Detecting ransomware using system calls through transfer learning on a limited feature set

Harpreet Kaur, Vimal Kumar✉, and Atthapan Daramas

The University of Waikato,
Hamilton, New Zealand
{harpreet.kaur,vimal.kumar}@waikato.ac.nz, ad134@students.waikato.ac.nz

Abstract. System and API call based ransomware detection methods have gained popularity recently since they are comparatively robust in the face of obfuscation attempts. In this paper, we present a system call based ransomware detection technique that utilises ResNet-50 image classifier model. Compared to previous work, we forego complex preprocessing and focus on five specific system calls. The system/API call datasets needed to train such models are scarce as they rely on the availability of active ransomware samples. Given the lack of publicly available ransomware samples for training and evaluating models, we employ transfer learning techniques on the ResNet-50 model. Our results show that transfer learning on models such as ResNet-50 can provide acceptable ransomware detection accuracy even when used with small datasets and a limited set of system calls with little pre-processing thus highlighting the importance of optimising or selecting the best feature sets in image-based detection to get better results and saving computational cost and detection time.

Keywords: Ransomware Detection · Transfer Learning · System Calls · ResNet-50 · Android

1 Introduction

Ransomware is currently one of the biggest cyber threats in the world. Ransomware's incidence rate has increased by 13% over the past five years, resulting in an average financial impact of $1.85 million per incident [1]. Cybersecurity Ventures [2] predicts that with a new ransomware attack occurring every two seconds, the cost to victims could reach approximately $265 billion annually by 2031. To develop solutions for protection against ransomware, dynamic analysis of these applications using artificial intelligence and learning-based approaches has been researched extensively, primarily using system and API call logs as described in [3–6]. These works employ resource and time-intensive feature engineering techniques to extract important features from system and API call logs which are then used for ransomware detection.

In practice, cyber-defense is implemented in the form of multi-layered solutions. However, if every layer in a multi-layered architecture relies on complex

pre-processing and has to process a long list of features, it can introduce delays and increase resource demands in reaching a detection decision, which can be catastrophic, in particular when defending against ransomware in a resource-constraint environment, such as mobile devices. On the other hand, utilisation of complex pre-processing and large feature sets, as has been done extensively in literature [3–8] demonstrates better outcomes such as accuracy, precision and recall. A significant challenge in training learning-based models is the lack of publicly available ransomware-related datasets. Although generic malware samples are available in abundance on platforms such as `MalwareBazaar` and `VirusShare`, few of them are active ransomware, resulting in a limited number being available for training and testing the models.

This paper presents a lightweight technique for detecting ransomware using a minimal feature set. Our methodology offers a practical solution in scenarios where data and resource constraints limit model training and testing capabilities, serving as a viable alternative in comparison to complex ransomware detection approaches. To address the data-related challenges, our work employs a transfer learning approach. Our evaluations show that lightweight techniques based on transfer learning can achieve acceptable metrics. In particular, we show that our lightweight technique only incurs an accuracy penalty of 1-4% compared to complex state-of-the-art techniques. Below is the approach we followed and our contributions in this work:

1. Leveraged transfer learning (TL) to address the issue of limited ransomware related dataset availability for training the model.
2. Utilized only five specific system calls, `openat`, `read`, `write`, `unlinkat` and `renameat` to train our model.
3. Created synthetic ransomware test set mirroring the distribution of real test samples for robust evaluation of our model's performance.
4. Evaluated the performance of our image-based ransomware detection approach on the dataset we collected and the dataset we generated in addition to the dataset provided by [9].

The rest of the paper is structured as follows: Section 2 reviews other works in the related area. Section 3 describes our overall methodology, including the description of our dataset. Section 4 discuss the results, followed by the conclusion of the paper in Section 5.

2 Related work

Ransomware is a type of malware that is designed to restrict access to a system or data until the attacker's demanded ransom is paid [10]. It has become the preferred choice of malware for financially motivated cybercriminals. Techniques used to analyse ransomware and malware in general can mainly be categorised into static, dynamic and hybrid approaches. Static analysis refers to the analysis of the code of the given application without executing it to determine whether it is malicious or benign. These techniques are generally cheap and fast; however

their effectiveness is limited when malware implements obfuscation techniques. In dynamic analysis on the other hand applications are executed and observed within a controlled environment to determine whether they are malicious or not. Dynamic analysis techniques can comparatively be resource and time intensive [11]. The two approaches can also be combined to create a hybrid malware analysis approach. In this work, we conducted dynamic analysis to acquire behavioral features of ransomware samples through system calls.

Malware analysis based on various Machine Learning (ML) techniques has been extensively studied in the literature. To efficiently represent malware features for training ML models, two types of techniques are mainly used: n-gram representation and image based representation [12]. In [4], n-gram feature sets of lengths 2, 3, 4, and 5 are generated from sequences of system call logs to evaluate the performance of various models including k-nearest neighbors(KNN), Logistic Regression (LR), Random Forest (RF), Decision Tree (DT) and Support Vector Machine (SVM). The authors noted that the volume of n-gram frequencies was quite large which affects processing and training time and memory usage. Since not all of these features contribute to the classification of malware samples, they employed the enhanced maximum relevance and minimum redundancy (EmRmR) feature selection method to remove noisy features and retain only those with high correlation to the target class. They then selected top-k (30,60,90,120) features to run the experiments. The top 90 feature set performed well using the SVM model, providing an F1-score of 98.6%. However, the authors acknowledged that the process of selecting appropriate features was intensive and time-consuming. In our review of the existing literature, we focus on the F1-score as it offers a balanced evaluation of the model's performance, taking into account both precision and recall. The work by [5] also analyzed dynamic system call behaviour for malware classification and applied feature extraction techniques using chi-square score method to select the top 30 features with the highest chi-square score. They evaluated their feature set on ensemble and individual machine learning techniques, among which stacking showed the best results, achieving an F1-score of 96.90% for text messaging based malware.

Nataraj et al. [13] show that the image representation of the malware feature set is resilient to obfuscation. They observed that digital images of malware binaries belonging to one family appear similar and distinct from those of other malware families. They achieved an accuracy of 98% by employing Gabor filters for feature extraction and k-nearest neighbors for classification on 9,458 standard images from 25 malware families. Their classification method reads a malware binary as a vector of 8-bit unsigned integers to convert it to a 2D grayscale image. This technique, however, is equivalent to a static malware analysis approach and allows an attacker to create countermeasures for evasion. To address this, [3] proposed a method to extract sequences of a malware's API calls to create a visual fingerprint for each malware using API categorization and colour mapping. The dataset for this process consisted of 9 malware families with 1000 variants in each family. Their feature set, consisting of 16 API categories, achieved an F1 score of 98.12% for the ransomware family. More recently, [6] employed sys-

tem call grouping on the AWSCTD dataset [14] for malware classification using Support Vector Machines (SVM) and Decision Tree (DT) models. Their method achieved an F1 score of 98% on SVM and 97% on DT.

The models in the above approaches are trained from scratch and, therefore, learn to extract relevant features tailored to the selected malware types from extensive training data. Such models, however, suffer greatly if the training data is limited, which is often the case when dealing with cybersecurity problems. One such problem is ransomware detection on Android operating system where there is a lack of publicly available samples. In such scenarios, Transfer Learning (TL), where models already trained on large datasets, such as ImageNet [15], are fine-tuned for classification, can be useful. For instance, [7] employed the Bi-model technique for malware classification on image representations of Malimg malware data using transfer learning. To develop a Bi-model, the authors combined pre-trained models VGG-16 and ResNet-50 to extract hybrid feature sets from the image data, which were then used with machine learning algorithms including SVM, RF, KNN, and LR for classification. The authors noted that this method incurs high computational cost in training. Similarly, [8] employed transfer learning techniques to train an ensemble deep learning model combining VGG-16 and ResNet-50 on the images generated using semantic features of portable executable headers of ransomware files. The authors used machine learning models to select the most suitable features to encode into ransomware images. With the 10 best features, this approach attained an F1 score of 97.4%.

In this work, we build a ransomware detection method using transfer learning utilising only five specific system calls. Our results demonstrate that despite using long list of features, complex preprocessing, feature engineering techniques, and the use of ensemble model in the works mentioned in [3–8], the F1-score only improves by approximately 1-4% compared to our simple approach that utilizes behavioural patterns observed in only five specific system calls. This highlights the significance of directing research efforts towards optimizing the best feature set for training the ML models with image-based data for ransomware classification.

3 Methodology

We use the Android Operating System's system call logs of applications in our work. The usage of system call logs is inspired by the work of Chew et al. [9, 16] in which the sequence of system calls is used for ransomware classification via finite-state machines. The authors highlighted that the system calls can infer applications' behavioural patterns that are related to ransomware behaviour. For instance, multiple iterations of specific block sizes of read, followed by write system calls can be a result of files being encrypted by an encryption-type ransomware [9]. While Chew et al. have incorporated multiple system calls, only five system calls are used in this work, namely, `openat`, `read`, `write`, `renameat` and `unlinkat`.

In this section, we describe our process to translate these manually observed patterns in system call sequences to visual imagery to evaluate the effectiveness of these patterns in image based ransomware classification through transfer learning.

Architecture : Our overall architecture consists of four main stages: System call filtration, preprocessing and image generation, classification, and evaluation. Firstly, System call filtration filters out the system calls that are not used in this work. Next, preprocessing and image generation translate the sequence of interested system calls into imagery. The images are passed into the classification process, where transfer learning is used on a modified deep convolutional neural network to use computer vision techniques for malware classification. Finally, the evaluation component interpret the predictions made by the model. Fig. 1 provides an overview of our architecture.

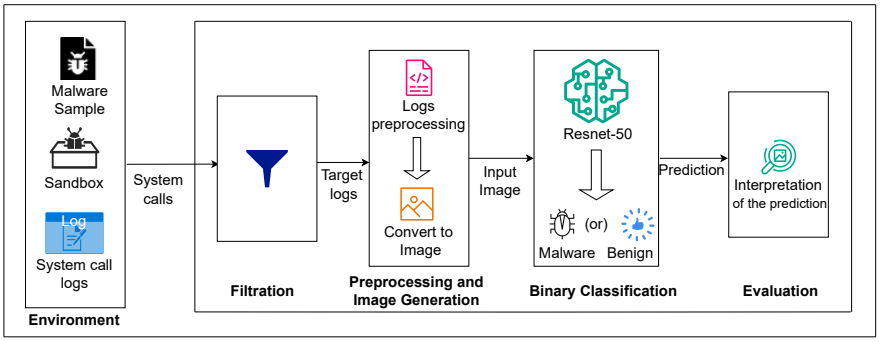


Fig. 1: Pipeline for malware classification using transfer learning

Dataset - Ransomware & Benign Samples : We collected 332 Android ransomware samples from `MalwareBazaar` and `VirusShare` as very few of the available samples are infact tagged as Android ransomware. Each ransomware was run in Android Studio for 5 minutes to capture system call logs using Strace. Different samples may require different actions to be taken to get activated. In general, these actions were: enter a number, press a button, call the device, deliver a message, and send a text file. We kill the malware process after 5 minutes to have a reasonable log size for observation. We collected benign samples from `APKMirror` and `APKPure` platforms. Along with our collected samples, we are also using the system call dataset provided by [9] to compare the performance of our technique on two datasets. Table 1 provides the total number of samples for both datasets.

System call extraction : Android uses the Linux system calls API which has over 300 different system calls. Therefore the system call log contains a large amount of information and system calls that aren't necessary for malware anal-

Table 1: Dataset

Dataset	Classes	Total Samples
Chew et. al. [9]	Benign	225
	Malware	213
This work	Benign	336
	Malware	332

ysis complicating the analysis process. To remove this unnecessary information and to provide the model with a limited feature set, we focused on just five system calls. The five system calls were `openat`, `read`, `write`, `renameat` and `unlinkat` as demonstrated by [9].

Base Model for Transfer Learning : We used ResNet-50 [17] as our base model. To enable transfer learning, we remove the top layer and replace it with two dense layers. The first replaced layer is a fully connected layer with 1024 nodes. It uses a ReLU activation function, which helps the model to map and learn patterns specific to our input of malware and benign imagery [18]. The second layer employs a sigmoid activation function to generate probabilities that show the likelihood of the input belonging to a particular class. This process of transfer learning is depicted in Figure 2. During training with this configuration, only the parameters of the final two layers get updated, without any effect on the frozen layer parameters. This approach preserves the features learned by the base layers from the source task to transfer into the target task. Further, we use Adam optimiser [19] to compute sparse gradients and binary cross entropy [20] to calculate the loss, which compares the difference between the predicted probabilities and the actual class labels.

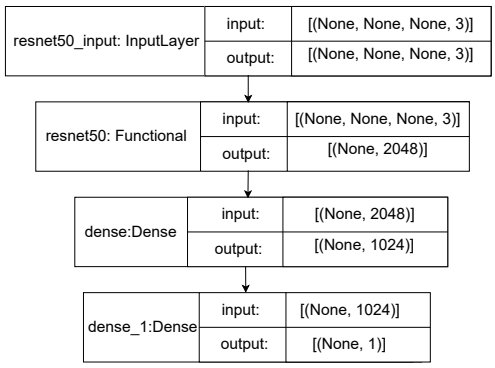


Fig. 2: The modified ResNet-50 model used for transfer learning

Preprocessing and Image Generation : To fit ResNet-50's image size requirement of 224×224 pixels, we first divide the extracted system call sequence into 224 equal parts which we call splits. We then count the occurrences of each of the five whitelisted system calls within each part/split . To normalize the count value into the image pixel's colour value range of [0-255], we first convert the count of each call into a percentage value within each split. Subsequently, we scale these percentage values to the range [0, 255]. This scaling operation ensures that gray-scale intensity values are spread across the full dynamic range of an 8-bit colour channel, where 0 represents black and 255 represents white. By distributing intensity values in this manner, we ensure optimal contrast and visual clarity in the resulting image. This process yields an image size of 5×224. To comply with ResNet-50's image size requirement, we repeat each colour value 44 times row-wise, and add four rows of zeros to extend the image size to 224×224. This gives us a single channel grayscale image. The channel are replicated three times to create an RGB image. Algorithm 1 shows our preprocessing and image generation process.

Algorithm 1 Preprocessing and Image Generation

Require:
- *whitelisted_ calls*: set of whitelisted system calls.
- *syscalls_ array*: array storing the system call sequence from the log file.
- *syscalls_ split*$_i$: Represents the i-th split of *syscalls_ array*, where $i = 1, 2, ..., 224$.
- *call_ counts*$_c$: Array storing the counts of the whitelisted call c in each split.

Store the system call sequence in the syscalls array
Split the syscalls array into 224 splits: syscalls split$_1$, syscalls split$_2$, ..., syscalls split$_{224}$
for each whitelisted call c **do**
  Create an empty array call_counts$_c$
  for each syscalls_split$_i$ **do**
    Count occurrences of call c in syscalls_split$_i$
    Append the count value to the array call_counts$_c$
  end for
end for
Stack the arrays for each split into a single array
Calculate the percentage of each split in the stacked array
Scale the percentage value to the range [0, 255] by multiplying by 2.55
Convert grayscale to RGB for ResNet compatibility
Repeat each colour value 44 times row-wise, achieving an image size of 220×224
Add four rows of zeros to extend the image size to 224×224, matching the ResNet model's requirements

Fig. 3 shows visual representations of system call sequences from four applications. The images depict how benign and malware applications interact with the system kernel and illustrate the relationship between different calls over time. Each row showcases a particular system call i.e, one of, `openat`, `read`, `write`,

`renameat`, and `unlinkat`, respectively from top to bottom. A sequence of system calls can be interpreted by observing from the leftmost to the rightmost of the image, i.e., observation from the start to the end of the log file. We utilize this representation as input for transfer learning to classify between malware and benign files.

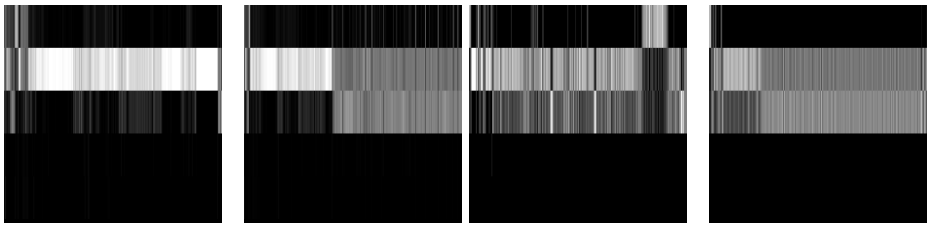
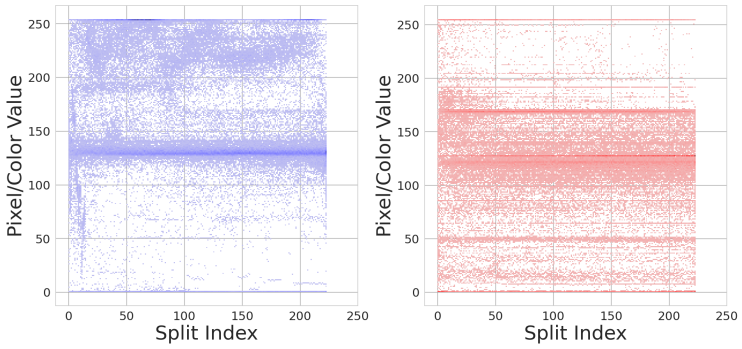


Fig. 3: Examples of images. First two are from benign apps and the last two from malicious apps

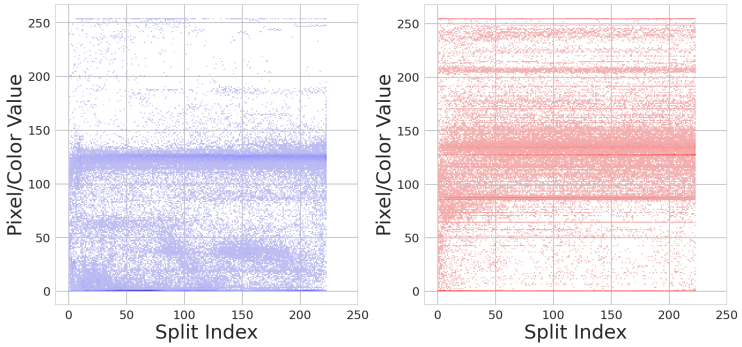
The presence of white chunks (255 colour value) in the second row on the first two images (benign applications) in Fig. 3 highlights the dominating `read` operations over the other operations, suggests that in those particular splits, mostly `read` calls have been made instead of other calls, which reflects the nature of benign processes. In contrast, the predominantly grey patterns observed in malware images (the last two images in Fig. 3) indicate interleaved `read` and `write` operations. This interleaved read-write operation aligns with the behavior of encryption-type ransomware, which typically operates by reading user files in specific block sizes and writing encrypted data back to files using similar block sizes, thereby establishing a systematic pattern of file access and modification. These distinct patterns provide valuable insights into the operational differences between benign and malicious processes, ultimately assisting the model for effective classification.

The heat maps in Fig. 4 highlight the distinction between `read` and `write` operations in benign and malware applications. The figure shows heat maps of pixel values associated with the split index (sequence in the log file) of all benign and malware applications. The distinctiveness between `read` and `write` can be observed in benign applications where the `read` color values are mostly higher than `write`'s. On the other hand, the color values between `read` and `write` mostly overlap in malware applications. This aligns with the findings in [9]. An image classifier should be able to capture these observations in its classification.

Synthetic Test Set : As Table 1 shows, our dataset contains limited number of samples.To evaluate our model with a larger dataset in the absence of real samples we chose to generate a synthetic test set mirroring the distribution of the original dataset. To create the synthetic dataset, we examined the pixel-wise distribution of the system call sequences across all the malware images in the original dataset. We then used the `distfit` package to identify the best-fit dis-



(a) Read Count in Benign Files (b) Read Count in Malware Files



(c) Write Count in Benign Files (d) Write Count in Malware Files

Fig. 4: Heat maps of the split index and its associated pixel/color value.

tribution in order to obtain the parameters needed to generate the synthetic set. For example, in order to generate the synthetic set corresponding to the pattern of the `openat` system call within each split of malware imagery, we first collected pixel values from the first pixel/split for the `openat` row in every malware image. This provided us with 34 values - one for each sample in our dataset. We then applied distribution fitting to these values to determine the range of values into which the first pixel value of `openat` falls. Using the best-fit distribution parameters, we proceeded to generate 1000 synthetic samples that follow the pattern observed in the original dataset, incorporating some inherent randomness introduced by the distribution parameters. This process was repeated for all system calls. We then randomly selected 10 sets, each containing 100 samples of real benign and synthetic malware data to evaluate our model. The algorithm for creating synthetic set is shown in Algorithm 2.

4 Results and Analysis

We trained, validated, and tested the ResNet-50 model on our collected system call datasets and the dataset provided by [9] using transfer learning and without using transfer learning to demonstrate the benefit of using the ResNet-50 model in the absence of sufficient data. In this sectuon we show the results of our evaluation. It should be noted that [9] collected system call logs for 2 minutes while we collected the logs for 5 minutes to capture more activities and patterns of malware interaction.

Algorithm 2 Synthetic Malware Image Generation

Require: – *images*: Malware Images Real Test Set.

Create an empty array synthetic_image
for $r \leftarrow 1$ **to** 5 **do**
  for $i \leftarrow 1$ **to** 224 **do**
    Create an empty array pixel_values
    for each image *img* in *images* **do**
      Append pixel values for $img[r][i]$ to pixel_values;
    end for
    Fit a distribution model on pixel_values using the *distfit* library;
    Sample a value from the fitted distribution;
    for $j \leftarrow 1$ **to** 44 **do**
      synthetic_image$[(44 \times (r-1)) + j][i] \leftarrow$ sample value;
    end for
  end for
end for
Generate an image using synthetic_image;

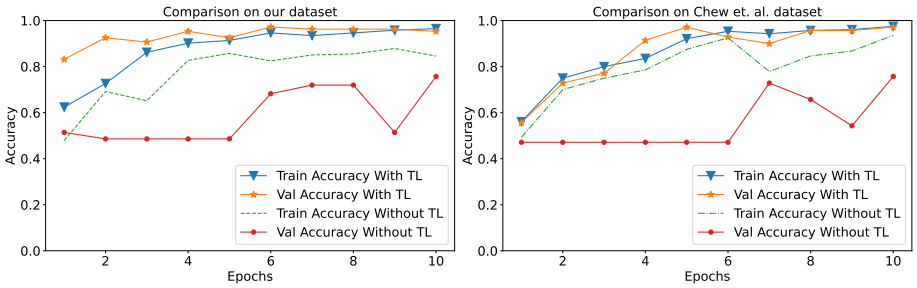
For the transfer learning (**with TL**), we trained two models: one with our collected real samples and the other with the dataset from [9], using 5-fold cross-validation for 10 epochs. Given the limited collection of ransomware samples, we employ 5-fold cross-validation to mitigate the risk of overfitting. We used 80% of the total samples for training and validation for cross-validation process and 20% for final evaluation using the best model identified through cross-validation. Table 2 provides the number of files for the distribution.

To compare the performance of the model without transfer learning (**without TL**) we trained ResNet-50 [17] model from scratch, similarly for two datasets. This requires approximately 25 million parameters to be trained. Considering our relatively small dataset, this leads to overfitting [18]. I contrast, with transfer learning, we only trained the last added dense layer with approximately 2 million parameters, which is an eighth of the original parameters. Consequently, after training 25 million parameters with a small dataset, the performance was poor on the validation set in 10 epochs. This can be seen in Fig. 5a that shows the training and validation accuracy of the best model through cross validation, with and without transfer learning on both datasets. Fig. 5b provides a visual

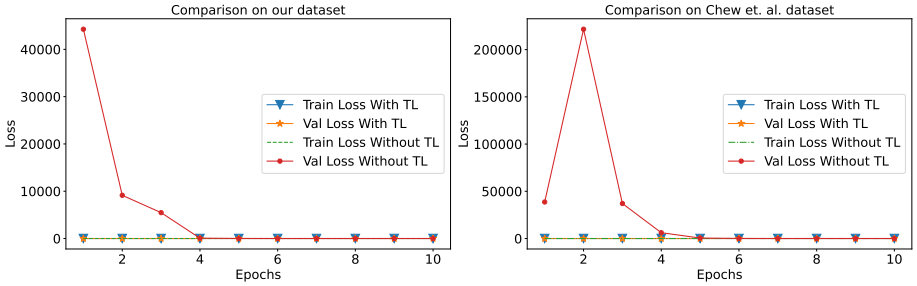
Table 2: Dataset Distribution for Cross-Validation and Evaluation

Dataset	**Classes**	**Training**	**Evaluation**
Chew et. al. [9]	Benign	180	45
	Malware	170	43
This work	Benign	269	67
	Malware	266	66

understanding of training and validation loss. Loss quantifies the penalty for inaccurate predictions made by the model at each point. Ideally, the loss should be zero. These figures demonstrate the effectiveness of using transfer learning with small datasets.



(a) Train and Validation Accuracy Comparison with and without TL



(b) Train and Validation Loss Comparison with and without TL

Fig. 5: Comparison of our approach with and without TL on two different datasets

Our results for the measurement for the model's performance are shown in Table 3. It can be seen that utilizing the pattern in only five system calls, without any complex pre-processing such as, assigning specific RGB colour codes [3] or risk scores [6], etc. provided an F1 score of 96% on our original dataset and

90% on [9]'s dataset. Importantly, it is evident that not using transfer learning, dropped the F1 score to 0.82% for our original dataset. In particular, looking at recall and precision metrics in Table 3, with TL, the model missed approximately 5% of positive samples and incorrectly predicted 3% of the samples as false positives on our original dataset. In contrast, without TL, the model misclassified 30% of the samples as false positives. For the dataset in [9], the model with TL incorrectly predicted 3% of samples as false positives, while without TL, this rate increased to 10%. However, both models, with and without TL, missed 17% of the positive samples.

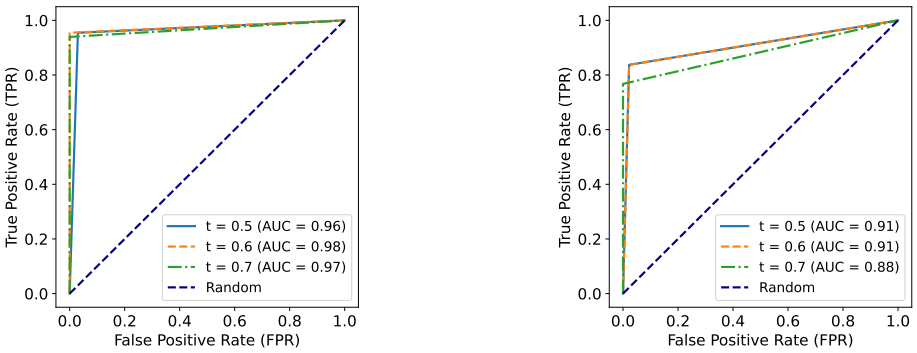
We also tested our model, on the synthetic dataset to see its performance on a larger dataset. We ran the model on 10 sets of synthetic test sets, each containing 100 benign and 100 malware samples. Table 3 shows the average results of the performance on all 10 synthetic test sets. Despite the noise introduced in the synthetic samples through the distribution parameters, the model correctly classified all of the true positive samples and incorrectly predicted approximately 2% of the samples as false positives. Given the lack of publicly available and working ransomware samples, we will explore more ways of creating synthetic samples in the future to thoroughly test the performance of the model with larger sets of test samples.

Table 3: Evaluation Metric

Test Set	Without TL				With TL			
	Acc	Rec	Pre	F_1	Acc	Rec	Pre	F_1
[9]	0.87	0.83	0.90	0.86	0.91	0.84	0.97	0.90
Ours	0.78	0.97	0.70	0.82	0.96	0.95	0.97	0.96
Synthetic	-	-	-	-	0.99	1.00	0.98	0.99

Note: Synthetic test set evaluation metrics represent the average results of the performance on all test sets.

We show the AUC-ROC (Area Under The Curve of the Receiver Operating Characteristics) curve in Figures 6a and 6b. The graphical representation illustrates the performance of the binary classification model across various prediction thresholds (0.5,0.6,0.7), on our original dataset and the dataset in [9] respectively. The higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. This quantifies the overall performance of the classification model, where an AUC value closer to 1 indicates superior classification ability, while a value around 0.5 suggests random guessing. It is worth noting that in our evaluation metric, we used the default 0.5 prediction threshold for both synthetic and real test sets.



(a) Our original Dataset

(b) Chew et. al's dataset

Fig. 6: AUC-ROC curves

5 Discussion

As we mentioned in the Introduction to this paper, in a layered defense model for detecting ransomware, should every layer rely on solutions with complex methodologies? Or is there a case for solutions that prioritise early detection over high accuracy? While the answer to this question will depend upon the weightage given to computational requirement, timeliness and evaluation metrics in any given scenario, our work highlights the importance of asking this question in image-based ransomware detection where early detection is paramount and a trade-off may be needed, especially when datasets are small. Table 4 shows a qualitative comparison of the literature with our work. Our model, trained with a minimal feature set, achieved an accuracy as well as an F1 score of 96% on the original dataset compared with techniques that incur high computational costs. Comparing our F1-score with the F1 scores reported in these works we see that the penalty of using a small dataset, limited features as well as a model trained using transfer learning is at most 1% when compared with Dam et. al. [8] and 3% when compared with Rustam et. al. [7].

6 Conclusion

In this work, we have demonstrated that a small set of carefully inspected system calls can provide acceptable results in image-based ransomware detection compared to using extensive feature lists extracted from complex feature engineering tasks. However, it is important to note that our chosen tool for procuring system call logs, Strace, is susceptible to tamper by sophisticated malicious programs that may hide their tracks, as Strace operates at the user level where such programs might already have access. In a real world scenarios, tamper-proof provenance tools such as Progger [21] is advised to use. Having said that, our methodology offers a practical solution in scenarios where data constraints

limit model training and testing capabilities, serving as a viable alternative in comparison with complex ransomware detection approaches.

Table 4: A comparative summary of existing literature on malware detection.

Method	**Dataset**	**Number of Features**	**F1-Score**	**Note**
CNN on feature images generated using colour mapping rules [3]	9000 malware samples from `VirusShare` & modified Malimg	16	0.98 (for ransom family), overall (0.99)	Results show that visualisation and CNN are effective for malware classification.
Five ML classifier (KNN, LR, SVM, RF, DT) on n-grams (length 2-5) on features selected by Em-RmR [4]	1354 ransomware samples from various sources	90	SVM - 0.99 DT - 0.81 KNN - 0.36 LR - 0.75 RF - 0.55	Time-intensive feature engineering
Ensemble and individual ML techniques on behavior features include system calls, binders, and complex Android objects [5]	CICMalDroid-2020	30	0.96	Small sample sizes affects the approach and provide less accurate classification results
Transfer learning on grayscale images using Bi-models(VGG16 and ResNet-50) and ML classifiers (SVC, RF, LR) [7]	Mailimg	2048	1.0	High computational cost in training time.
SVM and DT on system-call sequence grouped by function risk value [6]	AWSCTD	1000	SVM - 0.98 DT - 0.97	This method requires a sufficiently large dataset with distinct separation between clean code and malware functions.
Transfer learning using peIRCECon model that integrate ResNet-50 and VGG16 on visual representation of PE headers [8]	8399 self-collected ransomware samples	12	0.97	While the approach has shown promising results, it is not immune to the advanced ransomware techniques, which may affect its future performance.
ResNet-50 on channel-replicated images of system calls **(this work)**	332 self-collected android ransomware and 1000 synthetic samples	5	0.96	Acceptable classification accuracy and F1-score with using only five features and minimum processing in terms of training time.

References

1. C. Kidd, "Ransomware families & raas groups," *splunk*, 2023.
2. "Cybercrime to cost the world $9.5 trillion usd annually in 2024," *Cybersecurity Venture*, 2023.
3. M. Tang and Q. Qian, "Dynamic api call sequence visualisation for malware classification," *IET Information Security*, vol. 13, no. 4, pp. 367–377, 2019.
4. Y. A. Ahmed, B. Koçer, S. Huda, B. A. Saleh Al-rimy, and M. M. Hassan, "A system call refinement-based enhanced minimum redundancy maximum relevance

method for ransomware early detection," *Journal of Network and Computer Applications*, vol. 167, p. 102753, 2020.
5. P. Bhat, S. Behal, and K. Dutta, "A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning," *Computers & Security*, vol. 130, p. 103277, 2023.
6. T. Vyšniūnas, D. Čeponis, N. Goranin, and A. Čenys, "Risk-based system-call sequence grouping method for malware intrusion detection," *Electronics*, vol. 13, no. 1, 2024.
7. F. Rustam, I. Ashraf, A. D. Jurcut, A. K. Bashir, and Y. B. Zikria, "Malware detection using image representation of malware data and transfer learning," *Journal of Parallel and Distributed Computing*, vol. 172, pp. 32–50, 2023.
8. T. Q. Dam, N. T. Nguyen, T. V. Le, D. L. Tran, S. Uwizeyemungu, and T. Le-Dinh, "Visualizing portable executable headers for ransomware detection: A deep learning-based approach," *Journal of Universal Computer Science*, vol. 30, no. 2, pp. 262–286, 2024.
9. C. J.-W. Chew, V. Kumar, P. Patros, and R. Malik, "ESCAPADE: Encryption-type-ransomware: System call based pattern detection," in *Network and System Security*, pp. 388–407, Springer International Publishing, 2020.
10. H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A survey on ransomware: Evolution, taxonomy, and defense solutions," vol. 54, no. 11s, 2022.
11. Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–40, 2017.
12. A. Jyothish, A. Mathew, and P. Vinod, "Effectiveness of machine learning based android malware detectors against adversarial attacks," *Cluster Computing*, pp. 1–21, 2023.
13. L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, pp. 1–7, 2011.
14. DjPasco, "Awsctd." GitHub repository.
15. ImageNet, "ImageNet: A Large-Scale Hierarchical Image Database," 2021.
16. C. J. W. Chew, V. Kumar, P. Patros, and R. Malik, "Real-time system call-based ransomware detection," *International Journal of Information Security*, pp. 1–20, 2024.
17. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
18. J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), p. 3320–3328, MIT Press, 2014.
19. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
20. L. Liu and H. Qi, "Learning effective binary descriptors via cross entropy," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1251–1258, 2017.
21. T. Corrick and V. Kumar, "Design and architecture of progger 3: A low-overhead, tamper-proof provenance system," in *Inernational Conference on Ubiquitous Security*, pp. 189–202, Springer, 2021.