CSEC759 (Advance Malware Forensics) Final Project Report

Praj Sanjay Shete
Dept. of Computing Security
Rochester Institute of Technology
Email: ps7600@rit.edu

I. Introduction

In recent years, the threat of malware attacks has increased significantly, leading to the loss of sensitive information and financial resources. Among various malware families, ransomware is a type of malware that is specifically designed to block access to computer systems or data until a ransom is paid. The outcome of this project can be used as an initial defense against ransomware attacks and can help to prevent significant financial losses to organizations and individuals.

A. Malware Area focused on

Ransomware is one of the most prevalent and dangerous forms of malware and poses a significant threat to organizations and individuals alike. In this project, I have worked on developing a deep learning-based image classification model to detect ransom notes in images. The model is trained using a dataset containing two classes - ransom note and non-ransom note. We employed a convolutional neural network (CNN) architecture with multiple layers, including convolutional layers, max-pooling layers, and dropout layers, to classify images. The proposed model can help in detecting ransomware attacks by analyzing the images containing ransom notes.

B. Problem solved

In the dynamically changing attack types, signature based detections are no more effective, the attacks are now more sophisticated and hence detecting such attacks is a great challenge for the organisation. In case of malware attacks, behavioral based detections is the most researched area. One such feature I observed in ransomware attacks is the ransom note which has been remaining consistent in all attacks. Since, the consistency of this feature remains in all attacks I decided to use this feature to detect the presence of a ransomware. I reviewed [1], which analyzes ransom notes in.txt and.html files, as a point of reference. My approach, however, differs from this since I choose screen lockers rather than textual notes.

II. Tool Description

The tool I built is a CNN model that trains a set of dataset and predicts the probability of the class the image belongs to. The dataset consists of 208 ransom note images as one of the class and other class of Windows desktop consist of 200 images. The program takes the dataset path as the input to feed for the training. Before training it pre-processes the images loaded by applying uni-dimensional size, converting color profiles, these are mandatory since computer visions libraries use these standards.The program then splits the entire data into training and testing image sets, in my code I have used the split ratio as 30%. My tool defines a sequential model in TensorFlow Keras with the following layers:

- A 2D convolutional layer with 32 filters, each of size 3x3, and ReLU activation function, with input shape of (224, 224, 3)
- A max pooling layer with pool size of 2x2
- A 2D convolutional layer with 64 filters, each of size 3x3, and ReLU activation function
- Another max pooling layer with pool size of 2x2
- A dropout layer with a rate of 0.5, which randomly drops 50% of the input units to prevent overfitting.
- A flatten layer to convert the 2D feature maps into a 1D feature vector
- A fully connected dense layer with 64 units and ReLU activation function
- Another dropout layer with a rate of 0.5
- A fully connected dense layer with 2 units and softmax activation function, which outputs the probabilities of the input image belonging to each of the two classes (not a ransomware and ransomware).

I then used adam optimizer and categorical cross entropy as my loss function to compile the model. I provided 10 epochs with a batch size of 32 and provided 20% validation data to testing. My program then takes input image path which the model has never seen before, it pre-processes the image and predicts the class of the image by comparing the probability value obtained.

III. Overall Architecture

CNNs (Convolutional Neural Networks) are a popular deep learning model used in many machine learning applications, including the analysis of images and videos, NLP, and other domains. Convolutional layers are used by CNNs to extract features from input photos. A convolutional layer is made up of a number of filters or kernels that move over the input image while computing a dot product for each small area of the image they pass over. A feature map is created as a result, emphasizing specific patterns or features in the image.

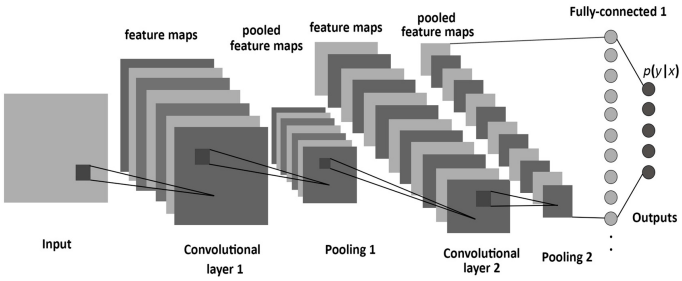

Fig. 1. CNN Architecture [2]

In order to add non-linearity to the model, the output of the convolutional layer is then sent through an activation function, such as ReLU. The feature map is then compressed using a pooling layer by taking the greatest or average value in each individual tiny region. The output of the pooling layer is then passed through a second set of convolutional and pooling layers, each of which learns increasingly complex and abstract features from the one before it. In order to classify the input image, the final output of the CNN is often passed via one or more fully connected layers and an output layer.[Fig.1] [2]

IV. Installation and Usage

A. Installing Dependencies

- Download and Install Python latest version from https://www.python.org/downloads/
- Download and Install Visual Studio Code from https://code.visualstudio.com/
- From Extensions tab in Visual Studio Code install Python Extension provided by Intellisense
- Open Command Prompt having admin rights and install opencv library using the command: pip install opencv-python
- Install the tensorflow library using the command: pip install tensorflow
- Install scikit-learn(sklearn) library using the command: pip install scikit-learn

B. Code Execution

- Extract the Zip folder attached in the submission.
- Open the Final_Proj.py file in Visual Studio Code.
- Provide path to the dataset on the line mentioned DATASET_PATH (Make sure the path is complete. Visual Studio doesn't accept backslash(\) in the path change the backslash to frontslash(/))
- To provide the input image to the model classification provide the input image path(You may choose any 1 from the project directory, as all of them are new to the model).
- After running the code the model will first get trained and provide classification results.

V. Results from Testing Process

For testing I ran the code on the machine having Intel Core i9 processor, 32Gb of RAM and having a Intel iRIS-Xe 1080 GPU. The testing accuracy of the model fluctuated from

Fig. 2. Input 1 (Ransom Note)

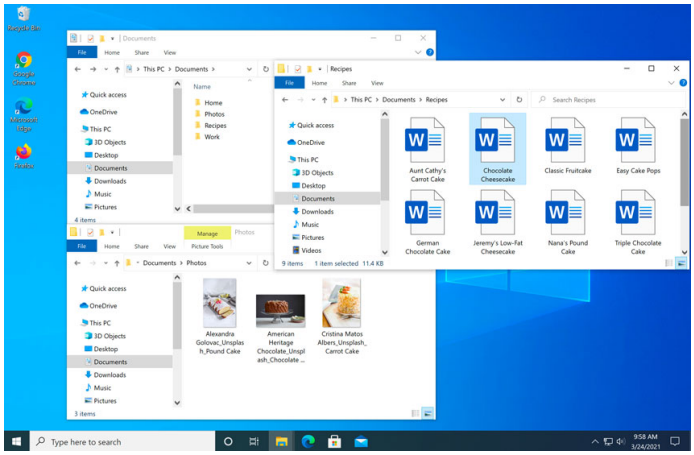

Fig. 3. Input 2 (Not A Ransom Note)

50-60% after each run. The low accuracy I am assuming is because the limited dataset and of course the hyper-parameters and overall architecture of the model, which I will try to improve in my future work. For testing I provided 4 images, 2 of class ransom note (Fig.(2), Fig.(4)) and 2 images of class not a ransom note (Fig.(3), Fig.(5)). The images were completely unknown to the model and the model was able to classify the class of the images provided. However, the prediction score was marginal, this was again possibly due to limited training of the model as the dataset might not be sufficient. Fig.(6) represents the output of Fig.(2) as the input. Fig.(7) represents the output of Fig.(3) as the input. Fig.(8) represents the output of Fig.(4) as the input. Fig.(9) represents the output of Fig.(5) as the input.

VI. Final Measurements

The tool is working satisfactorily although the improvements can be made and is in a development stage, it needs more attention with dataset and architecture perspective. The

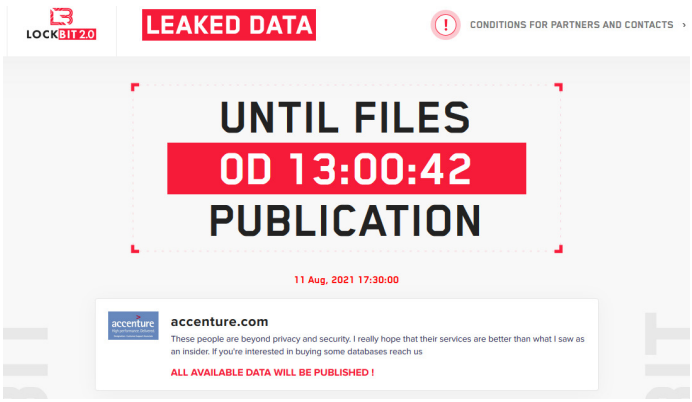

Fig. 4. Input 3 (Ransom Note)

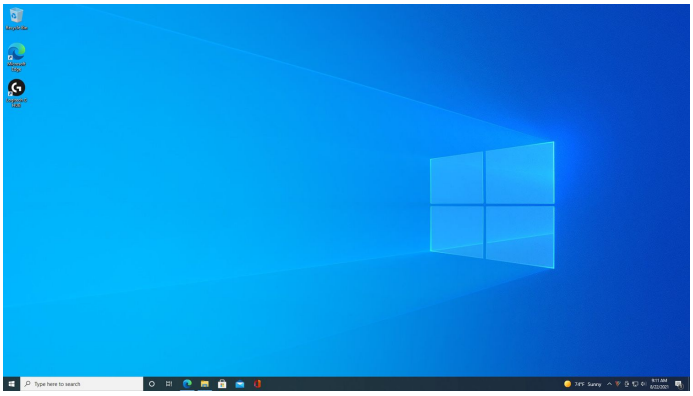

Fig. 5. Input 4 (Not a Ransom Note)

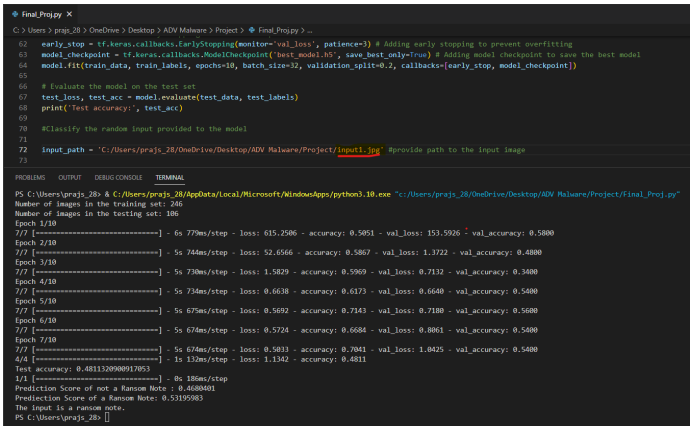

Fig. 6. Output for Input 1

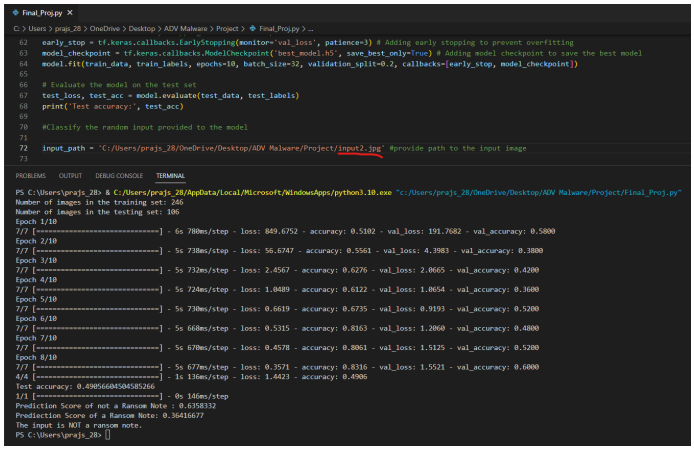

Fig. 7. Output for Input 2

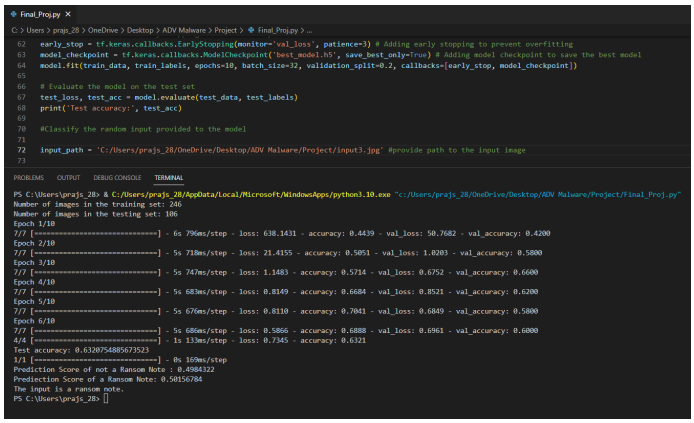

Fig. 8. Output for Input 3

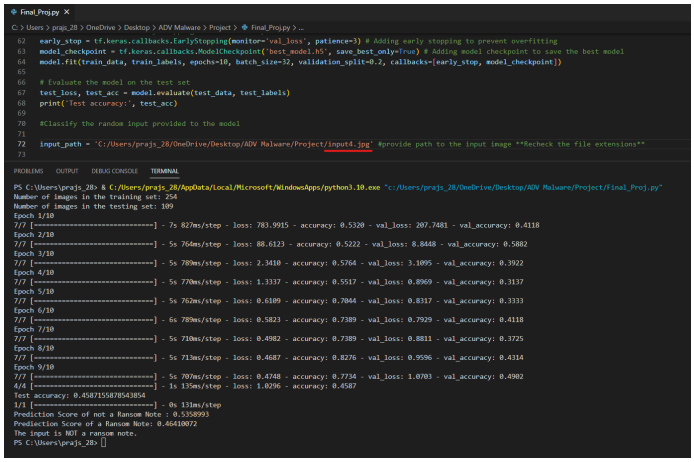

Fig. 9. Output for Input 4

image classification is been done on specific type of images and needs more variety. As of now the tool is predicting the class of the image.

VII. Tool Use Case

Dynamic malware analysis refers to a technique for analyzing malicious software (malware) by executing it in a controlled environment, typically a virtual machine, and monitoring its behavior as it runs. This allows security researchers to observe the malware's actions, including any attempts to modify system files, create network connections, or perform other suspicious activities. Dynamic malware analysis is often used as part of the malware analysis process to help identify the specific behavior of a given piece of malware. By observing the malware's actions in a controlled environment, security researchers can gain insight into its intended purpose and potential impact on targeted systems.

The tool will be helpful in determining the surety of the ransomware presence in the machine. Malware attacks have now been more sophisticated and detecting such malware's is complex. Even though technologies such as ML help determine the behavioral characteristics of a malware, the malware authors continuously evolve and try and hide their malware presence into the victim machine. Ransom note is one such feature that has remained consistent throughout.

My model can be integrated with dynamic analysis tools and can be in operation when the dynamic analysis tool detects the presence of suspicious files through triggers [Fig.10]. These triggers can be:

- Specific text such as "Your Files have been encrypted", the tool will then take a screenshot of the note and pass it through the model for detection.
- File extensions found: When ransomware encrypts files, it typically adds a specific file extension to the encrypted files, such as ".locky" or ".cryptowall". Monitoring file changes and taking a screenshot whenever it detects files with these specific extensions.
- Other suspicious artifacts: These include the dynamic analysis tools that always hunt for such as API calls, Network traffic, registry changes, files/ payloads downloaded from C2, tracking memory usage etc. such condition matches can help the tool to take the screenshot of the machine.

VIII. Future Work

Current model is able to classify screen locker notes that are present in the form of a decrypter software which demands bitcoin payments to the attackers address. However, the model fails to classify the ransom notes left on the desktop present in the .txt format. I will leave this to the future work, where the model can learn text patterns either by using NLP or the character based convolutional neural network or the LSTM as present in the research paper [1].

Secondly, the models testing accuracy is not up to the mark and needs some extra attention to bump up the accuracy. This can be due to the small dataset containing of only around 200

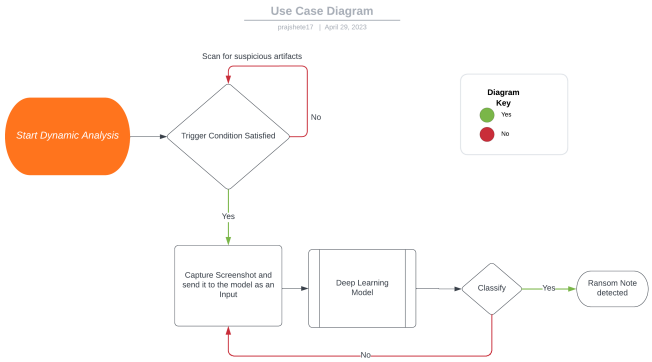

Fig. 10. Use Case Diagram

images, due to limited availability of images it was a challenge to gather dataset of ransom notes, going ahead I will possibly try to increase the dataset and improve the accuracy of the model.

References

[1] Y. Lemmou, J.-L. Lanet, and E. M. Souidi, "In-depth analysis of ransom note files," *Computers*, vol. 10, no. 11, p. 145, 2021.

[2] S. Albelwi and A. Mahmood, "A framework for designing the architectures of deep convolutional neural networks," *Entropy*, vol. 19, no. 6, p. 242, 2017.