On the Use of Machine Learning for Modern IoT ELF Malware Detection

Cristian H. M. Souza, Daniel Macêdo Batista
Department of Computer Science
University of São Paulo, Brazil
{cristian.souza, batista}@ime.usp.br

***Abstract*—The growing adoption of Internet of Things (IoT) devices has elevated concerns about malware threats, as the limited resources of these devices and the diversity of vendors complicate the design of robust security mechanisms. In this paper, we address the challenge of detecting IoT ELF malware by combining static and dynamic features; our dataset is made of 5,169 unique files — 3,306 malicious samples and 1,863 benign Linux binaries. We then train and compare seven machine learning classifiers: Random Forest, Support Vector Machine, Multi-Layer Perceptron, K-Nearest Neighbors, Logistic Regression, Naive Bayes, and XGBoost. Our results show that Random Forest achieves the highest accuracy (99.33%), while providing a 82.15ms mean classification time. The K-Nearest Neighbors (97.33%) and the Logistic Regression (96.80%) closely follow. We make the dataset publicly available to encourage further research in the field. Overall, our findings underscore the importance of combining static and dynamic features to enhance the detection of modern malware threats targeting IoT and industrial control system environments.**

***Index Terms*—machine learning, malware detection, internet of things, IoT, ELF**

I. Introduction

Security flaws in Internet of Things (IoT) devices can be critical to users' privacy and personal safety. The use of insecure protocols can expose users' sensitive information to attackers through network traffic interception. Furthermore, a compromised device (e.g., in an industrial network or other cyber-physical systems) can be leveraged to perform actions in the physical environment, posing risks to human lives [1].

The diversity of manufacturers and IoT devices hinders the implementation of effective security features for threat detection, since such devices have limited storage and processing capacities [2]. In addition, the heterogeneity of protocols requires the use of adaptive approaches and a holistic view of the infrastructure for effectively mitigating modern threats [3].

Malicious software remain one of the main challenges in securing computing systems. The advent of the IoT paradigm was accompanied by an increase in the number of malicious programs designed for ARM (Advanced RISC Machine) and MIPS (Microprocessor without Interlocked Pipeline Stages) architectures [4]. An example of this is the use of the Mirai botnet [5], whose goal is to infect and control embedded devices (such as IP cameras and routers) in order to perform distributed denial-of-service (DDoS) attacks. More recently, in January 2025, TrendMicro alerted about large-scale DDoS attacks orchestrated by an IoT botnet exploiting vulnerable IoT devices such as wireless routers and IP cameras [1].

These threats are responsible for various damages, such as the compromise of data integrity, theft of confidential information, and financial losses for users and corporations. Recently, ransomware have been widely used by criminals to block access to files through data encryption, demanding a payment to retrieve the data [6]. The effectiveness of ransomware attacks gave rise to the criminal market of Ransomware-as-a-Service (RaaS), making the purchase of advanced threats accessible to everyone.

According to the *ICS and OT Threat Predictions for 2024* [7], published by Kaspersky Lab, a leading company in cybersecurity, ransomware continue to be among the main threats to industrial environments. Meanwhile, the *2024 Incident Response Analyst Report* [8], published by the same organization, highlights that 24% of incident response requests worldwide in 2024 occurred in industrial environments, and that one in every three attacks involved the use of malware. Furthermore, the widespread use of IoT in combination with 5G technology has raised numerous discussions about the security of such devices and the environments in which they operate [9].

Several advances have been and continue to be made by industry and academia in the areas of malicious artifact analysis and detection [10]. Detection tools must be able to identify and contain unknown threats. To achieve this, multiple analysis techniques must be employed to increase the reliability of solutions, with behavioral analysis being the most effective at detecting zero-day malware [11].

In this context, and since many IoT devices are based on Linux and run ELF binaries, this work focuses on detecting IoT ELF malware through combined static and dynamic analysis. We use a dataset of 5,169 binaries, compare seven classification algorithms based on Machine Learning (ML), and assess both detection accuracy and classification speed. Our results show that Random Forest achieves the highest accuracy (99.33%) and good mean classification time (82.15ms), followed closely by K-Nearest Neighbhors and Logistic Regression. As an addition contribution, we share the dataset as open data.

[1]https://www.trendmicro.com/en_us/research/25/a/iot-botnet-linked-to-ddos-attacks.html

The rest of this paper is organized as follows: we explore the necessary background in Section II, review related work in Section III, describe the data collection procedure in Section IV, detail the methodology and experimental setup in Section V, analyze results in Section VI, and conclude with final remarks and potential future directions in Section VII.

II. Background

Malware analysis is fundamentally important to information security because, by understanding the artifact's behavior and Assembly code, efficient detection and mitigation tools can be developed [12]. However, the time an analyst spends dissecting a malicious binary and deploying rules into the developed solution may be unfavorable to the end user.

As specified by [13], malware analysis typically requires that the chosen solution have a strong capacity to classify new files based on previous investigations. The authors highlight the diversity of available tools and note that choosing the wrong one can delay the analysis process. Furthermore, the very purpose of analysis may differ: while one analyst might be interested only in determining whether an artifact is malicious, another analyst might delve deeper, aiming to discover which malware family it belongs to.

The analysis of a malicious program can be performed either statically or dynamically. It is important to note that one strategy does not replace the other; both should be employed to gain a complete understanding of the malware in question.

A. Static analysis

Static analysis is commonly the first step in the process, during which the analyst inspects the artifact in detail without executing it. This approach can be divided into basic and advanced techniques.

Basic static analysis can confirm whether a file is malicious and provide information on its functionality, as well as data that can be used to generate signatures. In this phase, the file type, its hashes, headers, readable strings, function imports from shared libraries, the use of packers, and other characteristics are identified. Solutions such as PEStudio[2] and DIE[3] can be used for an initial evaluation of the binary. However, basic analysis is ineffective against sophisticated threats.

Advanced static analysis, in turn, involves reverse-engineering the malware to understand its behavior. This can be achieved by analyzing its instructions in a disassembler. It is important to emphasize that this type of analysis requires deep knowledge of the Assembly code specific to the architecture for which the binary was compiled, as well as of the malware's target operating system. Tools such as IDA[4], gdb[5], and radare2[6] are widely used in this category.

[2]https://www.winitor.com

[3]https://github.com/horsicq/Detect-It-Easy

[4]https://hex-rays.com/ida-free/

[5]https://www.sourceware.org/gdb/

[6]https://rada.re/n/

B. Dynamic analysis

Unlike static analysis, this stage involves executing the malicious artifact in a controlled environment, known as a sandbox, with the goal of monitoring its behavior. Dynamic analysis is also divided into basic and advanced techniques.

In basic dynamic analysis, the analyst identifies system calls made, processes running, DNS queries, traffic directed to known services (such as HTTP, FTP, and SMTP), files created in the file system, and other events. A commonly used analysis environment in both academia and industry is Cuckoo Sandbox[7]. However, as with basic static analysis, certain details may not be uncovered.

Advanced dynamic analysis enables the extraction of detailed information about the malicious program. At this stage, a debugger (such as x64dbg[8]) is used to examine the binary's instructions. In this way, it is possible to step through the operations one by one, as well as view the information the malware uses and stores in memory.

C. The ELF format

The Executable and Linkable Format (ELF) serves as the standard binary format on Linux systems. Any program that is compiled, be it an executable or a shared library, ultimately produces an ELF file. This format contains all the necessary details for the program loader (`ld`) to interpret its contents correctly. Within the file, various sections like `.text`, `.data`, and `.bss` each carry out a specific function, helping to organize code and data efficiently [14].

Although the ELF format is known for its modular design and cross-platform compatibility, making it a universal standard across Linux-based systems, this very universality has made it an attractive target for malicious actors. While most malware research has traditionally focused on Windows-based executables, recent trends indicate a growing number of attacks targeting Linux environments, especially those embedded in IoT and Industrial Control Systems (ICS). The consistency and portability of ELF binaries, often seen as strengths, have also introduced new risks. These characteristics facilitate the spread of malware across different devices and architectures, and make reverse engineering attacks more scalable. Despite these growing threats, ELF malware remains underexplored in academic literature. This gap creates a critical blind spot in the cybersecurity landscape, particularly considering the widespread use of embedded Linux in IoT and ICS environments. A focused investigation into ELF-based threats is both urgent and necessary to anticipate evolving attack patterns and develop more effective defense strategies.

For detecting ELF malware, an automated examination of both the static and dynamic properties of the file can be carried out. Moreover, depending on how the malware is packaged, its malicious behavior may only become evident during runtime. Therefore, it is essential for machine learning models to consider both sets of features.

[7]https://cuckoosandbox.org

[8]https://x64dbg.com

III. Related work

Various efforts were made by the scientific community in the field of malware detection using machine learning techniques. In [15], the authors introduce an approach that uses system call data for detecting specially packed or obfuscated malware. The best results are achieved using Multi-Layer Perceptron (MLP) with a detection accuracy of 98.57%; however, the dataset used by the authors only contained 524 malware samples.

The study by [16] proposes a framework that uses static analysis to detect and classify malware into known categories. The system relies on opcode features and is trained on 6,000 samples. Although the system achieved an accuracy of 98%, relying solely on static features can be a drawback for detecting more stealthy samples. Similarly, [17] introduces a solution that extracts static features from ELF binaries for malware identification; the dataset used contained 1,773 malicious files and the approach achieved an accuracy of 99%.

Unlike other solutions, our work considers both static and dynamic attributes of malicious artifacts for classification. In addition, we compare different algorithms implemented, aiming to guide professionals in choosing the most appropriate models. Finally, we make the dataset used public in order to disseminate and encourage new research.

IV. Data collection

For constructing the dataset used in this study[9], we developed a Python script[10] capable of collecting malicious samples from the MalwareBazaar[11] project, which is a malware sample exchange platform. It is worth noting that when collecting data from this kind of source, other files, such as shell scripts, Windows PE files, and various other formats are also gathered. As a result, we specifically filtered for ELF files, which can be recognized by the signature `0x464c457f`. This resulted in 3,306 unique samples.

We then checked the files for details about the target architecture. As shown in Fig. 1, ARM stands out at 49.73%, followed by x86_64 and MIPS.

We also collected information about threat verdicts of the samples from VirusTotal[12] using Kaspersky's engine, as shown in Fig. 2. We observed that most of the collected samples belong to the Mirai and Gafgyt malware families, reflecting the threats that are mainly used to infect IoT environments.

For collecting the goodware samples, we relied on the artifacts present in pristine Linux systems, located at `/bin`, `/sbin`, and `/usr/bin`; resulting in a total of 1,863 artifacts. Table I shows the amount of samples in the final dataset.

V. Methodology

This section presents our methodology for IoT ELF malware detection, encompassing both feature extraction and classification using multiple machine learning algorithms. We first detail how we derive both static and dynamic features from ELF binaries, then describe the various models implemented. The implementation is evaluated in detail in Section VI.

[9]https://github.com/cristianzsh/malware-research

[10]https://gist.github.com/cristianzsh/2e88b3c33f58a9b83e268d0050eadbdb

[11]https://bazaar.abuse.ch

[12]https://www.virustotal.com

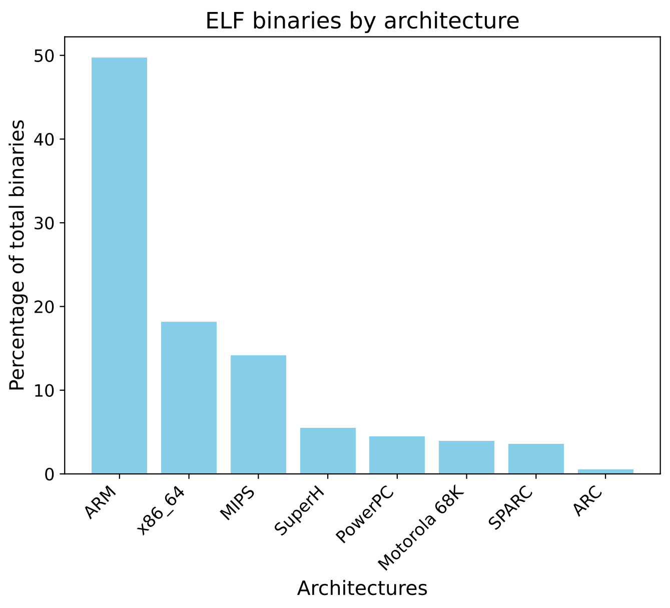

Fig. 1. Malware architectures.

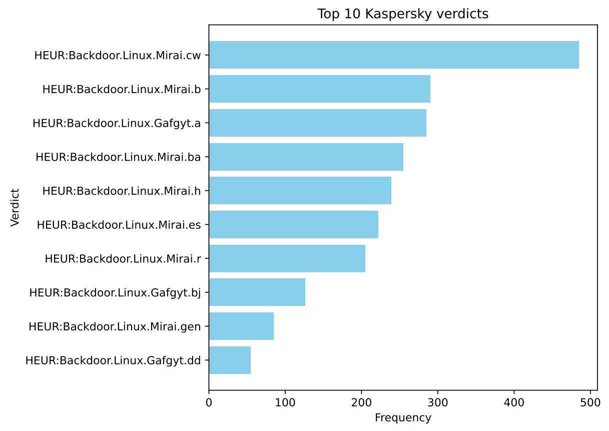

Fig. 2. Malware verdicts.

A. Feature Extraction

Feature extraction proceeds in two phases, gathering static and dynamic indicators and populating two types of feature lists: discrete (i.e., boolean flags indicating presence or absence of a property) and continuous (i.e., integer counts or sizes). Unlike discrete variables, which can only have a predetermined range of values, continuous variables can have an endless number of possible values. Each ELF file is abstracted as a binary object, storing the original label (goodware or malware) and the extracted features.

TABLE I
DISTRIBUTION OF SAMPLES

Type	Count	Percentage
Malware	3,306	64.0%
Goodware	1,863	36.0%
Total	5,169	100.0%

a) Static analysis: Static indicators are obtained by inspecting the ELF structure and performing string analysis:

- **Discrete features**: We set flags for properties such as whether the binary is packed with UPX, statically linked, contains no symbols in its symbol table, or references system-critical paths (e.g., `/proc`, `/sys`, or files like `/etc/passwd` and `/etc/shadow`). We also check whether any compiler strings (such as "GCC" or "CLANG") appear frequently. In addition, there is a dedicated component to parse strings extracted from the binary to search for URLs, IP addresses, and email addresses. Any findings here can be used to set flags or increase counters that represent suspicious network-related features.
- **Continuous features**: We count occurrences of certain substrings (such as the number of references to `/home` or `/var`), measure ELF section sizes (like `.text` or `.data`), determine the total binary size in megabytes and note how many dynamic libraries the binary links are compared to. We also record the number of ELF sections, the number of symbols in `.symtab`, and the number of relocation entries to capture more granular structural information.

b) Dynamic analysis: To approximate runtime behavior, we scan the binary's raw content for keywords suggesting process manipulation, system calls, and signals. Table II presents a full description of the syscalls and signals considered. These checks collectively aim to reveal suspicious or malicious runtime capabilities that the ELF file might exhibit if executed.

B. Algorithms Implemented

After extracting the aforementioned feature sets, we feed them into a suite of machine learning algorithms. Each algorithm is trained to distinguish malicious ELF files from benign ones, leveraging the static and dynamic features. Our classification pipeline is developed entirely in Python, making extensive use of the `scikit-learn`[13] library for training and evaluation. We also employ the Python implementation of `XGBoost`[14] for gradient boosting. A common superclass centralizes actions like data splitting, training, testing, and storing metrics, while subclassing allows the system to flexibly switch between algorithms. Below we outline the principal models used and key hyperparameters:

- **Random Forest (RF):** We utilize an ensemble-based approach with ten decision trees. Each tree is allowed to grow without a strict depth limit, relying on impurity-based criteria to split nodes. This configuration offers a balance of simplicity and performance, while remaining flexible for further tuning (e.g., by adjusting the number of trees).
- **Support Vector Machine (SVM):** This classifier can use either a linear or non-linear kernel, with a default cap of 1000 iterations. In the linear case, training focuses on finding a hyperplane that best separates malicious from benign samples. If a non-linear kernel is selected, a parameter for kernel shape is automatically determined, making the model adaptable to more complex feature boundaries.
- **Multi-Layer Perceptron (MLP):** A feedforward neural network is trained with two hidden layers of ten neurons each. Network optimization uses the Adam method, and we employ an additional regularization term (with strength $\alpha = 0.1$) to mitigate overfitting. The network's random seed is kept fixed to enhance reproducibility across runs.
- **K-Nearest Neighbors (KNN):** Classification is determined by the five nearest examples in the feature space. The label of a sample is decided by majority vote among these neighbors, giving a straightforward measure of similarity-based classification without requiring a parametric model.
- **Logistic Regression (LR):** This technique uses a regularization mechanism to prevent overfitting by discouraging excessively large model coefficients. Specifically, it penalizes the sum of the squared values of the coefficients, promoting simpler and more generalizable models. The training process runs for up to 1000 iterations or stops earlier if convergence is reached, typically achieving a good balance between performance and interpretability.
- **Naive Bayes (NB):** A Gaussian-based variant is used, treating each feature as though it follows a normal distribution. This probabilistic approach is lightweight and fast, providing a solid baseline when dealing with continuous features extracted from each ELF binary.
- **XGBoost (XGB):** Gradient boosting is conducted with a binary logistic objective, focusing on iterative refinement of weak learners. The performance metric relies on a log-loss function, which incentivizes accurate probability estimates. This method typically excels in handling tabular data containing mixed discrete and continuous attributes.

All models are evaluated using standard confusion-matrix statistics (true positives, false positives, true negatives, false negatives) and yield accuracy, precision, recall, and F1 scores. At the end of training, the selected model is saved using Python's serialization utilities, so that it can be readily reloaded for testing on new binaries. This design allows rapid substitution of one algorithm for another, facilitates parameter fine-tuning, and supports consistent performance tracking across different classification approaches.

[13]https://scikit-learn.org
[14]https://xgboost.readthedocs.io

TABLE II
SYSTEM CALLS AND SIGNALS CONSIDERED IN DYNAMIC ANALYSIS

Category	**Description**
System calls	
Process and thread manipulation	`fork`: Creates a new process by duplicating the calling process (child process). `execve`: Replaces the current process with a new program to execute another binary.
File and directory operations	`open`: Opens a file for reading, writing, or both, returning a file descriptor. `read`: Reads data from a file or input source into memory. `write`: Writes data to a file, socket, or another process's memory. `rename`: Renames files in the file system.
Network communication	`socket`: Creates a network socket for sending or receiving data. `connect`: Connects a socket to a remote server for communication.
Memory and resource management	`mmap`: Maps a file or device into memory for direct access. `mprotect`: Changes memory protection, like read/write/execute permissions.
Privilege escalation	`setuid`: Sets the user ID of the calling process to gain higher/lower privileges. `capset`: Modifies process capabilities for restricted actions.
System information	`uname`: Retrieves system information (e.g., OS name, version). `getpid`: Returns the process ID of the calling process.
Time and delays	`nanosleep`: Pauses execution for a specified time (in nanoseconds). `clock_gettime`: Retrieves current time from a specific clock.
Anti-debugging techniques	`ptrace`: Detects debugging by checking if a debugger has attached. `getppid`: Checks the parent process ID to identify potential debuggers.
Signals	`SIGTERM`: Termination request, sent to the program. `SIGSEGV`: Invalid memory access (segmentation fault). `SIGINT`: External interrupt, usually initiated by the user. `SIGILL`: Invalid program image, such as invalid instruction. `SIGABRT`: Abnormal termination condition (e.g., initiated by `abort()`). `SIGFPE`: Erroneous arithmetic operation such as divide by zero.

VI. RESULTS

Evaluating machine learning models using appropriate metrics is essential for assessing their overall quality. These metrics help determine whether a model adequately fits the data and meets the specific objectives of the task at hand [18]. Below are the most commonly used metrics considered in evaluating the models proposed in this study.

Accuracy indicates the overall performance of the model. It is the number of correct predictions divided by the total number of examples. For instance, if out of 100 observations, 90 are classified correctly, the model has an accuracy of 90%. **Precision** is calculated as the ratio between the number of examples correctly classified as positive and the total number of examples classified as positive. This metric places greater emphasis on errors resulting from false positives. **Recall** emphasizes errors caused by false negatives. This metric (also known as sensitivity) is defined as the ratio between the number of examples correctly classified as positive and the total number of examples that are actually positive. **F1-Score** is defined as the harmonic mean of precision and recall. A low F1-Score indicates that the model's precision or recall is low. Therefore, this metric serves as a more comprehensive indicator of the model's effectiveness, although it is less intuitive than accuracy.

The evaluations were carried out on a machine with an Intel Core i7-11800H CPU running at 2.30GHz (8 vCPUs), 32GB of RAM, and Ubuntu Server 24.04.1 LTS 64-bit (Linux Kernel 6.8.0-41). The training and testing data sets were split into the 7:3 proportion.

Table III compares the performance of the considered classification algorithms: Random Forest, SVM, MLP, KNN, Logistic Regression, Naive Bayes, and XGBoost. Random Forest attains the highest accuracy (99.33%), followed by KNN (97.33%) and Logistic Regression (96.80%). These three top-performing models also demonstrate strong precision, recall, and F1-Scores, indicating high reliability across multiple metrics.

The remaining algorithms still achieve competitive results, with SVM reaching 95.00% accuracy and Naive Bayes 92.83%, the latter showing notably high precision (0.97) and recall (0.99) but slightly lower overall accuracy. XGBoost follows at 90.98% accuracy, maintaining balanced precision, recall, and F1-Scores. MLP lags behind at 87.95% accuracy, suggesting potential challenges in network design or hyperparameter tuning. Despite varying results, each model exhibits particular strengths, reinforcing the importance of evaluating multiple algorithms to identify the most suitable choice for a given classification task.

We also evaluated the algorithms in terms of classification time. Table IV shows the mean classification time, as well as the lowest and highest records for each model. Overall, Naive Bayes achieved the lowest mean classification time (79 ms), while MLP required the longest on average (92 ms).

TABLE III
PERFORMANCE METRICS FOR CLASSIFICATION ALGORITHMS

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Random Forest	99.33%	0.97	0.99	0.98
SVM	95.00%	0.93	0.90	0.91
MLP	87.95%	0.88	0.86	0.87
KNN	97.33%	0.98	0.96	0.97
Logistic Regression	96.80%	0.96	0.96	0.96
Naive Bayes	92.83%	0.97	0.99	0.98
XGBoost	90.98%	0.91	0.90	0.90

Logistic Regression presented the smallest minimum time (65 ms), whereas MLP showed the highest maximum time (113 ms). Random Forest, the best in terms of accuracy, required the third smaller mean time (82.15ms). Despite these differences, all methods remained relatively close (within a range of roughly 30 ms), indicating that while some models have a slight speed advantage, classification time does not vary dramatically across the tested algorithms.

TABLE IV
CLASSIFICATION TIME (MS) FOR EACH ALGORITHM

Algorithm	**Mean (ms)**	**Min (ms)**	**Max (ms)**
Random Forest	82.150	68.184	93.076
SVM	85.845	69.797	103.146
MLP	91.971	73.474	112.544
KNN	86.344	72.969	101.106
Logistic Regression	82.631	65.084	99.746
Naive Bayes	79.247	69.547	88.131
XGBoost	79.903	73.474	86.163

VII. CONCLUSION

This study presented an approach for IoT malware detection using static and dynamic features from the binaries. By analyzing a diverse collection of 5,169 binaries, we demonstrated that leveraging multiple sources of the sample (file structure, strings, and syscall-related indicators) significantly boosts detection performance.

Among the seven classifiers tested, Random Forest achieved the highest accuracy at 99.33%. Although it is not the fastest (with a mean classification time of approximately 82ms) it still provides near real-time predictions suitable for many IoT scenarios. Despite slight performance variations across models such as KNN, Logistic Regression, and Naive Bayes, the consistently strong results underscore the viability of these algorithms for practical IoT security.

Future directions include refining the feature extraction process to capture more elaborate anti-analysis or obfuscation techniques, as well as incorporating additional architectures and malware samples to enhance model generalization. By releasing our dataset and methodology to the broader community, we hope to inspire further research and foster improvements in safeguarding IoT and ICS ecosystems against evolving malware threats.

ACKNOWLEDGMENTS

This research is part of the STARLING project funded by FAPESP proc. 21/06995-0. It was partially sponsored by CNPq grant #405940/2022-0 and CAPES grant #88887.954253/2024-00.

REFERENCES

[1] X. Yang, L. Shu, Y. Liu, G. P. Hancke, M. A. Ferrag, and K. Huang, "Physical security and safety of iot equipment: A survey of recent advances and opportunities," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 7, pp. 4319–4330, 2022.

[2] M. Kumar, K. Dubey, and R. Pandey, "Evolution of emerging computing paradigm cloud to fog: applications, limitations and research challenges," in *11th Confluence*. IEEE, 2021, pp. 257–261.

[3] J. Mohanty, S. Mishra, S. Patra, B. Pati, and C. R. Panigrahi, "Iot security, challenges, and solutions: a review," *Progress in Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2019, Volume 2*, pp. 493–504, 2021.

[4] C. H. M. Souza and C. H. Arima, "A hybrid approach for malware detection in sdn-enabled iot scenarios," *Internet Technology Letters*, vol. 7, no. 6, p. e534, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.534

[5] S. Kumar and B. Chandavarkar, "Analysis of mirai malware and its components," in *Machine Learning, Image Processing, Network Security and Data Sciences: Select Proceedings of 3rd International Conference on MIND 2021*. Springer, 2023, pp. 851–861.

[6] T. McIntosh, T. Susnjak, T. Liu, D. Xu, P. Watters, D. Liu, Y. Hao, A. Ng, and M. Halgamuge, "Ransomware reloaded: Re-examining its trend, research and mitigation in the era of data exfiltration," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1–40, 2024.

[7] E. Goncharov, "ICS and OT Threat Predictions for 2024," Kaspersky Lab, Tech. Rep., 2024.

[8] G. E. R. T. (GERT), "Incident response analyst report 2024," Kaspersky Lab, Tech. Rep., 2025. [Online]. Available: https://content.kaspersky-labs.com/fm/site-editor/33/3318ec849851138088d24f26d236f469/source/irreport.pdf

[9] F. Salahdine, T. Han, and N. Zhang, "Security in 5g and beyond recent advances and future challenges," *Security and Privacy*, vol. 6, no. 1, p. e271, 2023.

[10] M. Gopinath and S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Computer Science Review*, vol. 47, p. 100529, 2023.

[11] F. A. Aboaoja, A. Zainal, F. A. Ghaleb, B. A. S. Al-rimy, T. A. E. Eisa, and A. A. H. Elnour, "Malware detection issues, challenges, and future directions: A survey," *Applied Sciences*, vol. 12, no. 17, p. 8482, 2022.

[12] C. H. Souza, T. Pascoal, E. P. Neto, G. B. Sousa, F. S. Filho, D. M. Batista, and F. S. Dantas Silva, "Sdn-based solutions for malware analysis and detection: State-of-the-art, open issues and research challenges," *Journal of Information Security and Applications*, vol. 93, p. 104145, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212625001826

[13] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[14] J. Wu, Y. Wang, M. Sun, X. Xu, and Y. Song, "Towards a framework for developing verified assemblers for the elf format," in *Asian Symposium on Programming Languages and Systems*. Springer, 2023, pp. 205–224.

[15] I. Tahir and S. Qadir, "Machine learning-based detection of iot malware using system call data," in *2024 4th International Conference on Digital Futures and Transformative Technologies (ICoDT2)*, 2024, pp. 1–8.

[16] C.-W. Tien, S.-W. Chen, T. Ban, and S.-Y. Kuo, "Machine learning framework to analyze iot malware using elf and opcode features," *Digital Threats: Research and Practice*, vol. 1, no. 1, pp. 1–19, 2020.

[17] A. Ravi and V. Chaturvedi, "Static malware analysis using elf features for linux based iot devices," in *2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID)*, 2022, pp. 114–119.

[18] J. Zhou, A. H. Gandomi, F. Chen, and A. Holzinger, "Evaluating the quality of machine learning explanations: A survey on methods and metrics," *Electronics*, vol. 10, no. 5, p. 593, 2021.