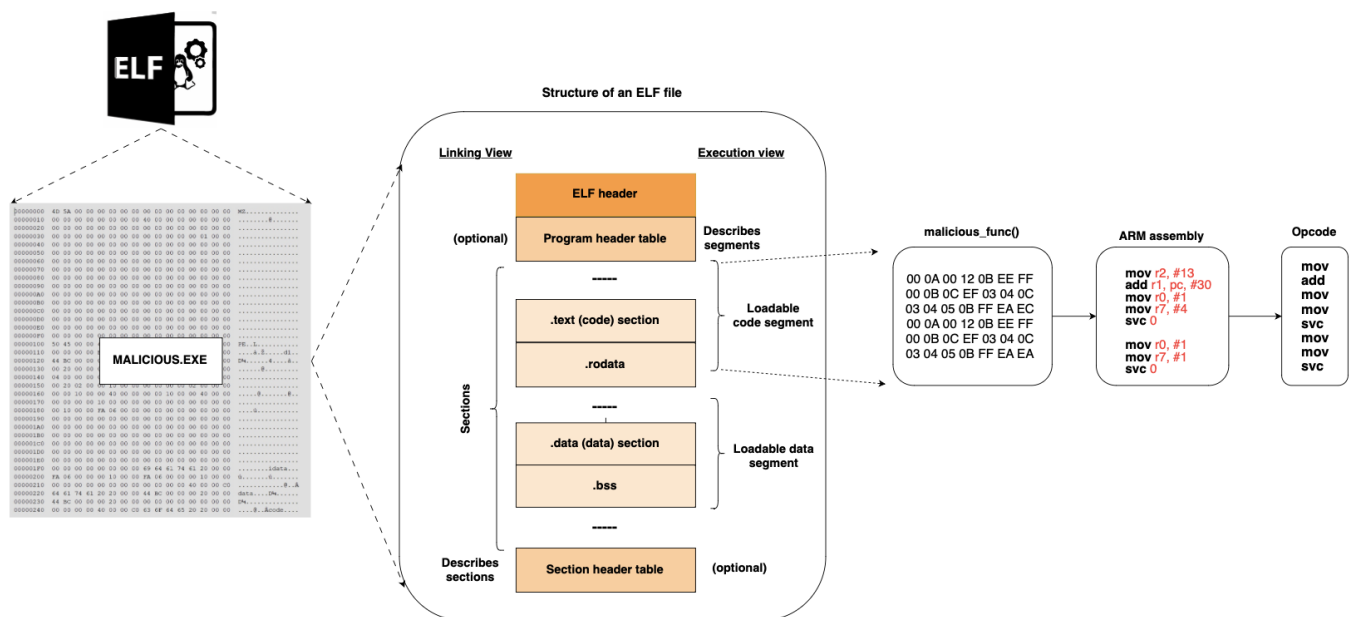


Figure 1. A dissection on the disassembly of an ELF file.

Binary lifting addresses RISC (Reduced Instruction Set Computing) architectures that have a smaller instruction set by preserving the semantics and abstracting the specifics to

Intermediate Representation (IR). For CISC (Complex Instruction Set Computing) architectures, binary lifting decomposes the instruction to simpler IR operations and ensures that all the functionality is accurately retained. In addition, differences in registers and memory access are normalized in IR, providing a consistent framework.

2.2. Characteristics of ELF Binaries

ELF binaries can vary based on whether they are stripped or non-stripped, as well as their linking characteristics, being either statically linked or dynamically linked.

2.2.1. Stripped and Non-Stripped Malware

ELF malware binaries can be stripped of all debugging information, symbol tables, and human-readable metadata. This resulting binary contains only the essential machine code required for execution. The process of stripping is commonly employed by malware authors as a means to hinder reverse engineering efforts. By eliminating function names, variable names, and other annotations, the analysis of these binaries becomes significantly more challenging. Malware Analysts will need to rely on heuristic techniques, pattern recognition, and dynamic analysis to discern stripped malware's functionality.

Conversely, non-stripped malware retains all or most of its debugging information, symbol tables, and metadata. This includes names for functions, variables, and other high-level information, providing substantial insights into the malware's operation. The presence of such information facilitates reverse engineering, enabling easy comprehension of the malware's structure, flow, and intent. We identify and label stripped binaries based on the existence of symbol tables and output from 'file' utility.

2.2.2. Static and Dynamically Linked Malware

Static linked malware refers to malicious software that incorporates all necessary libraries and dependencies directly into the executable file itself. This means that when the malware is executed, it does not rely on external libraries or shared resources from the underlying system. Instead, everything the malware needs to run is bundled within its binary. This bundling includes functions, routines, and other components required for its operation as shown in Figure 2. Static linking can increase the size of a malware's executable, as it incorporates all necessary code directly into the binary. This approach ensures that the malware operates independently on any system, eliminating potential compatibility and versioning issues with external dependencies.

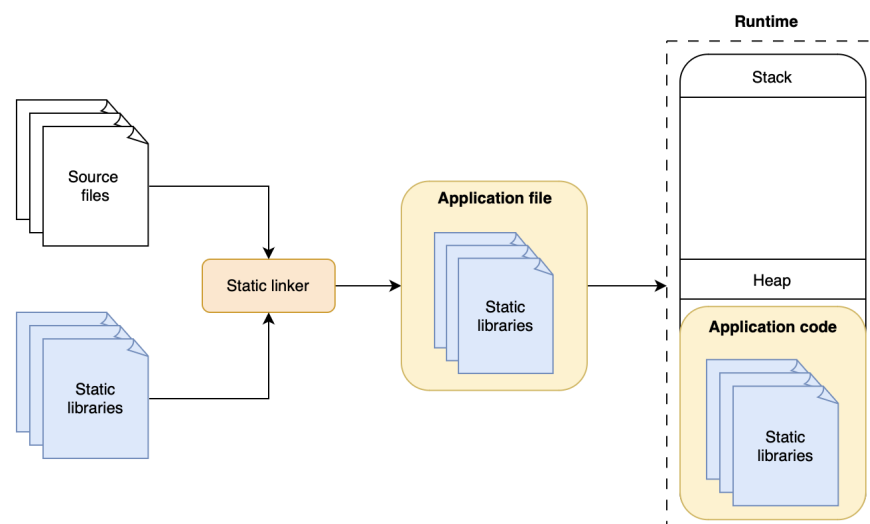


Figure 2. Inner workings of a statistically linked binary.

Dynamically linked malware, on the other hand, relies on external libraries and resources that are not included in its binary. When dynamically linked malware is executed, it accesses these libraries and resources from the system's shared libraries or external sources as shown in Figure 3. This approach reduces the size of the malware's binary since it does not need to include all dependencies within itself. However, it also means that the malware is dependent on specific library versions and system configurations. If these dependencies are not met on the target system, the malware may fail to execute or exhibit unexpected behavior.

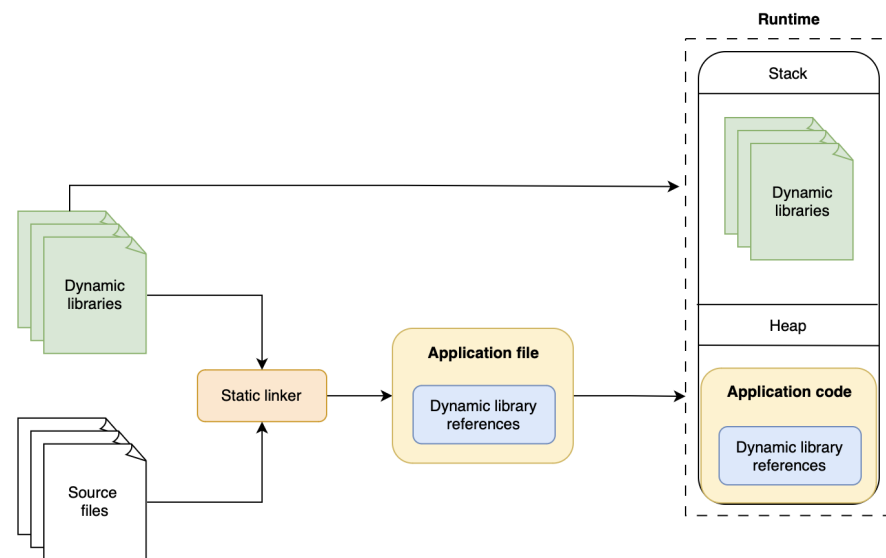


Figure 3. Inner workings on a dynamically linked binary.

3. Literature Review

Given the heavy reliance of global infrastructure systems on the Internet of Things (IoT), IoT devices are frequently exploited as entry points for cyberattacks due to their inherent security flaws. These vulnerabilities have led to the evolution of diverse IoT malware variants. In this section, we review the existing literature on malware classification using opcode sequence analysis and binary lifting.

Cozzi et al. [13] presents the largest study of IoT malware at the date of writing, reconstructing the lineage of IoT malware families using binary code similarity analysis. By tracking the relationships, evolution, and variants of these families, the study applies its technique to a dataset of over 93,000 samples submitted to VirusTotal over a period of 3.5 years. This approach facilitates the identification of various family variants and intra-family relationships due to code reuse. The paper also highlights the constant evolution of these threats by highlighting thousands of minor variations within each malware variant.

In [14], Moon et al. study the detection of IoT malware across different malware families by leveraging opcode sequence analysis. They create fixed-length training features from variable-length sequences with an entropy histogram, generating 2D visual representations that reveal intrinsic characteristics within homogeneous families while also providing robust training features. This visual differentiation aids in distinguishing between benign and malicious software, as well as correlated and uncorrelated malware. Machine learning algorithms such as 5-NN, SVM, Decision Tree, and Random Forest were then employed achieving a mean MCC of over 98.0%. Furthermore, the results also demonstrate that evolved malware can be detected with a model learned from its precedent malware.

In a similar vein, Lee et al. [15] propose a malware detection and family classification methodology. They represent IoT malware with fixed-length and low-dimensional features from opcode category information and their entropy values visualizing them as 2D images to identify patterns. The proposed features are evaluated on several ML models, including

5-NN, SVM, Decision Tree, Random Forest, and MLP yielding over 98% accuracy in malware detection and classification.

Gulmez and Sogukpinar [16] introduce a novel static analysis method for malware detection based on graph representations of opcode sequences. They disassembled PE files to obtain opcode sequences, which were then transformed into graphs. Using the histogram of node degrees within these graphs, they achieved a malware detection accuracy of 98% with machine learning algorithms such as Random Forest, KNN, Decision Tree, and SVM, with Random Forest performing the best. The study also compared the effectiveness of opcode histograms and node degree histograms, finding that the latter provided superior accuracy for malware detection.

Wang and Qian [17] introduce a classification method that leverages semantic features extracted from opcode sequences using word vectors. These sequences are treated as text sentences and fed into a text convolutional Neural network (textCNN) to classify malicious code families. The experimental results demonstrate high accuracy, with over 98% accuracy on the Microsoft Malware Challenge dataset and 91.93% accuracy on the SOREL-20M dataset. The study also optimizes model training speed by selecting key blocks containing call instructions. Overall, the proposed algorithm outperforms traditional byte n-gram representation methods in malicious code classification.

Similarly, HaddadPajouh et al. [18] explore the potential of using Recurrent Neural Networks (RNNs) to detect IoT malware. Their approach employs RNNs to analyze the operation codes (opcodes) of ARM-based IoT applications. Text mining techniques are then used to extract feature vectors from these opcodes. The authors train various machine learning models, including Random Forest, SVM, Naive Bayes, MLP, KNN, AdaBoost, and Decision Tree, using a dataset comprising of 281 malware and 270 benign samples. To evaluate the models, they tested them on 100 new IoT malware samples, using three different Long Short Term Memory (LSTM) configurations. The research findings reveal that the LSTM configuration with 10-fold cross-validation achieves the highest accuracy, reaching 98.18% in detecting new malware samples.

Furthermore, Darabian et al. [19] utilized sequential pattern mining to extract the most frequent opcode sequences in malicious IoT malware and benignware. The detected maximal frequent patterns (MFP) of opcode sequences are then used as the features to differentiate malicious applications from benignware. These features were used to train various machine learning models such as K Nearest Neighbor (KNN), Support Vector Machine (SVM), multilayer perceptron (MLP), AdaBoost, decision tree, and random forest achieving a 99% in malware classification.

Lastly, Kang et al. [20] proposed a methodology for Android malware detection and family classification using opcode n-gram features. They employed machine learning models such as Naive Bayes (NB), Support Vector Machine (SVM), partial decision tree (PART), and Random Forest. The study analyzed sequences up to 10 g, considering both binary counts (indicating the presence of specific n-opcodes in the application) and frequencies (indicating how often each n-opcode is used). The experimental results demonstrated that SVM achieved a 98% F-measure in both malware detection and family classification. Additionally, the authors concluded that binary n-opcodes provide more accurate results compared to frequency-based n-opcodes.

Addressing the challenge of classifying Linux malware across various heterogeneous architectures, Jeong et al. [21] proposes leveraging binary lifting. The core idea in this paper is to translate the binary codes of different architectures into a high-level intermediate representation (IR) using binary lifting. This creates a unified format for analyzing malware, regardless of the underlying hardware architecture. The translated IR sequences, which encapsulate malicious behavior patterns, are then fed into a deep learning model, specifically an LSTM (Long Short-Term Memory) model, for sequence learning achieving a 94% accuracy in detection and classification of various types of malware (rootkit, backdoor, worm, virus, etc.).

Our research stands out from the existing literature in several key aspects, as shown in Table 1. Unlike many studies that focus on specific architectures like ARM or PE files, our approach addresses a broader spectrum of architectures, allowing for a more comprehensive analysis. Moreover, our model achieves a competitive accuracy of 97% using Random Forest, showcasing its effectiveness. Additionally, we utilize ESIL from radare2 for Binary Lifting, offering support for a wide range of architectures and maintaining simplicity by working directly with opcode sequences without operands. What sets our research apart is the emphasis on function-wise analysis within each binary and across the dataset, coupled with rigorous statistical tests to establish the significance of differences in opcode sequences between malware and benign subsets. These unique aspects contribute to the robustness and depth of our analysis compared to the existing literature.

Table 1. Comparative study of research works that employ static analysis using opcode.

Research Work	Architecture	Features Used	Accuracy	Comparison
Moon et al. [14]	ARMv6-M	Opcode category sequences, entropy histogram	5-NN, SVM, RF (AUC: 0.99), DT (AUC: 0.97)	ARM ISA-specific. Needs opcode to be categorized.
Lee et al. [15]	ARM CPU	Sequence of opcode categories and entropy values	RF (99%)	ARM ISA-specific. Needs opcode to be categorized.
Gulmez et al. * [16]	PE files	Opcode sequences graphs	RF (98%)	PE files only. Creates graphs, subgraphs and then histograms from opcode sequences.
Wang and Qian * [17]	PE files	Vectors of opcode sequences	textCNN (98%)	PE files.
HaddadPajouh et al. [18]	ARM	Feature vector of opcodes	LSTM (94%)	ARM ISA specific.
Darabian et al. [19]	ARM	Maximal frequent patterns (MFP) of opcode sequences	Adaboost & DT (99%), MLP & KNN (96%)	ARM ISA-specific. Opcode categorization required. MFP opcode ranking based on its frequency.
Jeong and Kwak ** [21]	Multi-architecture	Opcodes IR	RNN, LSTM (94%)	Uses B2R2 for Binary Lifting and converts opcode + Operands into LowUIR representation. Results in a large amount of data (Figure 4 illustrates the LowUIR translation for one mov instruction) and therefore computationally intensive.
Our research	Multi-architecture	Opcodes IR	RF (97%)	Our research uses ESIL from radare2 for Binary Lifting and therefore supports a wide range of architectures. Use of just opcode sequence without operands or conversion to provide a high-level abstraction. Function-wise analysis—relative function analysis within each binary, and across the dataset. We conduct statistical test to establish that the opcode sequence between malware and benign subsets are significantly different.

“*” These works are for Windows binaries and do not address ELF binaries and therefore are not directly related to the current work. “***” The authors of this research use Binary Lifting technique but opted to generate LowUIR sequence from B2R2 (tool used for binary lifting). They also include operands in addition to opcode which is computationally intensive. Because of this, a single assembly instruction is translated to significant amount of data as illustrated in their research work. Our research uses only the opcode IR sequence to provide an abstract representation of each function, making it more efficient, less computationally intensive, and resulting in a better F1 score.

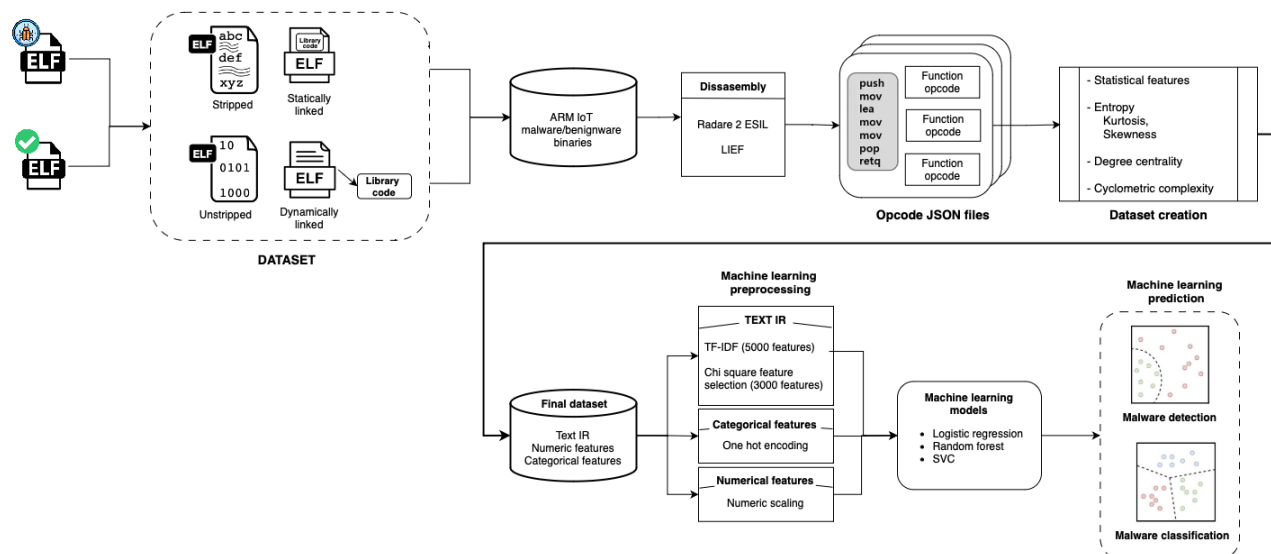


Figure 4. Workflow of the proposed approach with a dataset of raw Malware and Benign ELF binaries

4. Methodology

4.1. Dataset

This research uses ARM architecture binaries from the dataset described in [22]. It includes 65,956 open-source IoT malware binaries identified over a span of 14 years. This dataset features 1006 unique malware threat labels and is designed to encompass 15 different architectures. From this dataset, we extracted a labeled subset of ARM architecture-based ELF binaries for this study, consisting of both IoT malware variants and benign binaries, for the purpose of function analysis.

The ARM architecture binaries subset includes various malware variants such as Mirai, Gafgyt, Tsunami, Benign, Generica, Dofloo, and Jiagu. However, due to significantly less number of variants for some of these malware, we focused on the top three malware variants (Gafgyt, Mirai, and Tsunami) along with benign binary samples. Also, related works on IoT malware variants for ARM architecture have all focused on these malware. This also helped avoid extreme imbalance within our dataset. Gafgyt, Tsunami, and Mirai are all types of malware that target Internet of Things (IoT) devices, primarily using them to form botnets for various malicious activities.

Mirai, Gafgyt, and Tsunami are among the most prevalent threats targeting ARM architecture in IoT devices.

Mirai is a type of malware that primarily targets Internet of Things (IoT) devices such as routers, security cameras, and DVRs. It was first discovered in 2016 and became infamous for its role in launching massive Distributed Denial of Service (DDoS) attacks. Numerous variants have emerged since its source code was released publicly. These variants often include modifications to evade detection, add new exploits, or improve attack capabilities.

Gafgyt, also known as Bashlite, is another IoT-targeting malware that emerged around the same time as Mirai. Variants of Gafgyt have been developed to exploit different vulnerabilities and improve the efficiency of the botnet.

Tsunami, also known as Kaiten, is a malware that has been around since the early 2000s. It targets Linux-based systems and has been adapted to exploit IoT devices. While originally targeting general Linux systems, newer variants have been adapted to target ARM-based IoT devices. Variants have been developed to include additional exploits and to use more advanced command and control mechanisms.

To further refine the dataset, we selected a random sample of binaries containing fewer than 2000 functions. This step ensured that the number of functions per binary was manageable for batch analysis during reverse engineering. Following these stages, we

compiled the final dataset used in this research. Table 2 presents the number of ARM architecture malware variants that were reverse engineered for function analysis.

Table 2. Counts of binary samples.

Class	Binary Count	Function Count
Gafgyt	509	146,214
Mirai	490	79,273
Tsunami	268	78,037
Benign	238	39,358

Once we extracted our dataset, we reverse engineered the binaries to retrieve its functions. Functions are discrete blocks of code within the binary that perform specific tasks. Reverse engineering and analyzing these functions is crucial for understanding malware behavior. The reverse engineering of these binaries is carried out using radare2 and LIEF libraries which are instrumental in malware analysis.

Radare2's ability to analyze multiple architectures through its Intermediate Representation (IR) instruction in ESIL (Evaluable Strings Intermediate Language) aids in architecture-agnostic analysis. The advantages of binary lifting after disassembling is highlighted in Table 3.

Table 3. Comparison of our approach with Binary Lifting to architecture-specific disassembly.

Binary Lifting Approach	Architecture-Specific Disassembly
Allows analysis across different architectures without the need for architecture-specific tuning.	Tools and techniques can be optimized for a specific architecture, potentially providing deeper insights.
Increases scalability since we can target multiple platforms.	Fine-tuned optimizations for specific architectures.
Consistent framework and abstraction, which is efficient for comparative analysis.	Requires continuous updates and maintenance to accommodate different architectures.
Enables detection and classification of malware across a wide range of devices and environments.	Developing and maintaining multiple frameworks is tedious and resource-intensive.
Adds an additional processing step to abstract.	Multiple frameworks and toolsets can lead to fragmentation.

We assess whether an ELF binary is statically linked by examining references to libraries identified during the reverse engineering process. Additionally, we ascertain if a binary is stripped of symbols by checking for the presence of a symbol table in the binary sections, supplemented by findings from the 'file' utility. The labels for stripped and whether the binary is statically linked is determined based on best-effort, and therefore not definitive. The methodology workflow we used for this paper is outlined in Figure 4.

4.2. Static Analysis and Feature Extraction

In functional static analysis, function metadata such as the signature, number of basic blocks, cyclomatic complexity, and degree of centrality are extracted. A key aspect of our methodology is the use of radare2's Intermediate Representation (IR), specifically the ESIL (Evaluable Strings Intermediate Language), which provides an architecture-agnostic format for analyzing assembly code. This allows for a generic implementation to analyze functions.

For each function, we extract all IR opcodes maintaining the sequence and frequency. This data is encapsulated into a structured JSON format, ensuring each malware variant function is comprehensively documented. The primary dataset is then constructed with rows representing individual functions, each row tagged with identifiers like binary_id and binary hash, along with the extracted opcode sequences and additional function metadata.

4.3. Feature Engineering

In the feature engineering phase of our analysis, we include statistical metrics that attempts to capture the complexity and behavior of each function within the binaries. These metrics help in distinguishing malware and benign binaries while providing insights into the similarity of code patterns.

- **IR opcode entropy:** In information theory, entropy measures the randomness or unpredictability of data and mathematically represented as shown in Equation (1). We calculate the entropy of the sequence of IR opcodes for each function in the binary. Many related works have used entropy for code obfuscation and other encryption-related code.

$$H(X) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i) \quad (1)$$

where $p(x_i)$ is the probability of event x_i , and n is the number of different events. The logarithm is traditionally base 2.

- **Skewness of opcode Entropy distribution:** Skewness is a measure of the asymmetry of the probability distribution. Skewness helps us understand the distribution of opcode entropy values across all the binary functions. A positively skewed distribution suggests that most functions have lower entropy, with fewer functions exhibiting higher complexity where malicious activity is concentrated. Skewness measures the asymmetry of distribution around the mean and is mathematically represented as shown in Equation (2).

$$\text{Skewness} = \frac{E[(X - \mu)^3]}{\sigma^3} \quad (2)$$

where μ is the mean of the distribution, σ is the standard deviation and E is the expected value.

- **Kurtosis of opcode entropy distribution:** Kurtosis is a statistical measure that describes the tails of a distribution compared to a normal distribution as represented in Equation (3). In our analysis, examining the kurtosis of opcode entropy reveals whether the entropy values are heavily concentrated around the mean or if they are spread out across a wide range of values. High kurtosis in a malware binary indicates the presence of outlier functions either with extremely high complexity or extremely simple behavior such as a single instruction 'ret' or 'jmp' opcode, which are often associated with malicious payloads or complex evasion mechanisms. Low kurtosis is a suggestion of a more uniform distribution of function complexity.

$$\text{Kurtosis} = \frac{E[(X - \mu)^4]}{\sigma^4} - 3 \quad (3)$$

where μ is the mean of the distribution, σ is the standard deviation and E is the expected value. This measures the tail of the probability distribution of the data, with the normal distribution's kurtosis adjusted to zero by subtracting 3.

4.4. Statistical Analysis

By including several statistical features with the cyclomatic complexity (ccomplexity) of the function and ratio of indegree (number of calls to the current function) and outdegree (number of outgoing function/library calls) we provide meaningful features that describe a binary functionality based on static analysis alone, for the machine learning models to train on. To validate the statistical significance of the differences observed between the opcode entropy distributions of malware and benign binaries, we employ the Mann–Whitney (Wilcoxon) U test. This non-parametric test is suitable for skewed data for comparison from two independent groups. Our analysis reveals a negligibly low p -value $< 2 \times 10^{-16}$, confirming that the entropy features can reliably differentiate between malware and benign samples as shown in Figure 5.

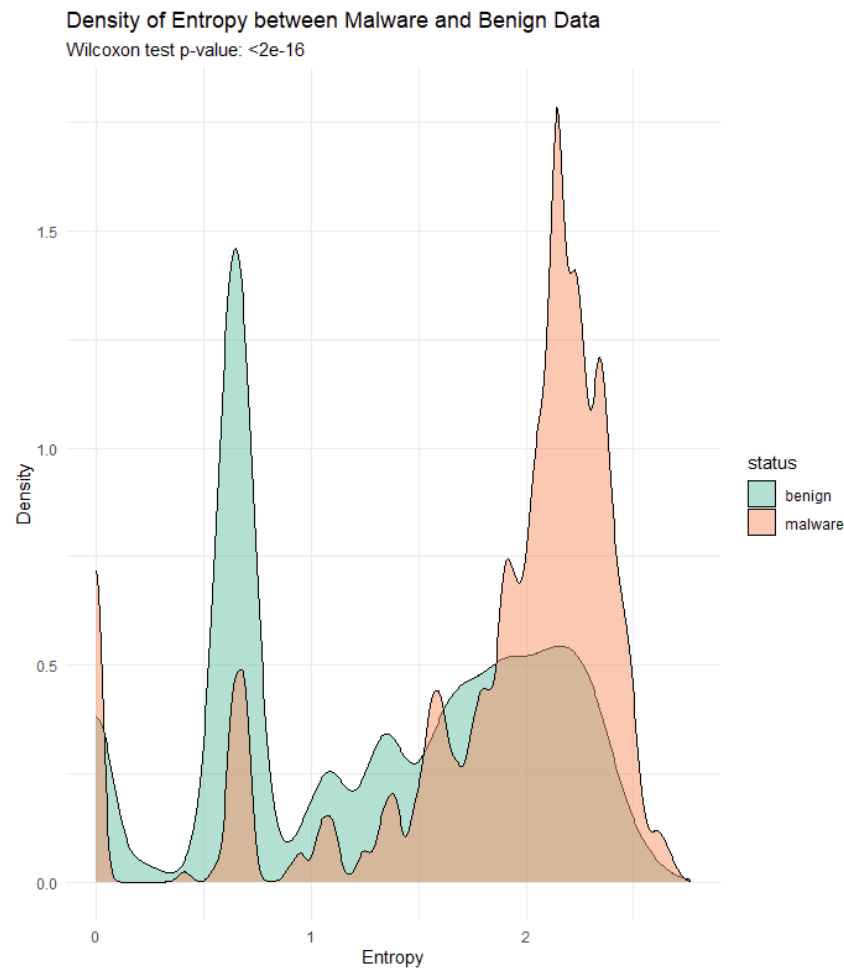


Figure 5. Opcode entropy differences between malware and benign subsets.

4.5. Data Organization

Each function within the ELF binary is indexed under a `binary_id`, facilitating aggregated analysis, and offering a comprehensive view of the overall functionality of the binary. By maintaining the functions within a `binary_id`, we capture behavioral patterns for the entire binary while preserving individual metrics such as complexity and centrality. This organization also supports the relative analysis of functions within a binary. To enable machine learning models to grasp the nuances of functions relative to the binary and across malware classes, the training data are structured in batches by `binary_id`.

The resulting dataset consists of the function and the ELF binary features as outlined in Table 4.

Table 4. Description of the IoT Malware Function Analysis dataset.

Field Name	Description
<code>binary_id</code>	Unique identifier for each binary
<code>hash</code>	Hash of the binary
<code>endian</code>	The endianness of the binary (LSB or MSB)
<code>stripped</code>	Binary attributes indicating if it is stripped of symbol information
<code>StaticLinked</code>	Binary attributes indicating if it is statically linked
<code>Class</code>	Classification of the binary (Mirai, Gafgyt, Tsunami, or benign)
<code>Name</code>	The name of the function within the binary

Table 4. Cont.

Field Name	Description
Type	The type of the function; can be function (fcn), symbol (sym), or location (loc)
Size	The size of the function in bytes
nargs	Number of arguments to the function
centrality, indegree, outdegree, indegree_ratio, outdegree_ratio	Network centrality metrics for the function
nbbs, ebbs	Number of basic blocks (nbbs) and extended basic blocks (ebbs) within the function
complexity	Cyclomatic complexity of the function, indicating the complexity of control flow
IR	Opcode sequence; binary-lifted Intermediate Representation (IR)
bin_avg_nbbs, bin_sd_nbbs, bin_avg_ebbs, bin_sd_ebbs, bin_sd_indegree, bin_sd_outdegree, bin_avg_nargs, bin_sd_nargs	Statistical features calculated across all functions in the binary, including averages and standard deviations for basic blocks, extended blocks, indegrees, outdegrees, and number of arguments
entropy	Function IR opcode entropy, measuring randomness and complexity
skewness_entropy	Skewness measure the asymmetry of entropy distribution across functions in the binary, indicating anomalies
kurtosis_entropy	Kurtosis of the entropy distribution, used to identify outliers and the “tailedness” of the distribution.

As part of our contribution, the Linux IoT Malware Function analysis dataset is available for future research efforts.

4.5.1. Data Pre-Processing for ML Based Classification

With the dataset created as outlined in the previous section, the numerical features, which include function IR opcode entropy, skewness and kurtosis scores, are standardized. The categorical features StaticLinked and Stripped columns, are One-Hot encoded. The delimited IR opcode sequences are parsed and vectorized using Term-Frequency Inverse-Document-Frequency (TF-IDF) vectorizer limited to a maximum of 5000 features. We opt to select the top 1000 features based on their highest scores, using chi-squared statistics which are suitable for multi-classification tasks. The pipeline for the data preprocessing and vectorization strategy is shown in Figure 6.

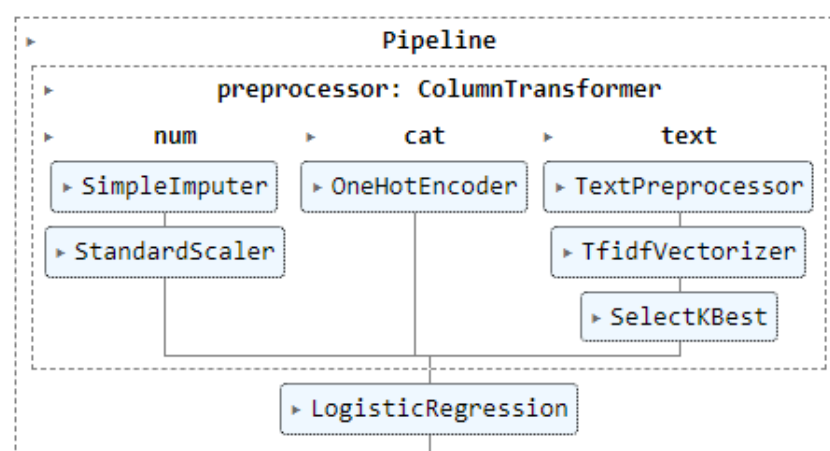


Figure 6. Data pre-processing and vectorization pipeline.

4.5.2. Machine Learning Classification of Malware Variants

The primary focus of this study is to identify and group variants based on their functionality and structure from static analysis without having to execute the malware. To this end, we employed various machine learning classifiers on the dataset we created from reverse-engineering Linux IoT malware and benign ELF binaries and augmented the dataset with additional statistical features.

The objective is to disassemble and reverse engineer malware functions to create a dataset that can be used for malware variant classification and detection irrespective of the Linux architecture with binary lifting. We have evaluated the approach and dataset with typically used ML classification models—Logistic Regression for multi-classification, Random Forest since it is robust to outliers and anomalies which is characteristic of malware, and SVC, which is effective in high-dimensional scenario where the number of dimensions exceeds the number of samples, as is often the case with detailed malware feature sets. We have also evaluated a Neural Network model, specifically MLP, which can capture complex patterns and interactions in the data through layered architecture. The strong performance of these models suggests that our dataset is well-suited for the classification task.

We compare the performance of multiple classifiers, such as Logistic Regression, Support Vector Classifier (SVC), Random Forest, and Neural networks with architectures like MultiLayer Perceptron (MLP).

4.6. Model Hyperparameters

For the dataset we generated, minimal tuning was required, and we mostly adhered to standard values from the sklearn library.

- **Vectorization:**
We used a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer for function IR opcodes, with a maximum of 5000 distinct tokens to extract, and an n-gram range of (1, 1). As indicated in the methodology, we selected the top 3000 features using a chi-squared function to compute the score for each feature. The top 3000 features were chosen for computational efficiency and performance, and can be adjusted in future studies. For numeric features, we employed the mean value for imputation and standardize (normalize) by removing the mean and scaling to unit variance. For categorical features, we handled unknown categories by ignoring them, as we generated and verified the categories for incorrect or null values while extracting from the binaries.
- **RandomForestClassifier:**
Through GridSearch, we identified the optimal number of trees as 200 and also adjusted initial class weights to accommodate class imbalance. The default values were retained for maximum depth of the tree, which was set to None, and Gini impurity is used for the split criteria.
- **LinearSVC**
For LinearSVC, we used the default squared hinge loss function with an l2 penalty and a set the tolerance of 1×10^{-4} for the stopping criteria.
- **MLP Classifier**
For the Neural network model, we employed the rectified linear unit function (ReLU) for activation and the 'adam' solver for weight optimization, with an initial learning rate of 0.001.
By following a logical approach to feature selection backed by statistical validation, the results are comparable across all the models, as discussed in the next section.

5. Results and Analysis

In this section, we compare and analyze the performance of various machine learning classification models on the function analysis dataset comprising IoT malware variants and benign binaries. The primary objective is to evaluate each model's effectiveness in

distinguishing between malware functionalities and benign ELF binaries. To evaluate each model's effectiveness, we utilized the following metrics:

1. Accuracy: Accuracy measures the overall correctness of a model by calculating the ratio of correctly predicted instances to the total instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

- TP = True Positives (correctly predicted positive instances);
- TN = True Negatives (correctly predicted negative instances);
- FP = False Positives (incorrectly predicted as positive instances);
- FN = False Negatives (incorrectly predicted as negative instances).

2. Precision: Precision measures the accuracy of positive predictions by calculating the ratio of correctly predicted positive instances to the total predicted positive instances.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (5)$$

3. F1 score: F1-score is the harmonic mean of precision and recall, providing a balance between precision and recall. It considers both false positives and false negatives.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6)$$

where recall is

$$\text{Recall} = \frac{TP}{TP + FN} \quad (7)$$

The models assessed include Random Forest, Logistic Regression, Support Vector Classifier (SVC), Multilayer Perceptron (MLP) Neural Network, and Long Short-Term Memory (LSTM) Networks. As outlined in Table 5, the Random Forest has the best performance with an accuracy, precision, and weighted F1 score all above 97%, indicating a high level of predictive reliability and consistency. In contrast, the LSTM model, while robust in handling sequence data, showed lower scores across all metrics, suggesting possible challenges in capturing the temporal dependencies within the static features of the dataset. The Logistic Regression and SVC models performed moderately with scores around 90%, which underscores the shortcoming of linear models in handling the nonlinear complexity of malware data. The MLP Neural Network performed better in capturing the non-linear function interactions with an F1 score of 94%.

Table 5. Performance metrics of machine learning models.

Model	Accuracy	Precision	F1 Score (Weighted)
Random Forest	97.17%	97.17%	97.176%
Logistic Regression	90.20%	90.39%	90.23%
SVC	90.09%	90.05%	90.06%
MLP NN	94.26%	94.37%	94.27%
LSTM	89.2%	89.27%	89.23%

Discussion

A closer look at the confusion matrices of the classifiers, as shown in Figures 7–10, provides accurate insights into the different models. For instance, Random Forest Classifier demonstrates the model's ability to classify different malware families and benign binaries with a higher degree of accuracy and precision score in distinguishing between the malware families, i.e., Gafgyt, Mirai, and Tsunami, indicating a strong capability to identify specific

malware characteristics despite their shared functionalities. Interestingly, the false positives could also mean code-reuse or core library functionality.

Malware binaries are characterized by extreme outliers. For example, a Mirai botnet variant has a function with over 600 consecutive ‘and’ opcodes. Random Forest classifier performed the best, with a 97.1% F1 score likely due to the model’s ability to handle outlier data. The Multi-Layer Perceptron (MLP) Neural Network followed with the next-best performance with 94.27%, possibly because of its ability to capture complex patterns among the functions within the binary. This success can be attributed to batching all the functions of a binary in a single batch, allowing the MLP to learn intricate relationships between the functions.

In contrast, the Long Short-Term Memory (LSTM) network showed the worst performance with an F1 score of 89.2%. We evaluated LSTMs, since they are suitable for sequential data and time-series prediction. Unlike Dynamic analysis, the lack of temporal data in our dataset due to the nature of static analysis might have limited its effectiveness. Moreover, LSTMs may require more extensive tuning and larger datasets to fully capture and learn the underlying patterns.

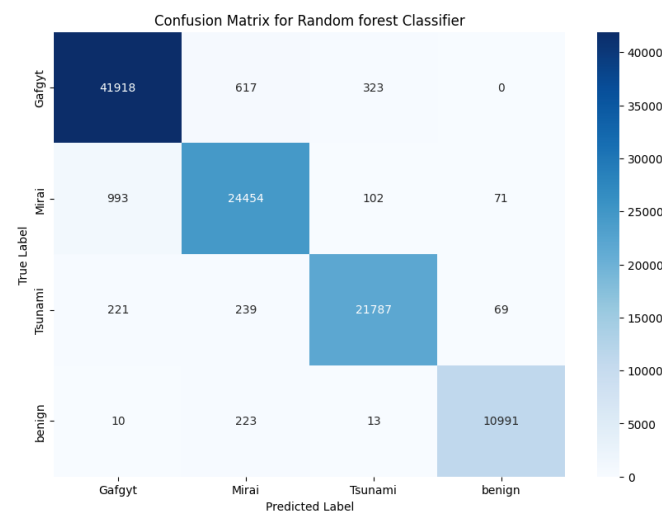


Figure 7. Random Forest classifier confusion matrix

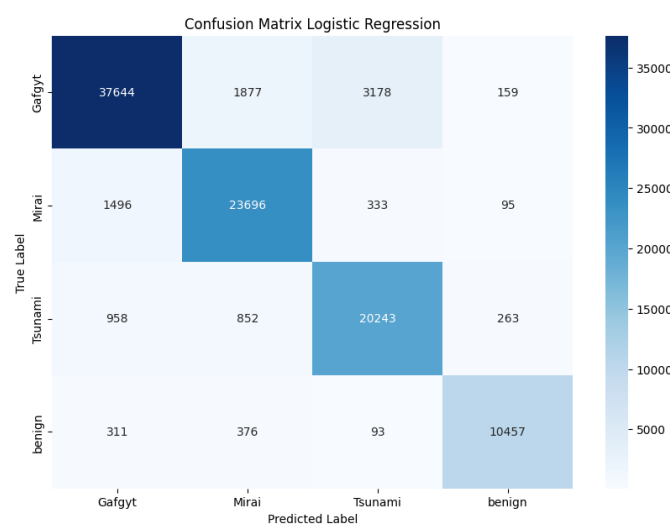


Figure 8. Logistic Regression classifier confusion matrix.

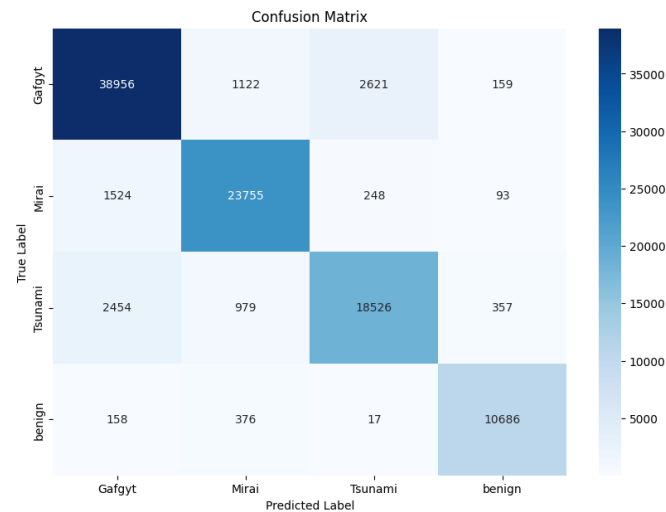


Figure 9. SVC classifier confusion matrix.

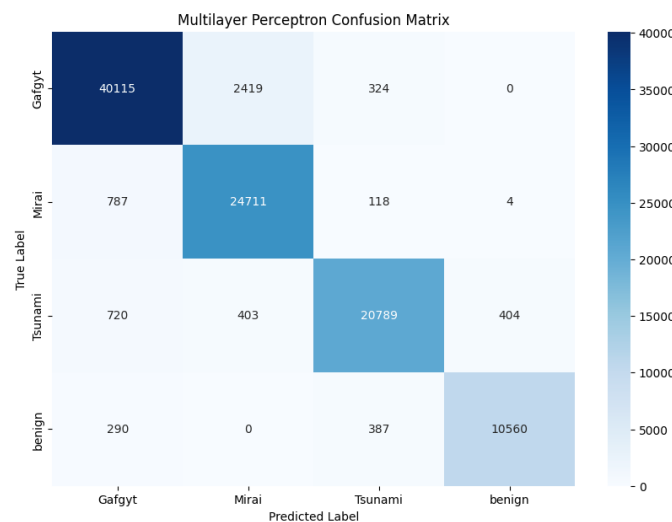


Figure 10. MLP Neural network classifier confusion matrix.

The results suggest that while there are common functionalities within the malware families, for example, use of networking libraries, encryption and other common functionality inherent to malware and benign binaries, the Classifier models are still able to differentiate these from benign behaviors.

6. Conclusions

This research uses an architecture-agnostic approach for detection and classification of Linux IoT malware variants and benign binaries. Our methodology involves the binary lifting of function opcode sequences and applying statistical entropy-based feature set based on Intermediate Representation (IR) function opcodes. We create a dataset with these features, and successfully detect and classify Linux IoT malware variants and benign binaries with multiple machine learning classifiers. The Random Forest classifier model resulted in an F1 score of 97% in detecting and classifying malware variants, including benign ELF binaries.

The proposed approach is computationally efficient as it focuses on analyzing opcode instruction sequences, enabling the extraction of statistically relevant features that abstract malware behavior from functions. Additionally, we validated the viability of this approach to detect and classify malware variants using statistical methods, confirming its effectiveness in practical applications.

Significantly, unlike related works that categorize opcodes based on different versions of the architecture, we propose a generic approach across various architectures, which is practical and reduces computational overhead. The comprehensive dataset created for this study includes function metadata alongside IR opcode sequences and entropy features, providing a solid foundation for systematic ELF binary analysis.

In conclusion, the methodologies and findings from this study provide a foundation for future detection systems that are efficient and capable of adapting to the evolving landscape of IoT malware.

7. Future Works

A limitation of the architecture-agnostic binary lifting approach is its effectiveness in handling proprietary architectures that have unique instruction sets and behaviors. These may not be supported by ESIL or other Intermediate Representation (IR) frameworks, which may result in incomplete or inaccurate results. To address this, it is essential to closely examine the IR from binary lifting to see if the assembly instructions are accurately represented and choose the right toolset for the architecture. However, these are edge cases, as most malware strive to maximize impact by supporting major architectures. Future studies can incorporate dynamic analysis to either complement or compare with the static analysis method and features used in this study, explore deep learning techniques for enhanced feature extraction, and test the models' efficacy across a broader range of IoT device architectures. Given that our approach is architecture-agnostic, we plan to apply this methodology to a diverse dataset of malware ELF binaries from various architectures. This study demonstrates that our computational and statistical approach provides a scalable and effective solution for addressing current and future challenges in IoT security.

Author Contributions: Conceptualization, J.R., K.G. and N.K.S.; methodology, J.R.; software, J.R.; validation, J.R., K.G., N.K.S. and C.V.; formal analysis, J.R.; investigation, J.R. and K.G.; resources, N.K.S. and C.V.; data curation, J.R.; writing—original draft preparation, K.G. and J.R.; writing—review and editing, K.G. and J.R.; visualization, K.G. and J.R.; supervision, N.K.S.; project administration, C.V.; funding acquisition, C.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original dataset presented in the study is openly available at: <https://github.com/jrmoorthy/Linux-Malware-Analysis> (accessed on 14 June 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Howarth, J. 80+ Amazing IoT Statistics (2024–2030)—explodingtopics.com. Available online: <https://explodingtopics.com/blog/iot-stats> (accessed on 9 May 2024).
2. Ngo, Q.D.; Nguyen, H.T.; Le, V.H.; Nguyen, D.H. A survey of IoT malware and detection methods based on static features. *ICT Express* **2020**, *6*, 280–286. [CrossRef]
3. Zscaler ThreatLabz Finds a 400% Increase in IoT and OT Malware Attacks Year-over-Year. Available online: <https://www.zscaler.com/press/zscaler-threatlabz-finds-400-increase-iot-and-ot-malware-attacks-year-over-year-underscoring> (accessed on 9 May 2024).
4. Angrishi, K. Turning internet of things (iot) into internet of vulnerabilities (ioV): Iot botnets. *arXiv* **2017**, arXiv:1702.03681.
5. Costin, A.; Zaddach, J. Iot malware: Comprehensive survey, analysis framework and case studies. *BlackHat USA* **2018**, *1*, 1–9.
6. Antony, A.; Sarika, S. A review on IoT operating systems. *Int. J. Comput. Appl.* **2020**, *176*, 33–40. [CrossRef]
7. AV-ATLAS Malware Portal. 2023. Available online: <https://portal.av-atlas.org/malware> (accessed on 10 May 2023).
8. Shabtai, A.; Moskovitch, R.; Feher, C.; Dolev, S.; Elovici, Y. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Secur. Inform.* **2012**, *1*, 1. [CrossRef]
9. Santos, I.; Brezo, F.; Nieves, J.; Penya, Y.K.; Sanz, B.; Laorden, C.; Bringas, P.G. Idea: Opcode-sequence-based malware detection. In Proceedings of the Engineering Secure Software and Systems: Second International Symposium, ESSoS 2010, Pisa, Italy, 3–4 February 2010; Proceedings 2; Springer: Berlin/Heidelberg, Germany, 2010; pp. 35–43.
10. Hossain, M.M.; Fotouhi, M.; Hasan, R. Towards an analysis of security issues, challenges, and open problems in the internet of things. In Proceedings of the 2015 IEEE World Congress on Services, New York, NY, USA, 27 June–2 July 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 21–28.

11. Lee, Y.T.; Ban, T.; Wan, T.L.; Cheng, S.M.; Isawa, R.; Takahashi, T.; Inoue, D. Cross platform IoT-malware family classification based on printable strings. In Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 29 December–1 January 2021; IEEE: Piscataway, NJ, USA, 2020; pp. 775–784.
12. Liu, Z.; Yuan, Y.; Wang, S.; Bao, Y. Sok: Demystifying binary lifters through the lens of downstream applications. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–26 May 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1100–1119.
13. Cozzi, E.; Graziano, M.; Fratantonio, Y.; Balzarotti, D. Understanding linux malware. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–24 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 161–175.
14. Moon, S.; Kim, Y.; Lee, H.; Kim, D.; Hwang, D. Evolved IoT malware detection using opcode category sequence through machine learning. In Proceedings of the 2022 International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 25–28 July 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–7.
15. Lee, H.; Kim, S.; Baek, D.; Kim, D.; Hwang, D. Robust IoT Malware Detection and Classification Using Opcode Category Features on Machine Learning. *IEEE Access* **2023**, *11*, 18855–18867. [[CrossRef](#)]
16. Gülmez, S.; Sogukpinar, I. Graph-based malware detection using opcode sequences. In Proceedings of the 2021 9th International Symposium on Digital Forensics and Security (ISDFS), Elazig, Turkey, 28–29 June 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–5.
17. Wang, Q.; Qian, Q. Malicious code classification based on opcode sequences and textCNN network. *J. Inf. Secur. Appl.* **2022**, *67*, 103151. [[CrossRef](#)]
18. HaddadPajouh, H.; Dehghantanha, A.; Khayami, R.; Choo, K.K.R. A deep recurrent neural network based approach for internet of things malware threat hunting. *Future Gener. Comput. Syst.* **2018**, *85*, 88–96. [[CrossRef](#)]
19. Darabian, H.; Dehghantanha, A.; Hashemi, S.; Homayoun, S.; Choo, K.K.R. An opcode-based technique for polymorphic Internet of Things malware detection. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5173. [[CrossRef](#)]
20. Kang, B.; Yerima, S.Y.; McLaughlin, K.; Sezer, S. N-opcode analysis for android malware classification and categorization. In Proceedings of the 2016 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), London, UK, 13–14 June 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–7.
21. Jeong, H.S.; Kwak, J. Massive IoT Malware Classification Method Using Binary Lifting. *Intell. Autom. Soft Comput.* **2022**, *32*, 467–481. [[CrossRef](#)]
22. Olsen, S.H.; OConnor, T. Toward a Labeled Dataset of IoT Malware Features. In Proceedings of the 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Torino, Italy, 26–30 June 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 924–933.

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.