Towards Effective Machine Learning Models for Ransomware Detection via Low-Level Hardware Information

Chutitep Woralert
Clarkson University
USA
woralec@clarkson.edu

Chen Liu
Clarkson University
USA
cliu@clarkson.edu

Zander Blasingame
Clarkson University
USA
blasinzw@clarkson.edu

Abstract

In recent years, ransomware attacks have grown dramatically. New variants continually emerging make tracking and mitigating these threats increasingly difficult using traditional detection methods. As the landscape of ransomware evolves, there is a growing need for more advanced detection techniques. Neural networks have gained popularity as a method to enhance detection accuracy, by leveraging low-level hardware information such as hardware events as features for identifying ransomware attacks. In this paper, we investigated several state-of-the-art supervised learning models, including XGBoost, LightGBM, MLP, and CNN, which are specifically designed to handle time series data or image-based data for ransomware detection. We compared their detection accuracy, computational efficiency, and resource requirements for classification. Our findings indicate that particularly LightGBM, offer a strong balance of high detection accuracy, fast processing speed, and low memory usage, making them highly effective for ransomware detection tasks.

CCS Concepts

• **Security and privacy** → **Systems security**; **Software security engineering**; **Intrusion detection systems**.

Keywords

Performance Monitoring Counters, Supervised Learning, Ransomware

ACM Reference Format:
Chutitep Woralert, Chen Liu, and Zander Blasingame. 2024. Towards Effective Machine Learning Models for Ransomware Detection via Low-Level Hardware Information. In *International Workshop on Hardware and Architectural Support for Security and Privacy 2024 (HASP '24), November 02, 2024, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3696843.3696847

1 Introduction

One of the most valuable assets of cyberspace is data. Recognizing the importance of data, hackers have developed a type of attack known as "ransomware", which locks users out of their critical data and demands a ransom for its release. The consequences of a ransomware attack can be severe, imposing significant burdens on the victim. This includes system downtime and substantial financial cost from either paying the ransom or human capital spent on system recovery. In recent years, many a new variant of ransomware has emerged with distinct functionality and method of spreading. Some ransomware variants serve dual purposes, such as spying on user activities or exfiltrating data, which can result in the leakage of sensitive information.

While static signature-based tools such as antivirus software can be effective against widely-spread known ransomware attacks, they fall short when it comes to zero-day attacks or unknown variants, due to a lack of matching signatures in the database. In contrast, dynamic analysis and anomaly detection tools offer the potential of detecting zero-day attacks or unknown variants through behavioral analysis. Ransomware typically generates an unusual amount of file system activities such as reading, modifying, and deleting the data to carry out its encryption process. Looking at this behavior, various tools have been developed to detect ransomware, including ShieldFS by Continella et al. [12], Redemtion by Kharraz et al.[23], and SSD-Insider++ by Baek et al. [8]. Additionally, network traces can also be used as another means for ransomware detection, as demonstrated by Almashhadani et al. [5].

Although performance monitoring counters (PMCs) were originally designed to diagnose program performance, they can also capture detailed low-level hardware information that effectively represents a program's behavior. This capability makes PMCs a powerful tool for detecting ransomware attacks, as demonstrated by previous work [4, 7, 17, 28, 29]. By analyzing the data collected from performance counters to identify anomalies, it is possible to detect ransomware attacks occurring within a system.

One way to build such a detection model is through a semi-supervised one-class learning method, which assumes no prior knowledge of the ransomware during the training stage. However, while this method doesn't require pre-existing knowledge of the attack, it can be prone to false predictions. In the ongoing battle against ransomware, every opportunity to improve detection is crucial. An alternative approach is to use a supervised two-class model that leverages existing knowledge of ransomware during training. This method helps to reduce false prediction cases while still detecting previously unseen ransomware attacks, as shown in [29]. Ganfure et al. [17] introduced a new approach that converts the performance counter data into images, which are then analyzed using a convolutional neural network (CNN) for image-based classification to detect ransomware.

In this work, we evaluated various supervised learning models for ransomware detection, utilizing either native time series data or converted image data. In addition to detection accuracy, we also assessed their associated data processing time, prediction time, as well as memory usage. Our results show that neural network models, such as Multilayer Perceptron (MLP) and CNN, can

HASP '24, November 02, 2024, Austin, TX, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1221-0/24/11
https://doi.org/10.1145/3696843.3696847

achieve exceptionally high detection accuracy, over 99.31% on average. The gradient boosting models, such as Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting (LightGBM) can acheive comparable detection accuracy as the neural network models, with the best detection accuracy being 99.97% compared to 99.98% of CNN image classification model. However, the gradient boosting models on average have much faster prediction time, with LightGBM only taking 7ms to perform classification, while the CNN model can take as much as 59ms to classify the same input window. On average the gradient boosting model also requires less memory to operate, with only 413MB of memory required for the LightGBM model while the CNN model can take up to 935MB of memory usage, which demonstrates the efficiency and lightweight nature of the gradient boosting models.

The contribution of this work lies in our investigation of state-of-the-art supervised learning models to identify the best approach in terms of performance and efficiency at detecting ransomware attacks using low-level hardware information. By evaluating various models, we aim to identify the optimal balance between performance and resource efficiency, providing insights into the best methodologies for enhancing ransomware detection capabilities on real-world systems.

The rest of this paper is organized as follows. In Section 2, we provide background information on the ransomware detection techniques and the performance counter. The architecture of the proposed framework is presented in Section 3. The evaluation of the proposed framework is presented in Section 4. Lastly, the conclusions are drawn in Section 5.

2 Background

In this section, we provide an overview of the ransomware detection and mitigation techniques, as well as the background and the use cases of the performance monitoring counter for ransomware detection.

2.1 Ransomware Detection Techniques

Traditionally, ransomware detection has relied on signature-based methods, where the program's signature, derived from its executable or source code, is compared against a database of known malware signatures. If a match is found, the program is flagged as malicious. The antivirus programs such as Norton, McAfee, Windows Defender, and ClamAV are common examples of tools that utilize this technique. However, for these tools to be effective, they must have prior knowledge of the attack signatures. To circumvent this, attackers continually create new ransomware variants with altered signatures, rendering signature-based detection less effective.

To address the limitations of signature-based detection, the defenders have been developing behavioral analysis-based tools. Ransomware attacks typically involve extensive file system operations, such as reading, writing, and deleting files, as the ransomware often deletes the original files to prevent recovery. Advanced ransomware that uses asymmetric encryption usually needs to communicate with a Command-and-Control (C2) server to store the encryption key, which can be detected through network packet tracing tools like Wireshark. These network traces become even more apparent if the ransomware attempts to exfiltrate the user data, generating significant network traffic while the data is being transferred to the attacker. These abnormal behaviors during a ransomware attack can signal anomalies in the system, making behavioral analysis a valuable detection method. However, some ransomware variants can evade detection by employing techniques like adding padding to the attack or mimicking benign programs. This highlights the need for more robust behavioral analysis models to effectively detect and counter these evolving ransomware threats.

2.2 Performance Monitoring Counters

The performance monitoring unit (PMU) can be found in most modern processors that support architectural and micro-architectural hardware event profiling. Performance monitoring counters (PMCs) are specialized registers inside PMU that capture low-level hardware information by monitoring specific hardware events that provide insights into program behavior. The PMU contains configuration registers to set up the parameters, including enabling or disabling the counter, selecting events to monitor, monitoring mode, and privilege level, for the performance counter monitoring.

Accessing and utilizing the performance counter can be complex, but fortunately, several tools have been developed to simplify this process. The tools such as Intel VTune, Xcode Instruments, Perf Linux[14], and K-Leb[27] make PMCs more accessible, broadening their applications and making it easier for users to use the performance counter for tasks such as optimization and anomaly detection.

2.3 Ransomware Detection with Performance Counter

Previous studies have investigated the use of performance counters in detecting ransomware attacks. Alarm et al. [4] demonstrated the effectiveness of using a one-class threshold-based LSTM autoencoder with the performance counter data collected from individual programs to identify ransomware attacks. Woralert et al. [28] conducted an extensive feature selection study specifically for ransomware detection. They also explored system-wide performance monitoring, offering improved scalability and the potential for cross-platform monitoring, rather than limiting the monitoring to individual process. Ganfure et al. [17] introduced a novel approach using convolutional neural networks (CNNs) for binary classification by converting performance counter data into the image domain to represent system behavior, thereby aiding the ransomware detection.

In this paper, we further explore the use of various two-class models with system-wide performance monitoring to detect ransomware attacks. Additionally, we evaluate multiple representative machine learning models for categorical classification, assessing their performance and efficiency in handling both time series and image data.

3 Detection Model Framework

In this section, we present our detection framework which is composed of two primary components: the data collection module and the classification module, as illustrated in Figure 1.

3.1 Data Collection Module

In this work, we employ K-LEB, a performance counter data collection tool developed by Woralert et al. [27] as the foundation for our data collection module.

K-LEB is selected due to its lightweight nature, which is essential for minimizing performance overhead during deployment. Maintaining low overhead is critical because the system must be continuously monitored in real-time with fine-grain granularity to perform real-time analysis of the time series data. Tools that require significant resources, such as Instruments, could introduce substantial overhead, making them less suitable for this monitoring purpose.

To ensure efficient monitoring, K-LEB is configured to collect the system performance counter data at an interval of 10ms. To safeguard the integrity of the performance counter data from potential ransomware corruption or encryption, the data log is secured with root privileges. Additionally, to prevent attackers from accessing the data log, the logs are segmented and periodically transferred to a separate machine, where they are reassembled and stored for further processing and logging. The original data log is then removed from the user's machine.

3.1.1 Time Series Data. The native data collection by K-LEB stores the data in a time series format in a comma-separated values (CSV) file. Each column represents a specific hardware event to be monitored, while each row represents the hardware event counts from the selected performance counters during each time step of the time series, as shown in Figure 2. The data is then split up into a smaller window size, which is used to represent behavior in that time period and used as input for the classification model.

3.1.2 Image Conversion. To perform classification using image classification models, we convert time series data into image snapshot data for each sliding window. In this work, we convert the time series data into grey-scale images that represent the hardware event count during different periods. First, the performance counter value is normalized and scaled to be between 0 and 1. Python Imaging Library (PIL) is then used to convert the scaled value to the greyscale image where the width is the size of a sliding window, and the height is the number of features that will be used for the classification, as shown in Figure 2.

3.2 Classification Module

In this section, we investigate different types of machine learning models that can be used as the classifier for our malware detection scheme. Specifically, we explore the MLP, LSTM, CNN, XGBoost, and LightGBM algorithms. The MLP, LSTM, and CNN models are implemented in TensorFlow [2]. We provide a brief overview of the classification algorithms in this section.

3.2.1 MLP. The Multilayer Perceptron (MLP) is another name for a modern feed-forward artificial neural network which can be trained via back-propagation [6, 9]. The MLP is one of the simplest kinds of neural networks, consists of multiple layers containing multiple "neurons" and a non-linear activation. A particular layer can be simply defined as an affine transformation composed with a component-wise non-linearity, i.e., given an input vector $x \in \mathbb{R}^n$, a weights matrix $\theta \in \mathbb{R}^{m \times n}$, a bias vector $b \in \mathbb{R}^m$, and component-wise non-linear function $f : \mathbb{R}^m \to \mathbb{R}^m$ the layer can be defined as

$$y = f(x\theta^\top + b) \tag{1}$$

with output vector $y \in \mathbb{R}^m$. The learnable parameters, $\{\theta, b\}$, are then learned using back-propagation in combination with an update algorithm like stochastic gradient descent (SGD) [10]. In practice, we train the neural network with a momentum-based Adam algorithm which has stronger convergence guarantees [24]. Despite its simplicity, the MLP functions as a universal approximator assuming arbitrary width or depth [21]. To train the MLP for classification, we use the softmax function on the last layer to scale the outputs into class probabilities. The softmax function $\sigma : \mathbb{R}^K \to [0, 1]^K$ is defined as

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2}$$

where $\mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$ is the input vector. In the case of binary classification, we have $K = 2$ and $\sigma(\mathbf{z})_1$ denoting the probability of benign behavior and $\sigma(\mathbf{z})_2$ denoting the probability of malicious behavior. Using this output we minimize the cross entropy between the parameterized distribution, $\mathbb{P}_\theta$ and the data distribution $\mathbb{P}_{data}$ with the cross entropy defined as

$$H(\mathbb{P}_{data}, \mathbb{P}_\theta) = -\mathbb{E}_{\mathbb{P}_{data}}[\log \mathbb{P}_\theta] \tag{3}$$

By minimizing the cross entropy, the MLP learns to model the probability of a given sample belonging to the benign or malicious class. The MLP models are trained with hidden layers of size 100 and with the ReLU [15] activation function.

3.2.2 LSTM. The Long Short-Term Memory (LSTM) architecture [20] is used as part of the classification pipeline. Recurrent Neural Network (RNN) is a type of neural network designed to handle time series data; however, they have the problem of vanishing gradients as the time series become longer [20]. To remedy this, the LSTM architecture introduces two states and three gates, see Equations (4) to (8) where σ denotes the sigmoid activation function, $W_{\cdot\cdot}$ denotes weight tensors, $b_{\cdot}$ denotes bias tensors, and $\odot$ is the Hadamard product.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{4}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{5}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{6}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{7}$$
$$h_t = o_t \odot \tanh(c_t) \tag{8}$$

The initial states are given as $c_0 = 0$ and $h_0 = 0$. The hidden state, h_t, is used to store encoded information about the previous time step; conversely, the cell state, c_t, acts as a form of global memory of the LSTM network over all time steps. Every gate operates on the current data, x_t, and hidden state from the previous time step, h_{t-1}. The input gate, i_t, is used to determine how much of the current data is to be remembered in the global memory, c_t. Likewise, the forget gate, f_t, is used to determine how much the global memory should remember past events. Finally, the output gate, o_t, determines how strongly the output signal, $\tanh(c_t)$, should be passed to the next hidden state and output, h_t.

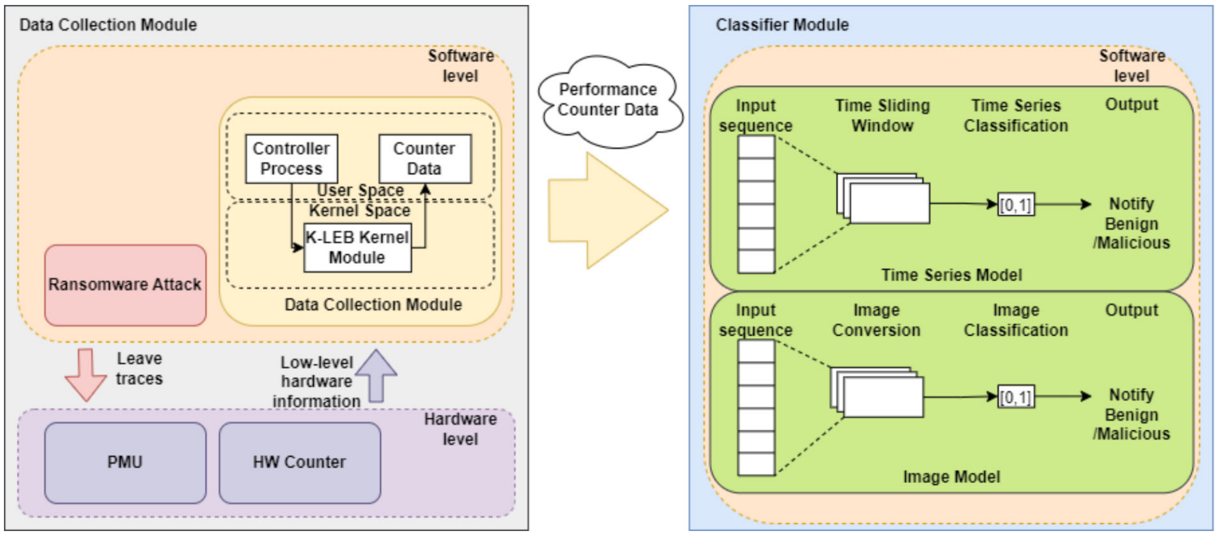


Figure 1: Proposed Detection Framework

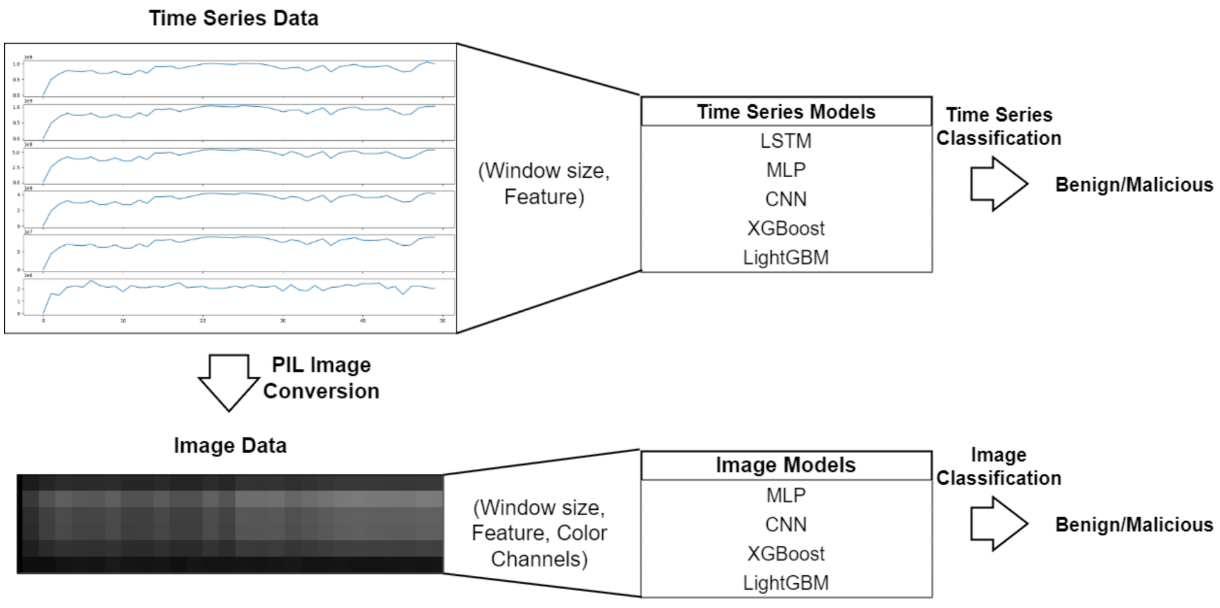


Figure 2: Time Series to Image Data Conversion

The LSTM model is used in a two-class scenario where, instead of training the LSTM to preform time series forecasting, the model is trained to predict the class of the given time series data. The LSTM backbone for the two-class approach consists of one LSTM layer of 128 units followed by a dropout layer and two dense fully-connected layers. After the output of the LSTM model, a two-layer dense feed-forward neural network is attached on top with the first layer using the ReLU activation function, and a softmax activation function on the last layer to scale the output to class probabilities. Like with the MLP model, the network is then trained to minimize the cross entropy.

3.2.3 CNN. Convolutional Neural Networks (CNNs) are a popular class of neural networks that has achieved a high degree of success in computer vision and imagining applications [16, 25]. The key strength of CNNs over a more simple MLP is that CNNs restrict the information flow between the layers by consider on "local" information, a restriction which is quite sensible for computer vision type tasks, allowing for smaller parameter counts and more efficient information routing in the network. A convolutional layer works by convolving the input tensor with a learnable kernel to create the output. Given an input tensor $x \in \mathbb{R}^{c \times h \times w}$ with c denoting the number of channels and h and w denoting the height and width of the spatial dimensions, a convolutional kernel $w \in \mathbb{R}^{c \times k \times k}$ with kernel size k, and bias $b \in \mathbb{R}^{c \times h \times w}$, the output $y \in \mathbb{R}^{c \times h \times w}$ is defined as

$$y_{i,j} = b_{i,j} + \sum_{h=1}^{c} \sum_{a=-k}^{k} \sum_{b=-k}^{k} x_{c,i-a,j-b} w_{c,a,b} \tag{9}$$

To decrease the spatial dimension between convolution layers, a convolutional stride or pooling layer can be applied. The output of the last layer of the CNN is flattened and fed into a feed-forward neural network with a softmax activation function. Like the MLP and LSTM networks, the model is trained to minimize the cross entropy. In this work the CNN time series models are trained with a kernel size of 3 with a channel size of 100 using the ReLU activation function. The image classification models follow the same hyperparameters as its time series counterpart.

3.2.4 Gradient Boosting. Gradient boosting is a popular kind of boosting algorithm that combines several weak learners, often decision trees, into strong learners where each new model aims to minimize a loss function of the previous model using a gradient descent algorithm [26]. This is accomplished by training a new weak model to minimize the gradient of the loss function with respect to the prediction of the current ensemble. The predictions of newly trained weak model are then added to the ensemble and the process is repeated until the stopping criterion is met. Gradient boosting has the advantage of being much lighter on system resources than neural network based models, as decision trees are less resource intensive. Moreover, recent work from Grinsztajn et al. [19] showed that tree-based models outperform deep learning based models on problems with tabular data. Therefore, these kinds of models should achieve state-of-the-art performance with minimal computational overhead compared to the deep learning based approaches enumerated above. In this work, we use two widely used gradient boosting libraries: eXtreme Gradient Boosting (XGBoost) [11] and Light Gradient-Boosting Machine (LightGBM) [22]. All the gradient boosting models in this work are trained with a max depth of 10 and a max leaves number of 256.

3.3 Feature Selection

Each hardware event reflects a specific behavior that occurs during a program's runtime. The AMD processor we used in this study, over 60 hardware events are available for selection. However, only six programmable performance counters can be utilized to monitor these hardware events concurrently, significantly limiting the number of events that can be tracked. Therefore, it is crucial to select the most relevant hardware events as features for the classification model.

In this work, we followed the study made by Woralert et al. [28], selecting a set of six events with the highest relevance ROC-AUC scores: BR_RET, INST_RET, DCACHE_ACCESS, LOAD, STORE, and MISS_LLC. These events were chosen based on their ability to capture behaviors indicative of various ransomware attacks, which manifest as deviations from the baseline behavior of benign programs, thereby signaling potential anomalies.

4 Experiment Analysis

To evaluate the effectiveness of the supervised learning two-class model, we deployed the data collection module on the user system and conducted both benign and malicious tasks on the user system. The performance counter data generated during these tasks were logged and transferred to a separate machine where the classifier module was running to perform classification, distinguishing between benign and malicious activities.

4.1 Experiment Setup

The experiments were performed on an AMD Ryzen 7 5800X @ 3.8GHz 8-Core Processor running Ubuntu 20.04 as the user machines, and an Intel Xeon Silver 4114 processor @ 2.20GHz running Ubuntu 20.04 as the classifier machine.

For the experiment, the K-LEB data collection module was configured to monitor six specific hardware events, which were sampled at a 10ms interval, as detailed in Section 3. The data collected from these events formed long time series data, encompassing both benign and malicious scenarios.

In the benign scenario, various workloads were executed on the user system, such as coding, file read/write operations, copying/moving files, web browsing, video streaming, stress tests, benign encryption programs, and other standard Linux commands. This mixture of benign workloads was used to train and test the classifier models. For the ransomware scenario, ransomware samples were downloaded from the malware database MalwareBazaar [1], providing real malware executables for research purposes. The primary criterion for selecting ransomware was its ability to execute and carry out its malicious encryption task. The ransomware families used in this study included *Alphv*, *Blackcat*, *Golang*, *Hellokitty*, *Holycrypt*, *Monti*, *Ransomexx*, *Revil*, *Sodinokibi*, and *Tellyouthepass*. Additionally, open-source ransomware that are available on GitHub, such as *C Ransomware* [3], and *Cryptsky* [13] was used as well. The user machine was populated with 65GB of various user files including videos, photos, source code, and documents. Each ransomware sample was individually downloaded and executed on the user system to collect performance counter data during the attack. This data was then used for training and testing the two-class classification models.

After each ransomware execution, the system was restored to a previous state using Timeshift [18]. Timeshift is a backup utility that can be used to take a snapshot of the system for backup. Timeshift was employed to restore the file system and remove ransomware infections, ensuring a clean environment before executing the next ransomware sample.

4.2 Model Detection Performance

While the LSTM model has demonstrated high effectiveness in analyzing time series data with relatively high precision, it is known to demand significant computational resources in order to perform the classification. In this section, we explore alternative classification models that can handle time series data for two-class classifications, such as XGBoost, LightGBM, MLP, and CNN models. Furthermore, we also investigate the application of image classification models as well.

Table 1: Detection Accuracy Comparison across Supervised Learning Models

Model/Dataset	Time Series Data				Image Data			
Window size 50	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
LSTM	97.05	98.77	95.27	96.99	N/A	N/A	N/A	N/A
XGBoost	99.81	99.93	99.70	99.81	99.73	99.93	99.53	99.73
LightGBM	99.77	99.89	99.65	99.77	99.78	99.95	99.61	99.78
MLP	98.41	99.07	99.07	98.73	99.40	99.31	99.49	99.40
CNN	97.94	97.43	98.56	97.97	99.91	99.93	99.89	99.91
Window size 100	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
LSTM	97.82	99.07	96.55	97.79	N/A	N/A	N/A	N/A
XGBoost	99.82	99.94	99.70	99.82	99.80	99.95	99.66	99.80
LightGBM	99.85	99.97	99.73	99.85	99.81	99.94	99.68	99.81
MLP	99.02	98.97	99.08	99.02	99.70	99.70	99.71	99.70
CNN	98.30	98.57	98.04	98.27	99.88	99.89	99.86	99.88
Window size 500	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
LSTM	98.03	99.30	96.75	98.00	N/A	N/A	N/A	N/A
XGBoost	99.93	99.97	99.89	99.93	99.91	99.98	99.83	99.91
LightGBM	99.93	99.97	99.89	99.93	99.91	99.99	99.84	99.91
MLP	98.95	99.35	98.54	98.92	99.92	99.96	99.87	99.92
CNN	99.15	98.81	99.50	99.15	99.94	99.99	99.89	99.94
Window size 1000	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
LSTM	98.50	99.65	97.30	98.46	N/A	N/A	N/A	N/A
XGBoost	99.96	99.99	99.93	99.96	99.95	99.99	99.91	99.95
LightGBM	99.97	100	99.95	99.97	99.95	99.99	99.91	99.95
MLP	99.24	99.29	99.20	99.23	99.92	99.95	99.90	99.92
CNN	99.24	99.02	99.47	99.25	99.98	99.99	99.96	99.98

Table 1 shows the result of each two-class model detection accuracy, precision, recall, and f1 score with different window sizes of 50, 100, 500, and 1000 for the prediction period. The data presented here are from an average of 10-fold cross-validation. The table presents the detection accuracy for both models that use Time Series (TS) data and the models that use Image (IMG) data.

The gradient boosting model XGBoost achieves a very high detection accuracy, over 99.87% on average across all window sizes with the highest detection accuracy being 99.96% at the window size of 1000 for the time series data, followed closely by the detection accuracy of 99.95% with the image data of the same window size. Similarly, LightGBM models also achieve exceptional detection accuracy, while being a lightweight model that uses a similar gradient boosting method to perform its classification. On average, the LightGBM models achieve an accuracy over 99.87% across all window sizes for both time series and image datasets. The LightGBM model achieves the highest detection accuracy across all models, being 99.97% on the time series data at the window size of 1000.

The neural network models are shown to have good detection accuracy overall on the image data. Both MLP and CNN models achieve very high detection accuracy across all window sizes with over 99.83% detection rate on average for the image dataset, the highest detection accuracy being 99.92% and 99.98%, respectively, at the window size of 1000. The neural network models are shown to have a slightly lower detection accuracy when dealing with the time series data, with an average detection accuracy of 98.78% .

Overall, the model shows an increase in accuracy relative to the window size input. The models with a bigger window size are shown to have better accuracy than the same type of models with a smaller window size. This is due to the fact that the larger window size can paint a clearer behavior that may be displayed over a longer time period. The recurrent neural network model such as the LSTM model requires long time series data to achieve comparable detection accuracy to other models and hence has a drop in accuracy with the smaller window size.

4.3 Deployment Resource Requirements

Even though the bigger window size input can help increase the accuracy of the model, it also requires more computational resources and takes longer to perform the classification. Table 2 shows the deployment resource required to perform classification in terms of data processing time, time to predict the input window, and memory usage during the deployment. The data presented in the table are collected during the deployment and averaged over 100 iterations.

While the CNN models have been shown to have very high accuracy, they also require more resources than other neural network and gradient-boosting models on average in both predicting time and memory usage. In fact, the CNN image classification model requires the most memory for the deployment as shown in Figure 3. The LSTM models have been shown to have relatively longer prediction time when compared to its peer supervised models, ranging

Table 2: Resources Requirement for Deployment.

Model	Process & Predict (s)	Memory (MB)
Window size 50		
LSTM(TS)	0.013 & 0.089	469
XGBoost(TS)	0.013 & 0.014	420
XGBoost(IMG)	0.052 & 0.019	445
LGBM(TS)	0.016 & 0.014	401
LGBM(IMG)	0.054 & 0.029	426
MLP(TS)	0.011 & 0.054	454
MLP(IMG)	0.056 & 0.055	459
CNN(TS)	0.012 & 0.056	461
CNN(IMG)	0.055 & 0.056	489
Window size 100		
LSTM(TS)	0.013 & 0.119	496
XGBoost(TS)	0.015 & 0.013	420
XGBoost(IMG)	0.068 & 0.023	445
LGBM(TS)	0.017 & 0.010	400
LGBM(IMG)	0.067 & 0.030	428
MLP(TS)	0.013 & 0.055	460
MLP(IMG)	0.068 & 0.051	468
CNN(TS)	0.014 & 0.057	470
CNN(IMG)	0.072 & 0.054	544
Window size 500		
LSTM(TS)	0.031 & 0.335	700
XGBoost(TS)	0.029 & 0.011	424
XGBoost(IMG)	0.176 & 0.026	464
LGBM(TS)	0.034 & 0.007	406
LGBM(IMG)	0.151 & 0.026	438
MLP(TS)	0.030 & 0.058	502
MLP(IMG)	0.182 & 0.047	523
CNN(TS)	0.030 & 0.062	556
CNN(IMG)	0.189 & 0.051	834
Window size 1000		
LSTM(TS)	0.050 & 0.607	828
XGBoost(TS)	0.048 & 0.015	434
XGBoost(IMG)	0.305 & 0.053	452
LGBM(TS)	0.051 & 0.007	413
LGBM(IMG)	0.341 & 0.038	452
MLP(TS)	0.050 & 0.061	536
MLP(IMG)	0.317 & 0.052	572
CNN(TS)	0.050 & 0.062	629
CNN(IMG)	0.312 & 0.059	935

from 89ms to 607ms. In terms of memory usage, it is only second to the CNN image classification models. The LightGBM models have the fastest prediction time with the least memory usage on average, which means they can be used to perform classification in a low computation power machine. XGBoost is also computationally lightweight with comparable memory usage to the LightGBM models, due to its similarity in using a simple gradient boosting tree method. However, the LightGBM still has a faster prediction time and uses less memory to perform the classification on average. The gradient boosting models are shown to be more efficient over the neural network models in both prediction time and memory usage during the deployment.

While the detection accuracy is important, the resource requirement for the model should also be considered. While the CNN models have very high overall detection accuracy, it is a computationally expensive model to be deployed, especially on the machine with less computational resources available. The MLP models are shown to have similar detection performance when compared to the CNN models, but they require less memory usage during the deployment, which might be a better model choice if the data does not require the high complexity of the CNN model to be effectively classified.

The gradient boosting models are shown to achieve very high detection accuracy, while still being more lightweight when compared to the neural network models. In particular, LightGBM models have achieved the highest detection accuracy overall, while being the fastest to perform the classification and at the same time require the smallest memory usage during runtime. This result shows the effectiveness of the gradient boosting models for classification problems when compared to more complex neural network models.

4.4 Time series vs Image Data Classification

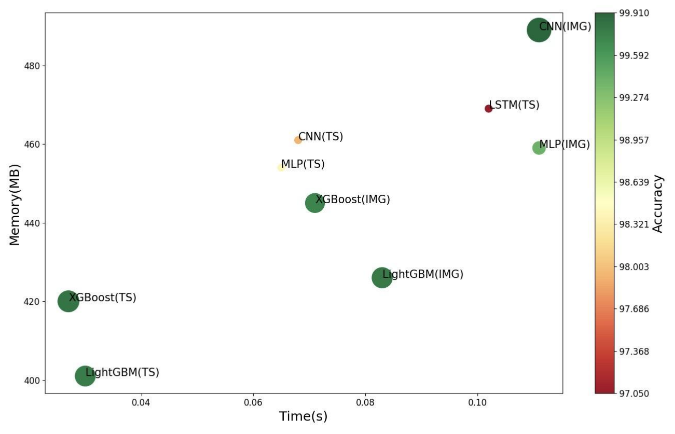
From Table 1, we can see while the image classification models show promising results with a small detection accuracy advantage over the time series models, the image classification comes with the additional overhead of the time required to process data conversion from time series data to the image data. As shown in Table 2, the image classification models require data pre-processing time more than 3X the time series data pre-processing time on average. The image models also require more memory usage during the deployment when compared to the time series models. At the same time, the memory required to store and process the data for the image classification is also bigger than the native time series data format.

The gradient boosting models perform better in terms of overall detection accuracy with the native time series data, at the same time require much less prediction time, less than three times of the image classification model on average with less memory usage during deployment.

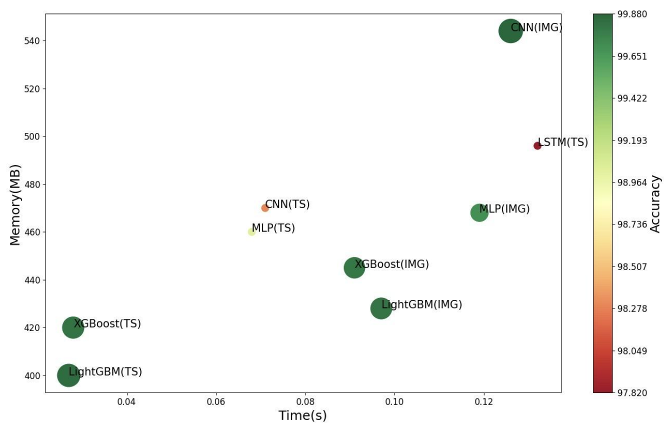
Depending on the type of the application and deployment scenario, it might not be necessary to convert the native time series data into image for classification, as it adds extra overhead with minor improvement for the neural network models. Compared with the gradient boosting models, the image classification only incur extra overheads with no additional benefit.

5 Conclusions

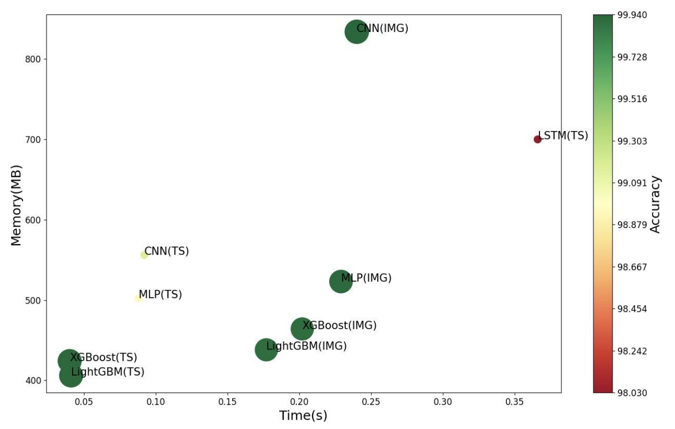
In this paper, we explored several state-of-the-art time series classification models, including XGBoost, LightGBM, MLP, and CNN. We compared their detection performance and the deployment cost of each model for real-time ransomware detection. We also examined the benefits and trade-offs of converting time series data into image domain for image classification models. While this conversion may



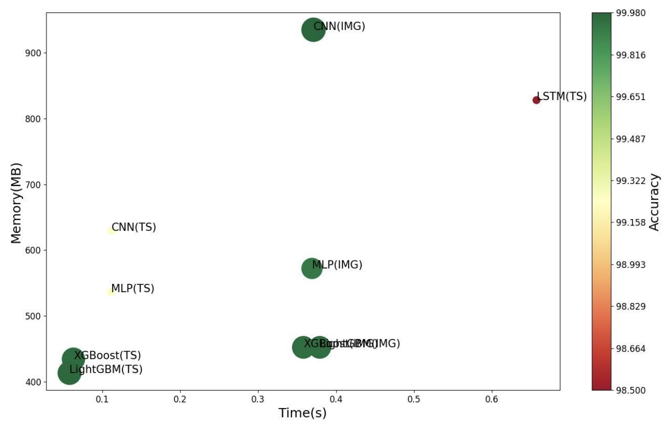
(a) Window Size of 50



(b) Window Size of 100



(c) Window Size of 500



(d) Window Size of 1000

Figure 3: Performance vs Efficiency Comparison across Different Models

offer a slight increase in detection accuracy for the neural network-based models, it also requires significantly more processing time and computational resources for both prediction and deployment.

Overall, our findings indicate that gradient boosting models using native time series data is a promising approach. The LightGBM model, in particular, exhibited outstanding performance, achieving the highest detection accuracy at 99.97% with the fastest prediction time of just 7ms on average, outperforming other models in terms of both speed and resource efficiency. This demonstrates its effectiveness in detecting ransomware attacks while maintaining a lightweight footprint.

For future work, we aim to explore the use of lightweight models to expand the scope of our detection framework, extending its capabilities to cover a broader range of malware and other cyberattacks to further enhancing its versatility and applicability in real-world scenarios.

References

[1] 2022. MalwareBazaar. https://bazaar.abuse.ch [Online]. Available: https://bazaar.abuse.ch.

[2] 2022. [Online]. Tensorflow. https://www.tensorflow.org Available: https://www.tensorflow.org.

[3] Daniele Affinita. 2020. Ransomware written in C. https://github.com/DaniAffCH/Ransomware [Online]. Available: https://github.com/DaniAffCH/Ransomware.

[4] Manaar Alam, Sayan Sinha, Sarani Bhattacharya, Swastika Dutta, Debdeep Mukhopadhyay, and Anupam Chattopadhyay. 2020. RAPPER: Ransomware Prevention via Performance Counters. arXiv:2004.01712 [cs.CR]

[5] Ahmad O. Almashhadani, Mustafa Kaiiali, Sakir Sezer, and Philip O'Kane. 2019. A Multi-Classifier Network-Based Crypto Ransomware Detection System: A Case Study of Locky Ransomware. *IEEE Access* 7 (2019), 47053. https://doi.org/10.1109/ACCESS.2019.2907485

[6] Shunichi Amari. 1967. A Theory of Adaptive Pattern Classifiers. *IEEE Transactions on Electronic Computers* EC-16, 3 (1967), 299–307. https://doi.org/10.1109/PGEC.1967.264666

[7] P. Mohan Anand, P. V. Sai Charan, and Sandeep K. Shukla. 2023. HiPeR - Early Detection of a Ransomware Attack using Hardware Performance Counters. *Digital Threats* 4, 3, Article 43 (oct 2023), 24 pages. https://doi.org/10.1145/3608484

[8] Sungha Baek, Youngdon Jung, David Mohaisen, Sungjin Lee, and DaeHun Nyang. 2021. SSD-Assisted Ransomware Detection and Data Recovery Techniques. *IEEE Trans. Comput.* 70, 10 (2021), 1762–1776. https://doi.org/10.1109/TC.2020.3011214

[9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.* 3, null (mar 2003), 1137–1155.

[10] J. Bilmes, K. Asanovic, Chee-Whye Chin, and J. Demmel. [n. d.]. Using PHiPAC to speed error back-propagation learning. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 5. 4153–4156 vol.5. https://doi.

org/10.1109/ICASSP.1997.604861

[11] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[12] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. 2016. ShieldFS: A Self-Healing, Ransomware-Aware Filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (Los Angeles, California, USA) *(ACSAC '16)*. 336–347. https://doi.org/10.1145/2991079.2991110

[13] Skyler Curtis. 2017. CryptSky. https://github.com/deadPix3l/CryptSky [Online]. Available: https://github.com/deadPix3l/CryptSky.

[14] Arnaldo Carvalho De Melo. 2010. The new linux'perf'tools. In *Slides from Linux Kongress*, Vol. 18. 1–42.

[15] Kunihiko Fukushima. 1969. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. *IEEE Transactions on Systems Science and Cybernetics* 5, 4 (1969), 322–333. https://doi.org/10.1109/TSSC.1969.300225

[16] Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* 36 (1980), 193–202. https://api.semanticscholar.org/CorpusID:206775608

[17] Gaddisa Olani Ganfure, Chun-Feng Wu, Yuan-Hao Chang, and Wei-Kuan Shih. 2023. DeepWare: Imaging Performance Counters With Deep Learning to Detect Ransomware. *IEEE Trans. Comput.* 72, 3 (2023). https://doi.org/10.1109/TC.2022.3173149

[18] Tony George. 2007. Timeshift. https://github.com/teejee2008/timeshift [Online]. Available: https://github.com/teejee2008/timeshift.

[19] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2024. Why do tree-based models still outperform deep learning on typical tabular data?. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (, New Orleans, LA, USA,) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 37, 14 pages.

[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9 (1997), 1735–1780.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

[22] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

[23] Amin Kharraz and Engin Kirda. 2017. Redemption: Real-Time Protection Against Ransomware at End-Hosts. In *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, Cham, 98–119.

[24] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[26] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. 1999. Boosting algorithms as gradient descent. In *Proceedings of the 12th International Conference on Neural Information Processing Systems* (Denver, CO) *(NIPS'99)*. MIT Press, Cambridge, MA, USA, 512–518.

[27] Chutitep Woralert, James Bruska, Chen Liu, and Lok Yan. 2020. High Frequency Performance Monitoring via Architectural Event Measurement. In *IEEE International Symposium on Workload Characterization (IISWC)*. 114–122. https://doi.org/10.1109/IISWC50251.2020.00020

[28] Chutitep Woralert, Chen Liu, and Zander Blasingame. 2023. HARD-Lite: A Lightweight Hardware Anomaly Realtime Detection Framework Targeting Ransomware. *IEEE Transactions on Circuits and Systems* (2023), 1–12. https://doi.org/10.1109/TCSI.2023.3299532

[29] Chutitep Woralert, Chen Liu, Zander Blasingame, and Zhiliu Yang. 2023. A Comparison of One-class and Two-class Models for Ransomware Detection via Low-level Hardware Information. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 1–6. https://doi.org/10.1109/AsianHOST59942.2023.10409333