MalFam: A Comprehensive Study on Malware Families with state-of-the-art CNN architectures with classifications and XAI

by

Abid Hossain Haque
20101453
Labiba Ifrit Jahin
20101467
Sheikh Yasir Hossain Katib
20301013
Saiwara Mahmud Tuhee
20101465
Maisoon Tasnia
20301076

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
January 2024

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

[illegible]
Saiwara Mahmud Tuhee
20101465

Tasnia
Maisoon Tasnia
20301076

Katib
Sheikh Yasir Hossain Katib
20301013

[illegible]
Abid Hossain Haque
20101453

Labiba
Labiba Ifrit Jahin
20101467

Approval

The thesis/project titled "MalFam: A Comprehensive Study on Malware Families with state-of-the-art CNN architectures with classifications and XAI" submitted by

1. Abid Hossain Haque (20101453)
2. Labiba Ifrit Jahin (20101467)
3. Sheikh Yasir Hossain Katib (20301013)
4. Saiwara Mahmud Tuhee (20101465)
5. Maisoon Tasnia (20301076)

Of Fall, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 18, 2024.

Examining Committee:

Supervisor:
(Member)

[illegible]

Dr. Muhammad Iqbal Hossain
Associate Professor
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)

Faisal

Md Faisal Ahmed
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

We hereby declare that this thesis is based on our own fndings from our own research discoveries. All other sources used in this work have been properly acknowledged. Furthermore, we confrm that this thesis has not been submitted or presented, either in its entirety or partially for the purpose of receiving a degree from any other educational institution or university.

Abstract

Just as the digital transformation of everything in this ‘Information Age’ has acted substantially to mitigate conventional crimes to a degree, the rate of cyber crime has parallelly elevated alarmingly. As malware has been the primary envoy in such criminal incidents, its metamorphosis is highly prevalent. This paper presents a systematic grouping of malware samples into distinct families extracted from two prominent datasets, MalImg and MaleVis through extensive research. Subsequently, six state-of-the-art advanced CNN architectures have been utilized including Inception ResNet V2, DenseNet, VGG16, ResNet50, EfficientNetB0 and XceptionNet. Then a comprehensive analysis of malware classification was conducted as the research aimed to discern the performance variances among these models concerning the classification of diverse malware families. Moreover, eXplainable Artificial Intelligence (XAI) techniques, particularly Local Interpretable Model-agnostic Explanations (LIME) has been introduced, to deduce the rationale behind the classification decisions made by each model. This involved analyzing and visualizing the salient features within the malware files that led to their identification as malicious entities. Lastly, the findings of this study not only provide a comparative evaluation of various deep learning architectures for malware classification but also offer insightful explanations through XAI methodologies, shedding light on the interpretability of model decisions in the realm of cybersecurity. The results furnish valuable insights for enhancing the understanding of malware behaviour and model interpretability, thereby contributing to the advancement of robust and explainable malware detection systems.

Keywords: MaleVis, MalImg, comparative analysis, Convolutional Neural Network (CNN), Inception ResNet V2, DenseNet, VGG16, ResNet50, EfficientNetB0, XceptionNet, XAI.

Dedication

We want to dedicate all of our sacrifces and educational efforts to our great parents, without whom we would be worthless. We also dedicate our thesis to Dr. Muhammad Iqbal Hossain sir, who served as our supervisor, guided us, and showed us how to develop our skills and personalities as successful professionals.

Acknowledgement

Firstly, all praise to the God for whom our thesis have been completed without any major interruption. Secondly, to our Supervisor Dr. Muhammad Iqbal Hossain sir for his kind support and advice in our work. He helped us whenever we needed help. And finally to our parents without their throughout support it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

List of Figures

Chapter 1

Introduction

The existence of malware can be dated back to the inception of computers or the idea of ‘computing’. Any type of malicious software that is engineered to cause damage to devices, exports personal data or passwords, mostly always infiltrates a device without the owner’s knowledge is termed as a malware. Despite the level of destructiveness, all malwares follow some basic pattern of invasion. Most common are unknowingly downloading a malicious software, clicking on infected links or to say visiting an infested website or through email attachments and so on. In case of different malwares, the reason why they get triggered varies but not vastly. For example: actions such as clicking on a compromised link within an email or visiting a deceptive website could trigger the malware. Other widely used ways are to distribute malware through peer-to-peer file-sharing platforms, bundled within free software downloads or in case of spreading malware among a large number of users, inserting malicious code into a widely used torrent or download is an efficient method for attackers [1].

Malware comes in many faces and differs themselves through their type of attack, infiltration, installation and casualties. Trojan, adware, spyware, trojan downloaders, dialer, ransomware, scareware, fileless malware etc are some of the many faces of malwares. Now-a-days, Advanced Persistent Threats (APT) which is an advanced malware software has caused a stir in the cyber world as it has become a common tool for attackers. This software uses many vectors and often serves multipurpose commercial and military uses. Users of this software can commit a wide range of cyber crimes including organized crime and state sponsored attacks against e-government sites. [2]

The analysis of malware code is not an easy task to begin with as comprehension of how a malware works, its attack system, code etc. can be very complicated. Thus, implementing deep learning on a wider scale can generate even more optimistic results due to its automatic feature extraction, adaptability etc. Though plenty of research has been carried out in this regard, it falls inadequate to the growing number of attacks and mutations of malware families growing every day.

In the domain of malware classification, CNNs play a pivotal role in analyzing binary files or visual representations of malware, such as disassembled code or byte sequences transformed into images. By training CNN models on these image-based datasets, the networks learn to identify subtle patterns and distinctive features indicative of malware behaviour and the architecture can discern nuanced differences in these images, enabling the detection of previously unseen malware variants.

Furthermore, the significance of CNNs in malware classification lies in their ability to automatically extract relevant features from images representing malware characteristics. This feature extraction capability empowers the models to generalize well to new and unseen instances thus enhancing the accuracy and efficiency of malware detection and classification systems. Moreover, CNNs can aid in understanding the underlying structure and patterns within malware and subsequently contributing to the development of more robust cybersecurity measures.

1.1 Research Problem

MaleVis and MalImg are two well-known datasets that have been applied in the research. Most of the time, users of these datasets are not aware of which malware belongs to which family. This study attempted to address that issue by categorizing malwares into families according to their pattern of infestation. this has been done through extensive research of the malwares and subsequent families. 45 malwares from both datasets were selected based on the availability of technical informations and grouped into 8 classes of malware families from. Moreover, MaleVis is recognized to be a balanced dataset while MalImg is known to be an imbalanced dataset. However, the purpose of this study is to create a balanced dataset, therefore, data augmentation and shut.li have been used to achieve this goal. Here, if there are less than 350 malware samples in a folder, some samples are randomly selected copied to reach the 350 mark. Again, in folders containing samples more than 350, samples were randomly selected and omitted.

The dataset was fed into six state-of-the-art CNN models. It is known that balanced and imbalanced datasets have differing levels of accuracy. The main goal of this research is to identify the rationale and causes underlying the accuracy of the existing models. Then, by using the power of XAI, in this case LIME, it was possible to comprehend and provide an explanation for the reason for an image's detection as malicious. A part of the image is highlighted whenever LIME is run, aiding in the identification of the image as malicious. Additionally, if a picture is not highlighted, the model is unable to identify it as malicious. So. It can be said that the necessity for malware classification stems from a number of factors.

1.2 Research Objectives

By using the CNN models, the research seeks to improve malware detection accuracy. The primary goals of this study are -

- Identifying and classifying malware families using the MaleVis and MalImg datasets with proper logics and arguments for malware family classification.

- Addressing the imbalanceness MalImg dataset and opting for a balanced dataset by applying shut.li approache and data augmentation.
- Implementing the balanced dataset to run six Convolutional Neural Network (CNN) models and evaluating the models' performance.
- Employing the CNN models with LIME and using eXplainable Artificial Intelligence (XAI) to decipher and elucidate decisions made by models.
- Creating a method to determine whether an image contains malware by utilizing trained models and examining instances where the model fails to recognize an image as malicious.
- Improving CNN models' interpretability to detect malware and providing insightful explanations of the variables affecting current models' accuracy in the context of malware detection using images.

https://www.overleaf.com/project/659ab8876312e0d5f389c081

1.3 Thesis Structure

- The research problems, objectives, and thesis structure is discussed in the first chapter.
- Then the literature review and background of model architectures and behaviors are presented in the second chapter.
- The third chapter describes the dataset and the preprocessing steps for classifying malware families.
- Chapter four presents the proposed models and the conducted experiments.
- Lastly, the results, such as model-wise comparisons and XAI analysis is discussed in the fifth chapter.
- Chapter six concludes the research and outlines future work.

Chapter 2

Literature Review

2.1 Related Works

The investigation outlined in reference [5] centered on employing a CNN-based deep learning framework for detecting Android malware. The primary focus involved identifying various malware families using an image-based strategy. Their proposed method harnessed the Vgg-16 architecture as a transfer-learned model for the identification and detection of malware families within the Android OS. Thus rather than relying on feature representations, their approach involved collecting RGB images from unprocessed malware binaries. and these images served to encapsulate intricate high-level attributes, effectively discerning between diverse malware families. To gather additional data from the source code RGB images were directly extracted from the raw APK files. Due to the large size of directly created images from APKs the convolutional layers were unable to handle the feature extraction part. As a result, the researchers trained the Vgg-16 model. The proposed CNN-based deep learning architecture for detecting and identifying malware families in the Android operating system achieved an impressive accuracy rate of 97.81%. This performance surpasses earlier studies conducted on the same topic.

Additionally, the study[6] approached multiclass malware classification using transfer learning based on image visualization. The proposed work was carried out on the MalImg Dataset, and the visualization process starts by converting the malware binaries into unsigned integers, later converting them into pixel values. The classification was conducted employing a hybrid model based on transfer learning principles. In this approach, an IVMCT framework incorporated three pre-trained models: ResNet, AlexNet, and DenseNet and the selection of these models was strategic, utilizing feature-based transfer learning to combine their features and generate fully connected layers. Ultimately, the terminal output incorporated the softmax activation function and the accuracy achieved by the IVMCT framework surpassed numerous other methods, achieving an accuracy rate of 99.12 percent. Additionally, it demonstrated benefits including real-time detection, reduced time consumption, and outperforming several previously utilized techniques.

This study [7] focused on classifying malware images, proposing a unique and lightweight deep Convolutional Neural Network (CNN) for this purpose. Their approach involved using a CNN model with the Adam optimizer to extract distinctive properties from malware samples and categorize them into specific malware families. As the aim was to provide valuable information for non-expert human analysts and enhance malware security in IoT applications, the model underwent multiple trials, demonstrating an impressive accuracy rate of 96.64%. Thus, notably, the developed CNN model was lightweight, making it suitable for resource-constrained applications.

For the purpose of categorizing malware binaries as images, the research [8] proposed employing six distinct Convolutional Neural Network (CNN) models. These models, namely CNN-SVM, GRU-SVM, MLP-SVM, Inception V3, ResNet 50, and VGG16, are utilized to process Portable Executable (PE) files containing malware. The study uses the MalImg daatset and compares the effectiveness of several established CNN-based deep learning models from ILSVRC competitions alongside additional CNN and mixed CNN models in classifying malware images and employ a static analysis technique to automatically extract features. As per the findings, the Inception V3 model displays superior performance compared to the M-CNN model, which is currently the state-of-the-art system, achieving a test accuracy of 99.24% in contrast to 98.52
Moreover, the study [9] suggest the creation of the first file-independent DL that uses executable file visual characteristics to categorize malware into families which is accomplished by educating a network of convolutional neural networks to represent the binary content of malware as grayscale images. Moreover, grayscale graphics are used to represent malware executables because they can be used to detect variations in samples since they can capture subtle changes while preserving the overall structure. The suggested neural network design encompasses an input layer and three sets of feature extractors, each comprised of four stages and these stages are responsible for learning hierarchical features by employing convolution, activation, pooling, and normalization layers. Additionally, to comprehensively assess the approach, the study utilized two openly accessible datasets: MalImg and the Microsoft Malware Classification Challenge datasets. Lastly,The evaluation process involved using common metrics like accuracy, precision, recall, and F1-score. It also compared image-based machine learning methods for categorizing malware. The study found that the proposed model performed better than previous methods, achieving accuracy rates of 98.48% and 97.49% for the respective datasets.

The study [10] illustrates the efficacy of data augmentation in mitigating the challenge posed by limited dataset sizes and focuses on leveraging the MalImg dataset for malware classification, encompassing 9,339 samples distributed across 25 diverse families, albeit with an imbalanced distribution. To address this imbalance, the Study has utilized data imbalance techniques as inputs to a CNN model and the model proposed in the study, showcases superior performance when compared to established CNN models like VGG16 (96.96%), ResNet50 (97.11%), InceptionV3 (97.22%), and Xception (97.56 %), achieving an accuracy of 98.03%. Notably, the suggested CNN model also demonstrates notably faster execution in contrast to other CNN models and on top of that the study incorporates a support vector machine into the proposed CNN model.

A novel model is introduced in the study [11] that aimed at identifying malware, which integrates similarity mining and a specialized deep learning structure for image examination. This hybrid deep learning model employs efficient image-based machine learning methods to effectively detect and categorize malware and to evaluate its performance on extensive collections of new malware families, the authors conducted tests using various public and private datasets. Additionally, supervised and unsupervised learning techniques was combined for image-based malware identification, resulting in the creation of the hybrid model. This model encompassed one unsupervised learning approach and two supervised learning models, enabling it to identify known malware, variations of known malware, and unfamiliar malware using a self-learning system and initially, the process converted malware binaries into grayscale images through an image-based deep learning technique. Deep learning frameworks, particularly a fusion of convolutional neural network (CNN) and long short-term memory (LSTM) models, were utilized to grasp both spatial and sequential information for capturing intricate attributes. Furthermore, the proposed deep learning architecture leveraged CNN and a bidirectional pipeline to efficiently categorize malware.

This experiment [12] was to employ a bespoke Convolutional Neural Network (CNN) and a deep neural network to classify malware photos into their respective families that demonstrated impressive performance by utilising CNNs' power in computer vision, achieving a 98.07% classification accuracy during testing and a 99.64% classification accuracy during training. Furthermore, transfer learning using the VGG-16 model demonstrated excellent accuracy, scoring 88.40% in testing and 87.37% in training, thus the relevance of CNNs in effectively handling the difficult work of malware classification was clear, even though ResNet-18 and Inception-16 attained validation accuracies of 90.21% and 92.48%, respectively.

This work [13] explores the field of machine learning (ML)-based Android malware detection, which has received a lot of attention from researchers studying mobile security. Although numerous studies assert elevated detection accuracy, this study highlights the dangers of impractical experimental designs that result in excessively optimistic results. Unlike previous research, Explainable AI (XAI) techniques are used in this study to find out what ML models learn as they are being trained. The results show that the training dataset's temporal sample irregularity leads to inflated classification performance, emphasizing a dependence on temporal variations rather than real harmful behaviours. Crucially, the research emphasizes the value of XAI in understanding the inner workings of malware classifiers based on machine learning, calling for a change in emphasis from accuracy to a more thorough comprehension of model behaviour.

The goal [14] of this research is to identify dangerous programmes by combining information from images with deep learning capabilities. Using the 'Malimg' dataset, the study uses visual representations to depict well-known malware families. The suggested methodology extracts features from malware sample byteplot grayscale pictures using a pre-trained model VGG16. These characteristics are then employed to accurately classify malware into distinct families by training a variety of classifiers, such as SVM, XGBoost, DNN, and Random Forest and their usefulness in obtaining high accuracy in identifying and categorizing malware families is demonstrated by the experimental evaluations. Thus, the study demonstrates the promise of image-based characteristics and deep learning methods for accurate malware identification.

The research [15] presents a novel use of the Xception model which is less prone to overfitting problems and more durable than popular models like VGG16. The Xception model performs exceptionally well in reaching the highest training accuracy and validation accuracy among other approaches, despite its limited usage in the literature currently in publication and lack of prior study in malware classification thus this demonstrates how effective Xception is at handling the challenging task of classifying malware images. The study emphasizes the special advantages of the Xception model, establishing it as a potent instrument in the field of deep learning for malware analysis.

The research [16] suggests a novel visualization-based method to get around these problems. Malware binaries are categorized using a deep learning network, namely DenseNet, and displayed as two-dimensional pictures. To solve class imbalance, the system uses a reweighted class-balanced loss function, which leads to notable performance gains. The system's excellent accuracy in detecting new malware samples is demonstrated by experimental findings on four benchmark datasets (98.23% for Malimg, 98.46% for BIG 2015, 98.21% for MaleVis, and 89.48% for the unseen Malicia dataset.

Lastly, the study[17] uses EfficientNet-based models that convert application files into images for deep learning analysis and in order to reduce dimensionality, the research uses kernel principal component analysis (KPCA) to extract unique features from the EfficientNet models. The meta-classifier that is created by fusing these reduced features performs better than DenseNet, ResNet, and InceptionResNet. The EfficientNet-based solution outperforms the state-of-the-art techniques and demonstrates robustness and generalizability on two different testing datasets. [17]

2.2 Background

2.2.1 CNN Architectures

Inception ResNet V2

Inception ResNet V2 is introduced as a combined version of two major architectures namely Inception and ResNet. The architecture includes advanced inception modules and rhese modules are notable for their capacity to record information at numerous sizes, allowing the network to distinguish details at different levels of abstraction. Inception ResNet V2 also incorporates residual connections which are intended to address the vanishing gradient problem, which is a typical challenge in deep neural network training. They make gradients flow more smoothly during the training process. The model is characterized by its depth and complexity, indicating that it has a significant number of layers and intricate designs. The depth and complexity are highlighted for their roles in facilitating multi-scale feature extraction. This means that the model can efficiently recognize patterns and details within the data at various levels of granularity. Within the dataset, the architecture is claimed to collect both local and global contextual information. This means that the model can recognize not just fine-grained details (local information), but also broader patterns and relationships (global information) in the dataset.

DenseNet

DenseNet's architecture is defined by a dense connectivity pattern. This design has complex relationships between layers as the architecture is said to be linked with feature reuse which refers to the sharing and reuse of information learned in one layer of the network by succeeding levels. The densely connected layers promote a direct flow of information between all layers which in contrast to typical systems, information is passed sequentially from one layer to the next as it supports direct connections. This improves information flow and communication between layers and improves the robust propagation of features. This means that features or patterns detected in one region of the network can instantly spread throughout the network, improving the model's ability to capture complicated patterns. This architecture is well-known for addressing gradient flow difficulties. As Gradients in typical deep networks may drop as they propagate backward during training, the design of DenseNet assists in reducing this issue by ensuring that gradients travel more effectively through the network during backpropagation.

VGG16

VGG16 is well-known for its layer architectural simplicity and consistency. This indicates that its design is simple and sticks to a consistent pattern across levels. VGG16's architecture is distinguished by recurrent 3x3 convolutional layers. This specific design choice contributes to the previously noted simplicity and consistency. The basic design of VGG16 is well-known for its ability to extract hierarchical features. Here the term "hierarchical" refers to a systematic arrangement of characteristics at several levels of abstraction which shows the effectiveness of capturing features at different scales thus contributing to a rich data representation. The study of this architecture is driven by its demonstrated success in feature extraction and the goal is to see how well this architecture reveals hierarchical patterns within a heterogeneous image dataset.

ResNet50

The introduction of residual connections is a fundamental aspect of ResNet50. These links are added to solve the difficulties that come with training deeper networks. This addition of residual connections facilitates gradient flow during backpropagation and is critical for avoiding the vanishing gradient problem which is common when training deep neural networks. This issue happens when gradients get too narrow during backpropagation thus preventing deep network training. Now, these connections allow for more efficient gradient flow over the network and one advantage of including residual connections is that they allow for the training of much deeper networks which is a significant step forward as training deep networks without such connections can be difficult due to gradient-related difficulties. This thesis research tries to capture detailed elements inside the image domain by using the architecture ResNet50. And the emphasis is put on utilizing ResNet50's capacity to learn complex hierarchical representations particularly when dealing with detailed patterns in the dataset.

EfficientNetB0

Compound scaling is a crucial feature of EfficientNetB0 as this is a way of scaling many model dimensions at the same time, such as depth, width, and resolution. This method seeks to identify a model configuration that is both balanced and optimal. It is intended to be a compromise between model size and performance which means that it strives for high performance while keeping computational resources in mind with the goal to maximize the efficiency of the model. Another important aspect is the architecture's adaptation to varied scales that makes it an appealing alternative for the task at hand especially in cases where computational resources are required. And evidently, this model's versatility and ability to balance model size and performance surely make it different from other models.

XceptionNet

One of the crucial aspects of XceptionNet is the usage of depth wise separable convolutions . This method divides the spatial and depth dimensions of convolutions resulting in more efficient feature extraction. And moreover, the use of depth wise separable convolutions in the architecture is designed to improve the its capacity to extract features effectively which encourages the development of improved feature hierarchies and thus capturing more subtle information from the data. With that in mind, this research aims to analyze XceptionNet in order to assess its capability for detecting nuanced patterns in a diverse malware image dataset.

2.2.2 eXplainable Artificial Intelligence (XAI)

The goal of Explainable Artificial Intelligence (XAI) is to provide a visible and comprehensible framework for artificial intelligence (AI) systems' decision-making processes. LIME or Local Interpretable Model-Agnostic Explanations, has been chosen in this work to gain a deeper understanding of the image features that influenced the predictions generated by our neural network models. It makes the examination of complicated models easier by providing insights into the decision-making process. LIME is a powerful technique in XAI that aims to shed light on the decision-making processes of complex machine learning models. In the context of the study with neural network models, the evaluation of various image attributes' impact on the predictions generated by these models has been emphasized. For instance, an optimal model should showcase adaptability to diverse data types, flexibility to handle variations, and the capability to discern intricate patterns. Furthermore, resilience in addressing challenging scenarios where the presented data diverges from the training set is paramount. However, while assessing a model's quality, it extends beyond mere accuracy metrics thus the focus lies in its ability to generalize novel scenarios, facilitate comprehension of underlying processes, and contribute to resolving practical real-world issues. Therefore, to harvest the benefits of such a dynamic model eXplainable Artificial Intelligence (XAI) has been implemented in this research.

Chapter 3

Dataset Description and Preprocessing

3.1 Dataset Description

3.1.1 MalImg Dataset

The MalImg dataset contains 9339 malware byteplot images from 25 families. In this dataset the malware binary is interpreted as a sequence of 8-bit unsigned integers and then structured into a two-dimensional array. This array can be represented as a grayscale image where values range from 0 to 255 (0 represents black, while 255 represents white). The image's width remains constant, while its height may change based on the file's size (see Figure 3.1). Table 1 provides suggested image widths for various file sizes, derived from practical observations.

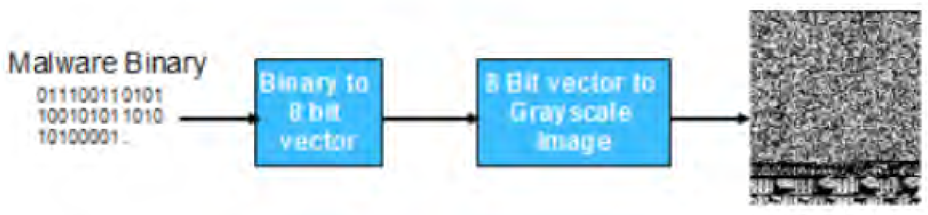


Figure 3.1: Visualizing Malware as an Image

The given figure depicts the image visualisation of a known Trojan called Donotova.A. Here, the executable code is stored within the .text section and examining

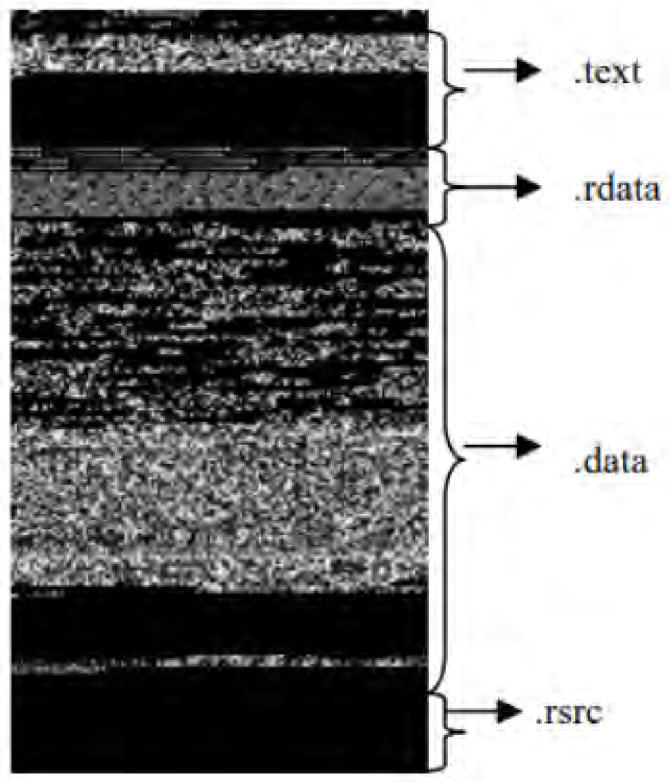


Figure 3.2: Visualizing Malware as an Image

the figure, it's evident that the initial segment of the .text section comprises finely detailed code texture. The remaining portion is filled with zeros (depicted as black), signifying empty space or zero padding at the section's conclusion.Then, the .data section encompasses both uninitialized code (depicted as a black patch) and initialized data, represented by a fine-grained texture.Lastly the .rsrc houses all the resources within the module, which might include icons utilized by an application. [18]

3.1.2 MaleVis Dataset

MaleVis dataset aims to offer an RGB based ground truth dataset so the analysis can be done based on the vision-based-multiclass malware recognition research. For this purpose, a collection of 26(25+1) class byte pictures are proposed. In the present scenario, one class represents "legitimate" samples whereas the remaining 25 classes represent various malware families. To be able to create the collection, firstly Sultanik's bin2png software was used to extract the binary images in three channel RGB color space from malware files. After the images were vertically lengthy, they were adjusted into two distinct square resolutions. (224*224 and 300*300 pixels). Moreover, the maleVis dataset comprises 5162 RGB images for validation and 9100 images for training. 350 picture samples are used in each of the training classes whereas the validation set includes a range of image counts. Since separating legitimate samples from malicious ones is the basis for malware classification and detection a somewhat larger amount of legitimate samples in the validation example (350vs1482) was included. [19]

3.1.3 Combined Dataset

After combining these two Datasets above mentioned we get a dataset of 45 malware families which is highly imbalanced.

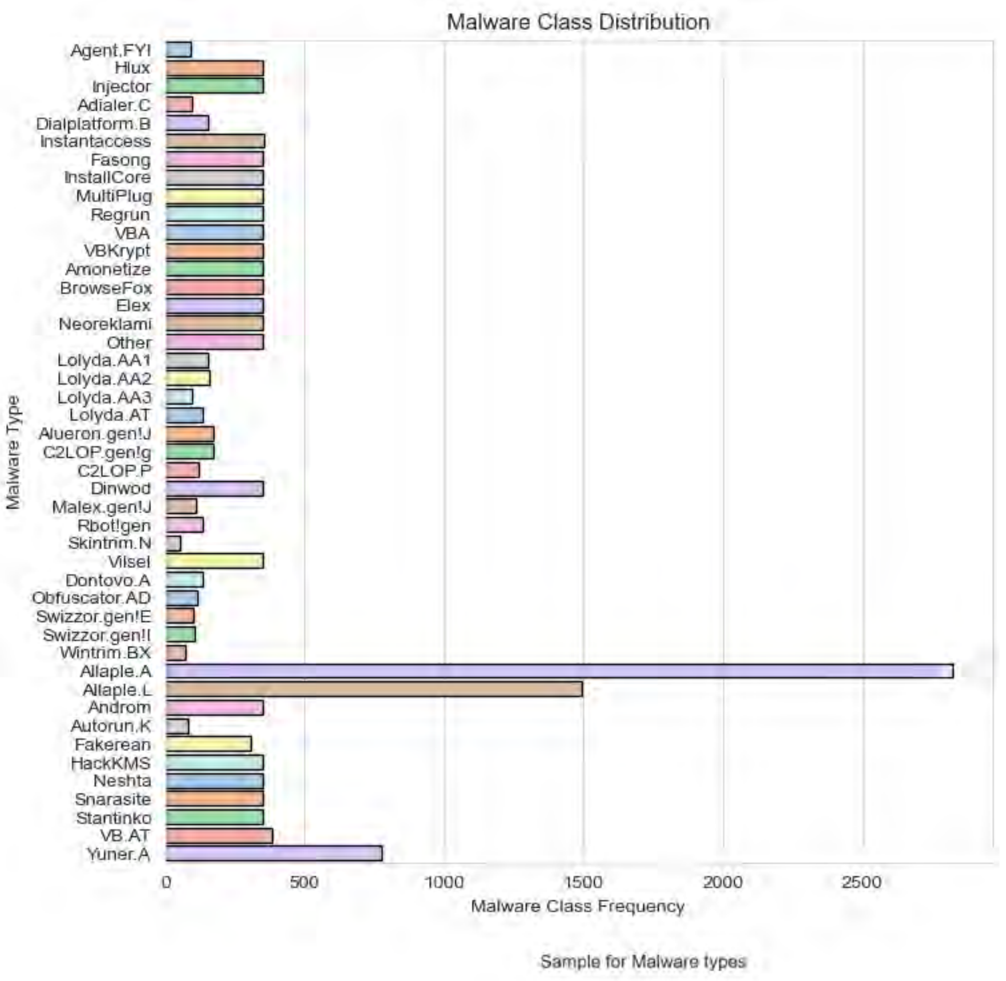


Figure 3.3: Combined Imbalance Dataset

Here, Skintrim.n has the lowest number of malware images at count 55 and with 2824 images Allaple.A is the largest one.

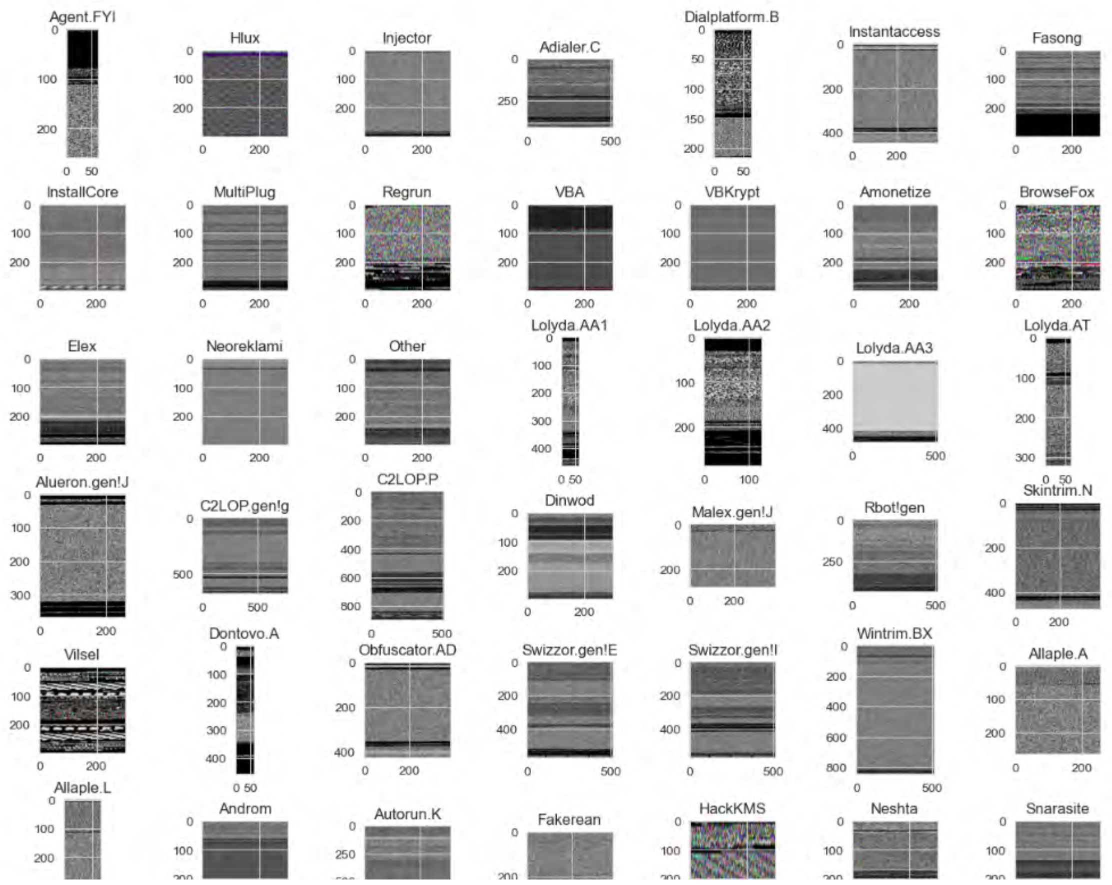


Figure 3.4: Malware Image Samples

3.2 Malware and Family Classification

In this section all the malwares and their designated families have been discussed extensively aith infestation process, dropped registry keys/processes and their casualties.

3.2.1 Adware/PUPs

A type of malware or unwanted software created to deliver targeted advertisements on infected computers. Adware can also collect information about the users and suggest them with relatable and customized ads. Trojans are one of the adware variants. Moreover, by taking advantage of the flaws that are downloaded or run by the system, a system can be infected by the malware operators with the help of adware. Side by side with that, the user might receive advertisements from adware once it is installed on the computer. The adware can also generate income as the adware operator is paid by the advertiser each time the ad is viewed. [20]

Amonetize.G is a type of adware that targets windows computers and tends to combine programs that are thought to be helpful with softwares that shows advertisements or needs to be paid for each installation. These software packages can also be called bundlers.

This Adware arrives on a system as a file dropped by other malware or as a file downloaded unknowingly by users when visiting malicious sites. [21]

Installation
This Adware adds the following processes:

```
%System%\DllHost.exe
/Processid:{F9717507-6651-4EDB-BFF7-AE615179BCCF}

%System%\svchost.exe -k netsvcs
```

This Adware adds the following registry keys:

```
HKEY_CLASSES_ROOT\foremilk.harbors.1
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\
foremilk.harbors.1\CLSID
HKEY_CLASSES_ROOT\foremilk.harbors
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\
foremilk.harbors\CurVer
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\
Wow6432Node\CLSID\{{GUID}}
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\
Wow6432Node\CLSID\{{GUID}}\
ProgID
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\
Wow6432Node\CLSID\{{GUID}}\
Version

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\

Wow6432Node\Interface\{0CCC84B3-DBBF-4FB2-8049-C0D4E2B58E13}

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\

Wow6432Node\Interface\{0CCC84B3-DBBF-4FB2-8049-C0D4E2B58E13}\
```

And it deletes the following registry keys:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\

Wow6432Node\CLSID\{{GUID}}\

LocalServer32\ServerExecutable
```

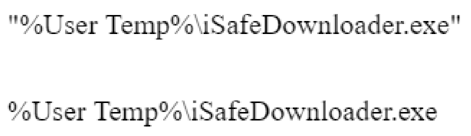
Browsefox.NHUSFDG is a type of adware family that uses various ways of browser hijacking and monetization. This appears on a computer either as a file deposited by previously installed malware or as a file obtained without the users' knowledge during visits to malicious websites.

This Adware deletes the following files:

%System%bigApp.exe [22]

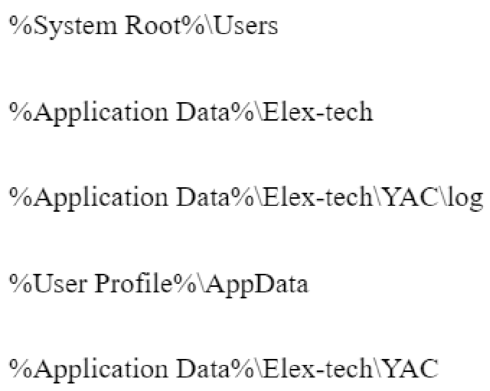
Elex.J is a type of adware that is retrieved from the internet and commonly functions as a tool to detect and eliminate adware. Additionally, it possesses the capability to deploy elex. It infiltrates a system either as a file placed by existing malware or as a file acquired inadvertently by users while visiting malicious websites. [23]

This Adware adds the following processes upon activation:



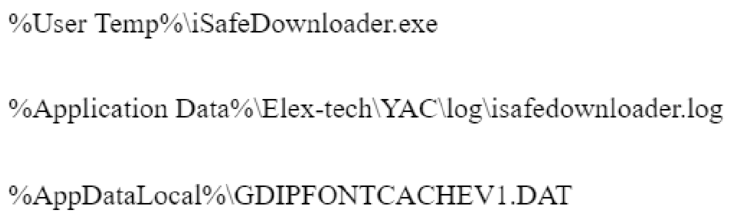
```
"%User Temp%\iSafeDownloader.exe"

%User Temp%\iSafeDownloader.exe
```

It creates the following folders:



```
%System Root%\Users

%Application Data%\Elex-tech

%Application Data%\Elex-tech\YAC\log

%User Profile%\AppData

%Application Data%\Elex-tech\YAC
```

This Adware drops the following files:



```
%User Temp%\iSafeDownloader.exe

%Application Data%\Elex-tech\YAC\log\isafedownloader.log

%AppDataLocal%\GDIPFONTCACHEV1.DAT
```

Neoreklami is a type of adware having various methods of showing advertisements on affected windows computers.

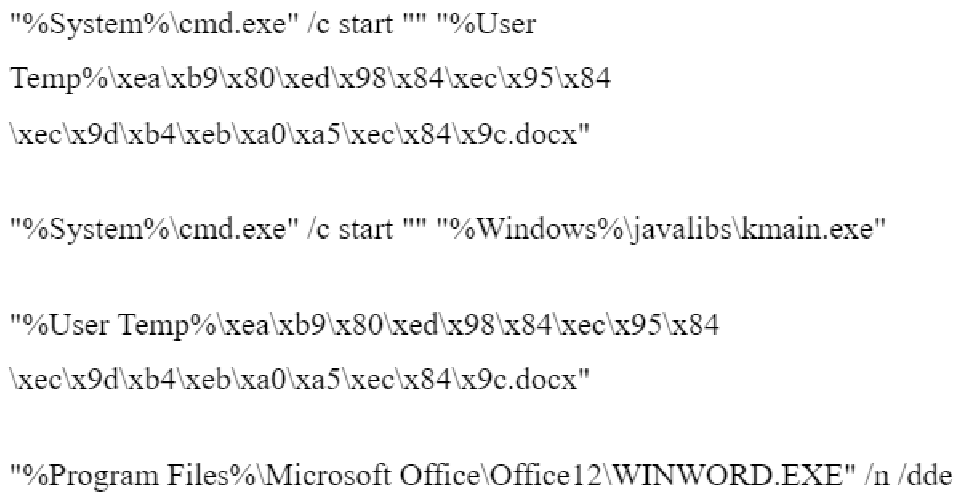
3.2.2 Worms

The name, worms, depicts a type of malware whose function is to copy itself and spread to other computers while continuing to function on systems that are compromised. A computer worm basically replicates in order to infect other computers. A worm normally goes unnoticed or undetected until its uncontrolled development uses system resources and leads other programs to lag. The worm is likely to take advantage of the holes that are present in the protection software of the computer to acquire confidential data, create vulnerabilities that help to have access to the system, corrupt files and perform more malicious activities. For which, servers, networks and individual systems tend to malfunction. However, a worm is distinct from a virus, as it can function independently of a host computer, whereas a virus is unable to operate without one. [24]

Androm is a kind of malware that accesses a system in a file that is dropped by other types of malware or by the way people accidentally download whilst browsing malicious softwares. This malware retrieves data from its binary picture as well as extracts data from the memory of the computer. On the computer, everything that is being typed, clicked or run passes via the memory. So these types of data could be read by a malicious software due to this vulnerability. Malware like this also tends to hide network activity. [25]

Installation

This malware adds the following processes



```
"%System%\cmd.exe" /c start "" "%User
Temp%\xea\xb9\x80\xed\x98\x84\xec\x95\x84
\xec\x9d\xb4\xeb\xa0\xa5\xec\x84\x9c.docx"

"%System%\cmd.exe" /c start "" "%Windows%\javalibs\kmain.exe"

"%User Temp%\xea\xb9\x80\xed\x98\x84\xec\x95\x84
\xec\x9d\xb4\xeb\xa0\xa5\xec\x84\x9c.docx"

"%Program Files%\Microsoft Office\Office12\WINWORD.EXE" /n /dde
```

```
%User Temp%\xea\xb9\x80\xed\x98\x84\xec\x95\x84
\xec\x9d\xb4\xeb\xa0\xa5\xec\x84\x9c.docx

%Windows%\javalibs\kmain.exe

%Windows%\splwow64.exe 12288
```

It creates the following folders:

- %Windowslibex [26]

Other System Modifications

This malware modifies the following file(s):

```
%Application Data%\Microsoft\Office\Recent\index.dat

%Application Data%\Microsoft\Office\Word12.pip
```

It adds the following registry keys:

```
HKEY_CURRENT_USER\Software\Microsoft\

Windows\CurrentVersion\Explorer\

FileExts\DDECache
```

HKEY_CURRENT_USER\Software\Microsoft\

Windows\CurrentVersion\Explorer\

FileExts\DDECache\WinWord\

System

Dropping Routine

This malware drops the following files:

%Application Data%\Microsoft\Office\Recent\Temp.LNK

%User Temp%\xea\xb9\x80\xed\x98\x84\xec\x95\x84
\xec\x9d\xb4\xeb\xa0\xa5\xec\x84\x9c.docx

%Application Data%\Microsoft\UProof\CUSTOM.DIC

%Application
Data%\Microsoft\Office\Recent\xea\xb9\x80\xed\x98\x84\xec\x95\x84
\xec\x9d\xb4\xeb\xa0\xa5\xec\x84\x9c.docx.LNK

%Application Data%\Microsoft\Office\Recent\index.dat

%Application Data%\Microsoft\Templates\Normal.dotm

%All Users Profile%\Microsoft\OFFICE\DATA\opa12.dat

%Application Data%\Microsoft\Office\Word12.pip

%Windows%\libex\dlmgr.dll

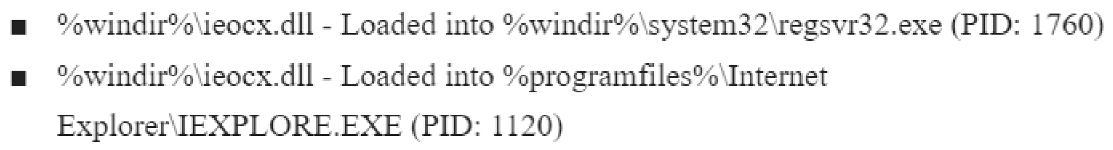
Fakerean.gen!A is a type of malware family of deceptive security apps that pretends to be malware detection tools and often finds various sorts of viruses on the computer. And even before cleaning the computer completely, it will suggest the users to pay for it. That being said, the program actually is not a malware scanner or anti virus tool and has no indication of finding any malware at all. Rather all it takes is a cursory glance to pay the programmers. Moreover, the program won't do anything even if it is paid to "unlock" it because the given computer is not really affected with the malware that it found. [27]

Installation-

During installation, the malware creates the following file:

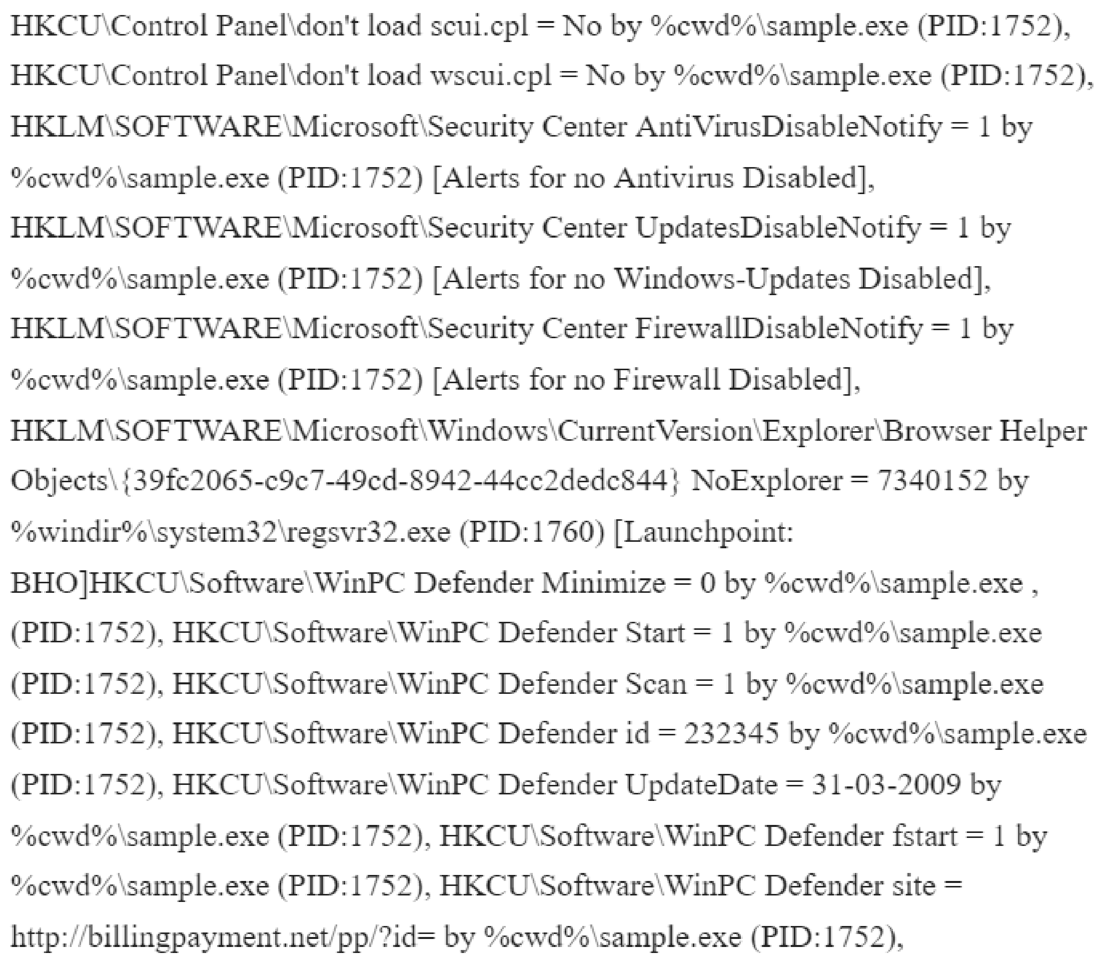
- %windirieocx.dll [28]

Where %windir% represents the Windows Directory.The following modules are then loaded into other processes:



- %windir%\ieocx.dll - Loaded into %windir%\system32\regsvr32.exe (PID: 1760)
- %windir%\ieocx.dll - Loaded into %programfiles%\Internet Explorer\IEXPLORE.EXE (PID: 1120)

Registry Modifications - Sets these values:



```
HKCU\Control Panel\don't load scui.cpl = No by %cwd%\sample.exe (PID:1752),
HKCU\Control Panel\don't load wscui.cpl = No by %cwd%\sample.exe (PID:1752),
HKLM\SOFTWARE\Microsoft\Security Center AntiVirusDisableNotify = 1 by
%cwd%\sample.exe (PID:1752) [Alerts for no Antivirus Disabled],
HKLM\SOFTWARE\Microsoft\Security Center UpdatesDisableNotify = 1 by
%cwd%\sample.exe (PID:1752) [Alerts for no Windows-Updates Disabled],
HKLM\SOFTWARE\Microsoft\Security Center FirewallDisableNotify = 1 by
%cwd%\sample.exe (PID:1752) [Alerts for no Firewall Disabled],
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper
Objects\{39fc2065-c9c7-49cd-8942-44cc2dedc844} NoExplorer = 7340152 by
%windir%\system32\regsvr32.exe (PID:1760) [Launchpoint:
BHO]HKCU\Software\WinPC Defender Minimize = 0 by %cwd%\sample.exe ,
(PID:1752), HKCU\Software\WinPC Defender Start = 1 by %cwd%\sample.exe
(PID:1752), HKCU\Software\WinPC Defender Scan = 1 by %cwd%\sample.exe
(PID:1752), HKCU\Software\WinPC Defender id = 232345 by %cwd%\sample.exe
(PID:1752), HKCU\Software\WinPC Defender UpdateDate = 31-03-2009 by
%cwd%\sample.exe (PID:1752), HKCU\Software\WinPC Defender fstart = 1 by
%cwd%\sample.exe (PID:1752), HKCU\Software\WinPC Defender site =
http://billingpayment.net/pp/?id= by %cwd%\sample.exe (PID:1752),
HKLM\System\CurrentControlSet\Services\BITS\Control ActiveService = BITS by
%windir%\system32\services.exe (PID:604),
HKLM\System\CurrentControlSet\Services\BITS Start = 12 by
%windir%\system32\services.exe (PID:604),
HKLM\Software\Classes\CLSID\{95dd14b6-a2ed-11da-9241-806d6172696f}\\{95d
d14b9-a2ed-11da-9241-806d6172696f}\\{95dd14b9-a2ed-11da-9241-806d6172696f
}\ BaseClass = Drive by %cwd%\sample.exe (PID:1752),
HKLM\Software\Classes\batfile\MUICache\ C:\Documents and
Settings\user\Application Data\asd.bat = asd by %cwd%\sample.exe (PID:1752),
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\{39FC2065-C9C7-49CD-8
942-44CC2DEDC844}\iexplore Type = 655360 by %programfiles%\Internet
Explorer\IEXPLORE.EXE (PID:1120),
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\{39FC2065-C9C7-49CD-8
942-44CC2DEDC844}\iexplore Count = 12 by %programfiles%\Internet
Explorer\IEXPLORE.EXE (PID:1120)
```

Neshta.A is a malware that tends to spread by adding the malicious codes to other executable files. [29]

When its executed, the virus tends to create the following files:

```
%temp%\tmp5023.tmp
%windir%\directx.sys
%windir%\svchost.com (41472 B, Win32/Neshta.A)
```

The following Registry entry is set:

```
[HKEY_CLASSES_ROOT\exefile\shell\open\command]
        "(Default)" = "%windir%\svchost.com "%1" %*"
```

This causes the malware to be executed along with any program.

The worm infiltrates files by adding its code at the start of the original program. The inserted code size measures 41472 B. Additionally, it spreads to files located on network and removable drives.
It steers clear of infecting files that contain certain specific strings within their paths.:
%temp%
%windir%
`\PROGRA~1\`

When selecting a file to infect, various other conditions are considered. Upon execution of an infected file, the initial program is copied into a temporary file and then executed.

The original file is stored in the following location:

`%\temp%\3582-490\%\filename%`

Snarasite is a type of malware that does actions without the user's knowledge. Some examples of these actions are downloading and uploading files, gathering system data, introducing other malware into the machine that was compromised, executing Dos attacks and many others.

How this malware can infect the PC -
Spam emails, malicious office macros, infected removal drives, hacked or compromised web pages.

Stantiko malware family is known for targeting Windows operating systems. Stantiko can be used to execute certain actions like filling out forms, searches etc. This malware is being used on hacked web servers and users' PCs worldwide. Moreover, the bots operate as a proxy agent for the virus which is activated through them. [31]

Upon activation, the malicious software will authenticate the configuration file present at `/etc/pd.d/proxy.conf`.

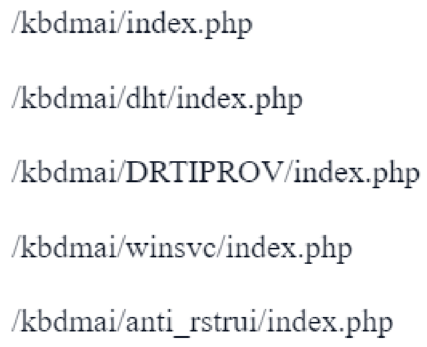
Upon execution, the malware validates the configuration file present at `/etc/pd.d/proxy.conf`. After confirming the existence of the configuration file, a function responsible for parsing the file, LoadConfigFromFile(), is invoked. This particular function, ParseConfigElement(), reveals a portion of the configuration file's structure, which should contain keys like proxy_ip, port, redirect_url, localhost, ip_header, and request_header_log_files.

Following the configuration's loading into the system's memory, the daemon commences its operations, as depicted in Figure 6 through the start_demon() function. This involves the creation of a socket and a listener for accepting new client connections. Upon receipt of a new client, the listen_socket() function is triggered. Subsequently, the client generates a new thread, executing the code within the on_client_connect function. Initially, this function checks the request method (GET, POST, or NOTIFY). In the case of a GET request, the program generates a 301 redirect HTTP response that incorporates the redirect_url parameter from the configuration file. This behavior implies that querying a C&C IP address via a web browser could result in the request being redirected to an arbitrary website, without any traces or malicious artifacts.

If the request method happens to be a POST or NOTIFY, the malicious software will generate a POST request to transmit to the C&C server. This request will be constructed based on the client's HTTP request header and content, utilizing the create_post_data() function.

Afterward, the malicious software proceeds to execute the `mysql_server_do_request()` function, which is tasked with transmitting the POST request to the C&C server. .

The POST request is sent to one of the following paths on the C&C server:



```
/kbdmai/index.php

/kbdmai/dht/index.php

/kbdmai/DRTIPROV/index.php

/kbdmai/winsvc/index.php

/kbdmai/anti_rstrui/index.php
```

The `detect_proxy_script()` function determines the C&C server's path by utilizing data provided by the client. The C&C server's IP address is a parameter extracted from the configuration file.[32]

HackKMS is a type of malware that is created to activate or operate unregistered Microsoft software. Moreover, these programs can be used with other malicious softwares. [33]

3.2.3 Backdoor

Trojan horses come in many faces and attributes that differentiates them amongst each other. A backdoor trojan is one of them and while they are very basic sort of malware they are potentially dangerous. As the name suggests, a backdoor trojan can load varieties of malware into the device while acting as a door or gateway for them. It is often the malware of choice to set up botnets which are essentially a network of computers infected by backdoors acting as per one malignant source. Thus, without our knowledge our computer partakes in the operations of a zombie network. [34]

Hlux is a type of backdoor malware that tries to connect to a dead IP:Port two times and delays analysis tasks along with altering proxy settings. It ciphers information on the victim's harddrive to prevent access. [35]

Injector is another type of trojan that 'injects' malicious code into processes that are already running on the computer and downloads external malware, disrupts web browsing and spies on the user's actions.

Families flagged by 'Injector' detections primarily exhibit this functionality although numerous malware families also utilize injection techniques to compromise a system. Injector malware commonly consists of Windows executable (EXE) and JavaScript files, but they may also arrive via spam emails, exploit kits, or as part of another malware's payload. The ramifications of code injection vary significantly among Injector trojans due to the diversity of families and some common actions include corrupting program data, providing unauthorized data access, inducing program crashes or denial of service, monitoring or manipulating web browser activity, overseeing or influencing user actions on the device, downloading additional programs or components onto the device, and enabling remote attackers to assume complete control over the compromised device. [36]

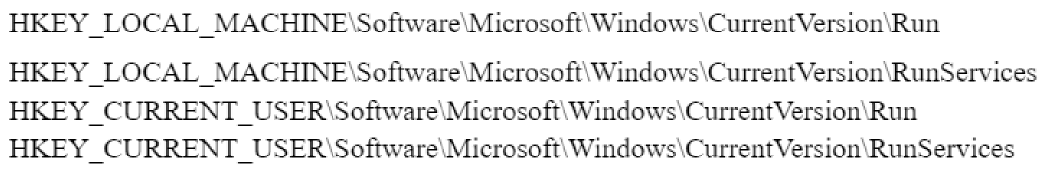
Agent is an extensive collection of software, primarily responsible for downloading and deploying adware or malware onto the target system. Variants within the Agent family might additionally alter settings related to Windows Explorer or the Windows interface. This umbrella term encompasses a diverse range of malware that doesn't neatly fit into established families. Within the Agent family, one can find trojans, worms, viruses, backdoors, and various other forms of malicious programs.
Agent.FYI is one such variant of the Agent family that exhibits similar behaviors with its peers. [37]

3.2.4 Trojan Horses

As the name suggests, the trojan horse malware mimics the historic incident of Greek Mythology. This virus impersonates a normal or legitimate file and downloads into the device. A more suitable explanation of what a Trojan Horse would be is that it's a form of malicious software that usually disguises itself as an attachment in an email or a freely available file. It then infiltrates the user's device. After being downloaded, the harmful code carries out specific tasks as intended by the attacker, like gaining unauthorized access to business systems, monitoring users' online behavior, or pilfering confidential information and so on. [38] Although it is a prominent type of malware, due to its attacking style it is not considered as a virus.

Rbot!gen is one such trojan that allows perpetrators to control the affected computers. Once a system is compromised, this trojan establishes a connection with a specific IRC server and joins a dedicated channel, offering a pathway for attackers to issue commands which enable the trojan to propagate by seeking out vulnerable network shares, exploiting weaknesses in Windows and subsequently spreading through accessible backdoor ports employed by diverse strains of malicious programs. Moreover, this trojan equips attackers with the capability to execute various backdoor functions, including launching denial-of-service attacks and extracting system information from compromised computers. When the malware runs it copies itself to **%windir%** or system folder.

The malware often includes a process of adding a value to one or multiple registry keys like:



```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices
```

The malware then proceeds to run every time the windows start.

Rbot!gen malware establishes a connection to an IRC server and enters a specific channel that allows it to receive various commands. These commands encompass a wide range of actions such as scanning for vulnerable computers within the network, inspecting network ports, fetching and running remote files, monitoring network activity, initiating HTTP/HTTPD, SOCKS4, and TFTP/FTP servers, managing the activation or deactivation of the DCOM protocol, retrieving comprehensive computer configuration data including Windows login details, user account specifics, accessible shares, file system details, and network connections. Additionally, the malware has the capability to log keystrokes, acquire game CD keys, capture screenshots and images from webcams, reroute TCP traffic, transfer files using FTP, dispatch emails, control processes and services, and execute denial-of-service (DoS) attacks. After receiving commands via IRC the malware propagates to remote computers by taking advantage of Windows vulnerabilities and the way the malware achieves this is by trying on weak passwords from a list. [39]

Vilsel is another member of the Trojan family and it is dropped by other malwares or gets downloaded into the device without the knowledge of the user. The malware, after gaining control of the user's computer clones itself under random names to the %temp%, %windir%, and %appdata% folders which are then joined to the list of programs that run as soon as the operating system is turned on. Additionally, the malware sends HTTP requests to the attackers server to download files that make changes to the window's registry such as deactivating Task Manager, Windows Firewall, Registry Editor etc. and adds its own registry entries for Windows services. [40]

Alueron.gen!J is a type of trojan that upon entering a network changes the DNS setting on routers. It gains access to the information of the network interfaces on the affected device and determines the IP address and the DHCP server of the said network. When the malware acquires an IP address, it proceeds to establish an HTTP connection, aiming to access either the router's default configuration page or any of the specified pages associated with typical models of network routers which serve as their web-based configuration interfaces. It does so by using one of the following pages:

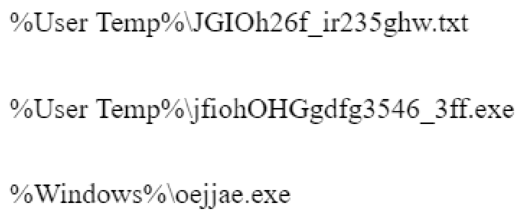
```
/index.asp
/dlink/hwiz.html
/home.asp
/wizard.htm
```

The malware first tries to authenticate itself on the router using commonly used or default login credentials and after gaining access, it tries to modify the router's DNS settings. To confirm its success, the malware then conducts a DNS query for the domain "infersearch.com" to verify that the returned IP address matches 69.50.190.107. Additionally, the malware sends data to the IP address 216.255.186.238 after attempting the aforementioned actions. [41]

Dinwod is one such virus that is dropped by other malwares and upon installation adds the following process:

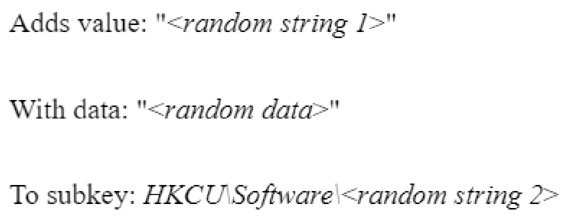
```
%User Temp%\jfiohOHGgdfg3546_3ff.exe /scookiestxt %User
                Temp%\JGIOh26f_ir235ghw.txt
```

also adds the following processes:



```
%User Temp%\JGIOh26f_ir235ghw.txt

%User Temp%\jfiohOHGgdfg3546_3ff.exe

%Windows%\oejjae.exe
```

C2LOP.P is one such trojan that manipulates web browser settings by adding bookmarks, bringing out pop up advertisements and so on. It enters a computer bundled up within other software files and launches "Internet Explorer", launches malignant code into the Internet Explorer operation and as like other malwares modifies the system registry.



Adds value: "*<random string 1>*"

With data: "*<random data>*"

To subkey: *HKCU\Software\<random string 2>*

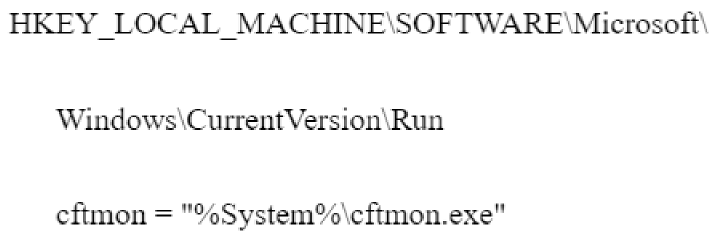
The malware connects to an offshore website in order to download and launch unspecified files and once they areb installed random pop up advertisements are launched, e.g. a specific sample of Win32/C2Lop.P attempts to connect to "ayb.host127-0-0-1.com" via TCP port 80 and download files. [7]

C2LOP.gen!g is another member of C2LOP.P family and exhibits similar kind of behaviour and installation habit.

Malex.gen!J is another trojan dropped by other malware and is installed once a malicious site is visited and copies itself as:

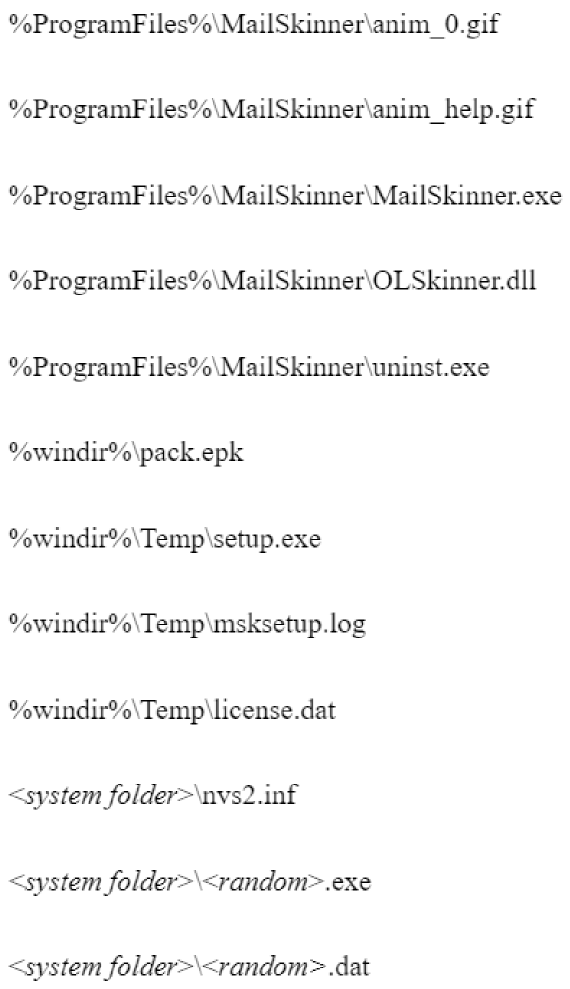
`%\ System%\ cftmon.exe`

It adds the following process so that it is run every time the system is run:



```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\

Windows\CurrentVersion\Run

cftmon = "%System%\cftmon.exe"
```

Skintrim.N is a variant of trojan that download and runs random files that include on unnecessary updates and additional malwares from existing sites and ads. Oner of its way to present itself is as 'Microsoft Outlook' add ons for emojis in emails. Upon execution of the trojan's installer, it generates the subsequent files: When the installer for this trojan is run, it creates the following files:



%ProgramFiles%\MailSkinner\anim_0.gif

%ProgramFiles%\MailSkinner\anim_help.gif

%ProgramFiles%\MailSkinner\MailSkinner.exe

%ProgramFiles%\MailSkinner\OLSkinner.dll

%ProgramFiles%\MailSkinner\uninst.exe

%windir%\pack.epk

%windir%\Temp\setup.exe

%windir%\Temp\msksetup.log

%windir%\Temp\license.dat

<*system folder*>\nvs2.inf

<*system folder*>\<*random*>.exe

<*system folder*>\<*random*>.dat

Moreover, skintrim initiates a mutex named "mymutsglwork" and proceeds to inject code into various executable files including `explorer.exe`, `WLMail.exe`, `Winmail.exe`, `Outlook.exe`, and `Thunderbird.exe`. Additionally, it intercepts specific APIs such as `send` from `ws2_32.dll` and `OpenProcess` from `kernel32.dll`. Furthermore, it establishes communication with the `updates.advert-network.com` server for the purpose of downloading a new component. [10]

3.2.5 Trojan Downloaders

A trojan-downloader belongs to the trojan category installing itself onto a system and patiently waiting for an Internet connection to connect to a remote server or website. Its primary function is to download additional programs, often malicious ones, onto the compromised computer. These trojans are frequently delivered as part

of another malicious program's payload such as a trojan-dropper, which stealthily places and installs the trojan-downloader onto a device. These are also commonly distributed through deceptive file attachments in spam emails. These attached files typically bear names that sound legitimate, like 'invoice' or 'accounts.exe', exploiting social engineering tactics and opening such file attachments leads to the installation of the trojan-downloader. [48]
One such Trojan downloader is **Dontovo.A.** Upon execution, the trojan initiates a copy of %Windows% .exe and injects code into it, subsequently deleting its executable file. Similar to most downloaders, it is commonly dropped by other malware and executes arbitrary files by injecting code into the svchost.exe process. Following this, the trojan contacts the domain 'iframr.com' to retrieve configuration data, potentially leading to further downloadable locations. Additionally, it has been observed to establish connections with 'videofx4you1.com'. Finally, the downloaded files are stored in the % temp % directory and executed.

Malware obfuscation refers to the practice of making a program's code challenging to uncover or comprehend, aimed at concealing its functionality from both humans and computer systems while maintaining its original operational behavior. The objective extends beyond mere code obfuscation, striving to completely conceal the program's existence. Moreover, threat actors commonly employ compression, encryption, and encoding techniques as primary obfuscation methods. Often, multiple methods are combined to thwart a broader range of cybersecurity defenses during the initial intrusion phase.
The ways this process works are:

- **Binary Padding**: Random code is generated through a function and saved as binary to exceed the maximum file size limit, typically 25 - 200 MB of malware scanners. This tactic creates a situation where scanning will take time and risks client timeout.

- **Software Packing**: UPX, a rather famous tool is used to compress malicious payload into an executable file that changes the payload/s size and signature. Moreover, it ensures that any efforts to reverse engineer the code are more challenging and the executable file might additionally undergo encryption to add further complexity and impede attempts to deobfuscate it.

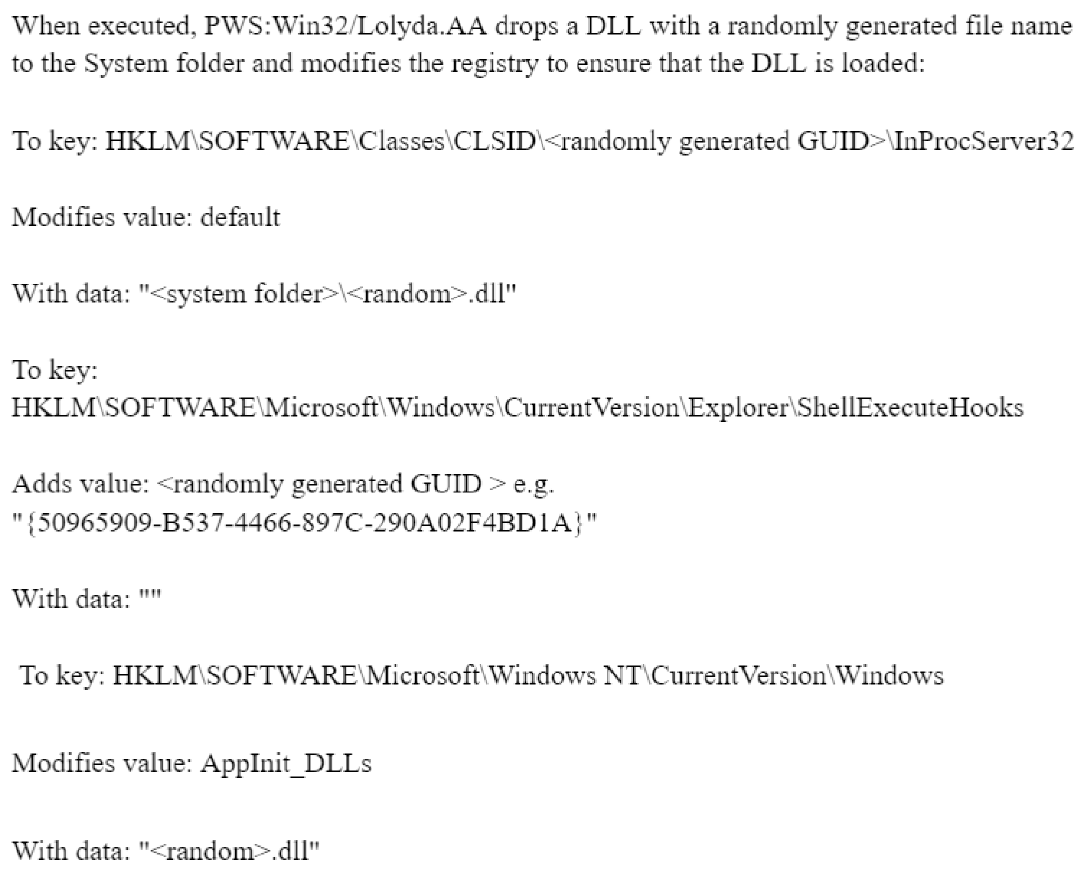
- **Compile after Delivery**: A ransomware piece arrives in an uncompiled form or as source code via spam email. Upon triggering by the user, it invokes a native compiler like `csc.exe` to compile its payload directly on the device, bypassing perimeter defenses like firewalls. Subsequently, the ransomware encrypts all files stored on the victim's hard drive. [49]

Obfuscator.AD is one such malware obfuscation malware variant and little is known about its functionality,threat and attack pattern. Swizzor.gen!E is one type of trojan downloader that infiltrates a system either through a file dropped by other malware or via files unknowingly downloaded while visiting malicious websites. Upon installation, it establishes specific mutexes, Global{random and Local{random, to ensure that only one of its instances operates at a given time. Additionally, it injects threads into normal processes like `IEXPLORE.EXE`. In terms of its adware behavior, this Trojan connects to specific URLs, such as `ads.BLOCKEDs-local.com`, to download and display advertisements. [50]

Swizzor.gen!I and **Wintrim.VX** other such trojan downloaders and are variants of the Swizzor family. Them, like their peers, download malignant code and files without their knowledge and execute them.

3.2.6 Password Stealers(PWS)

Lolyda is a malware known as "AA" obtains private data associated with well-known online games and transmits it to a remote attacker. Furthermore, it has the ability to download and run any file.



When executed, PWS:Win32/Lolyda.AA drops a DLL with a randomly generated file name to the System folder and modifies the registry to ensure that the DLL is loaded:

To key: HKLM\SOFTWARE\Classes\CLSID\<randomly generated GUID>\InProcServer32

Modifies value: default

With data: "<system folder>\<random>.dll"

To key:
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ShellExecuteHooks

Adds value: <randomly generated GUID > e.g.
"{50965909-B537-4466-897C-290A02F4BD1A}"

With data: ""

To key: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows

Modifies value: AppInit_DLLs

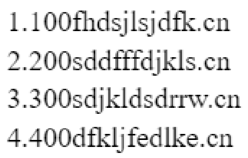
With data: "<random>.dll"

Lolyda.AA tries to scan the active process memory of various well-known online games with the aim of locating specific details, including the following:

- Username
- Password
- Server Address
- Character Information

This information is posted to a remote server. It also downloads and Executes Arbitrary Files.

Lolyda.AA may download and execute more files upon installation. The files are usually downloaded to the %TEMP % folder. The URLs used for these downloads are variable, for example, Lolyda.AA has been observed attempting to contact some pre-specified domains.



1.100fhdsjlsjdfk.cn
2.200sddfffdjkls.cn
3.300sdjkldsdrrw.cn
4.400dfkljfedlke.cn

Lolyda.AT attempts to search the running process memory of several online games to find particular information, such as the following:

- User name
- Password
- Server address
- Character information

The gathered data is subsequently forwarded to a distant server. Captures screen images The malware periodically checks the active Window title for specific text strings such as "ACDSee" or "Internet Explorer." If these strings are identified, it captures a screenshot, saving it as a JPEG file in the Windows Temporary Files directory, and then transmits it to a remote server. This process is intended to steal the "password protector" picture file commonly used in online games. Additionally, PWS:Win32/Lolyda attempts to terminate processes identified by their MD5 hash values listed in a predefined roster.[52]

3.2.7 Dialer Viruses

A Dialer Virus is a program that utilizes a computer's modem to establish a dial-up connection through the internet, usually to generate revenue from the calls made. Most malicious dialers operate similarly to existing computer viruses. **Adialer.C**, categorized as a Trojan Dialer, impacts computers equipped with modems connected via phone lines. Another Trojan dialer, **Dialplatform.B**, poses a similar threat, potentially resulting in unexpected and considerable telephone expenses. Another prime example of a dialer is **Instanccess**[53]

3.2.8 Others

Despite being such a diverse categorization of malware, there are still malwares in the wild that struggle to fit in with known families and **Fasong** is one such malware. Though much information could not be found about this malware but seems to work similar to worms only it does not actually fit with the worm family as well.
InstallCore is the type of malware that installs more than one application to the user's computer behind a known application combined with an adware. They are often served as updates for plug-ins like Java or Flash along with adware and PUPs. [54]
Multiplug utilizes a diverse range of techniques to distribute its advertisements.

It employs randomized names for files and folders and is typically installed through bundling processes. [55]
Regrun operates by reading information from its own binary image, establishing

itself to automatically run during Windows startup, engaging in network activities without leaving traces in API logs, attempting to modify browser security settings, displaying abnormal binary traits, encrypting the files on the victim's hard drive, rendering them unusable, and impeding regular access to the target's computer. These actions collectively contribute to its harmful effects on the target system and data security. [56]
Macro refers to a series of instructions that automate software to execute spe-

cific actions and threat actors exploit this functionality through converting them to Macro Malwares. This type of malware capitalizes on VBA (Visual Basic for Application) programming within Microsoft Office macros to disseminate various forms of malware. Typically, these are distributed through phishing emails, where the attacker entices the recipient to open an attached document and upon opening, a security warning appears that prompts the recipient to "Enable Content." Subsequently, the malicious macro runs, impacting the recipient's system. When the malicious macro is enabled, it typically triggers the execution of a base64 PowerShell code, leading to the download of a file either in the % UserProfile% or %Temp% directories. This downloaded file then runs shortly after its download.[57] **VBKrypt** is a diverse family of malware coded in Visual Basic. Depending on the specific variant, the malware may drop files, make changes to the registry, and perform other unauthorized actions on the affected computer system. [58]

3.2.9 Final Dataset

Based on this theoretical approach, the final dataset is made with 45 malware, which are then grouped into 8 classes each family with 350 samples.

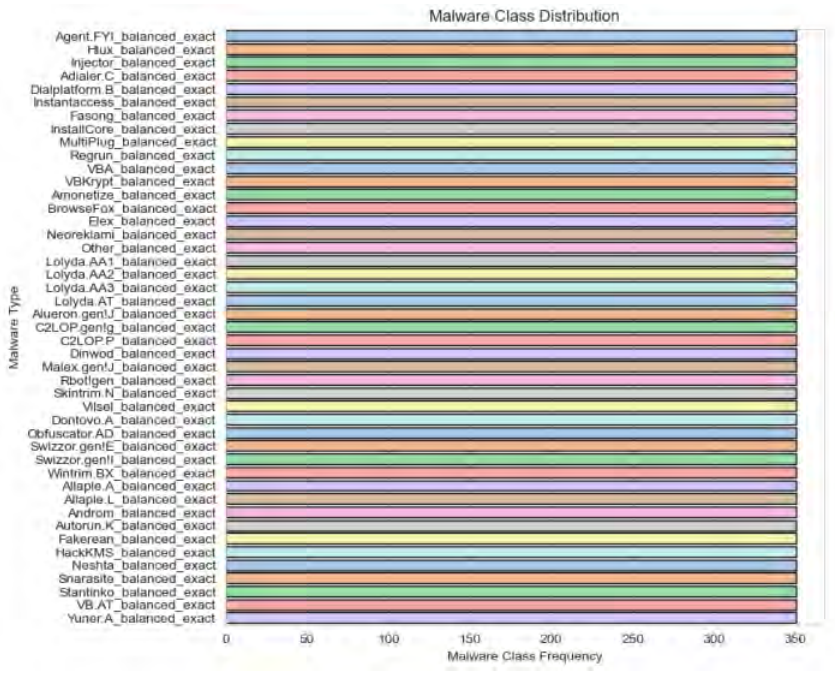


Figure 3.5: Balanced Dataset each having 350 samples

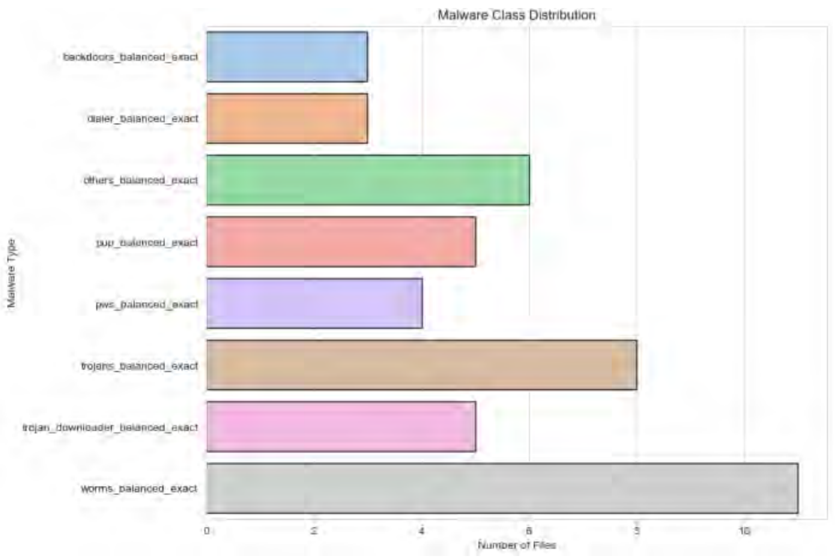


Figure 3.6: Malware Family Categorization

3.2.10 Data Pre-Processing

In this thesis research, the aim to provide a complete approach to preparing a balanced image dataset for implementing Convolutional Neural Networks (CNNs) in the domain of malware classification. Primarily, the goal is to optimize the dataset for optimal CNN training using the ImageProcessor class. This class is known for simplifying preprocessing operations that are critical for preparing images for model training later on.
Initially, 45 different kind of malware from both datasets were grouped into 8 classes of malware families. Malwares belonging from the same family exhibiting similar infiltration and execution patterns are grouped together.
Following dataset preparation, TensorFlow's ImageDataGenerator has been used to implement real-time data augmentation which entails randomly selecting images and applying various adjustments such as rotation, shifting, shearing, and flipping. Real-time data augmentation considerably improves the training dataset by diversifying the images supplied to the algorithm, allowing it to handle a broader range of instances.

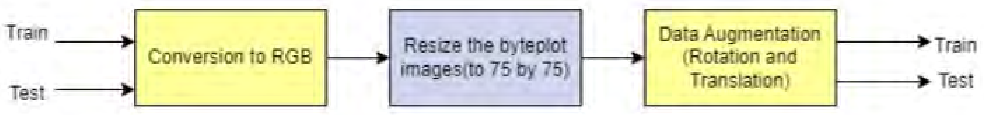

Train
Test
Conversion to RGB
Resize the byteplot images(to 75 by 75)
Data Augmentation (Rotation and Translation)
Train
Test

Figure 3.7: Pre processing dataset flow chart

During the training, and testing phases, the outcomes of the data augmentation process appear as generators, specialized structures, or functions that produce batches of images. These generators are important in two ways. First, during training, they provide the model with a constant stream of different images, assisting the model in learning to generalize successfully. Second, they are tailored to each step, ensuring that the model experiences a diverse set of data under a variety of conditions.

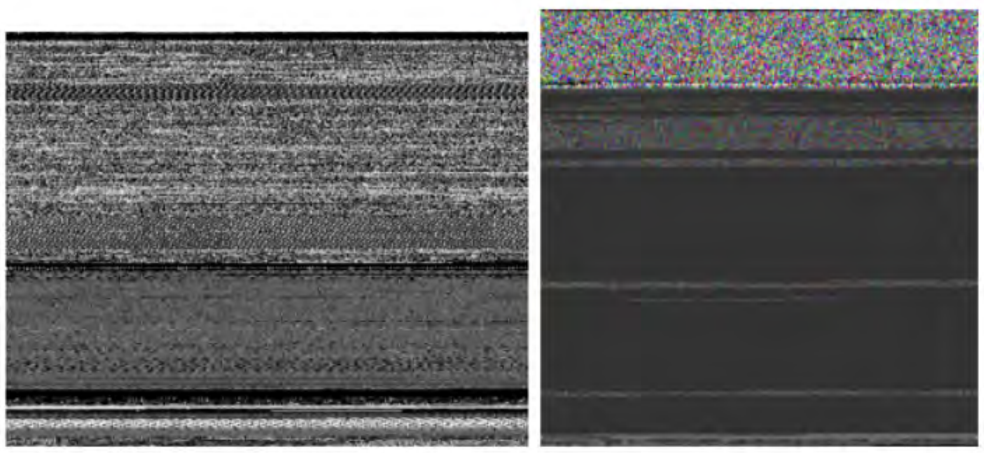


Figure 3.8: Samples

After separating the data into training and testing sets, all photos are converted to RGB and resized to 75 by 75 dimensions. Furthermore, rotation and translation are applied to images, promoting rotation and translation invariance in the model. Figure 3 represents the preprocessing technique.

This combination of the ImageProcessor class and TensorFlow's ImageDataGenerator not only increases the model's adaptability but also its performance. By subjecting the model to a wide variety of image variations, it becomes more robust, resulting in more accurate malware classification predictions.

Chapter 4

Proposed Model And Experimentation

4.1 Overall Approach

Initially, we had two datasets Malimg and Malevis, where one has imbalanced class distribution and the other has equally balanced class distribution. Since we aim to create a balanced dataset, we first calculate how many files must be added or removed to reach the target file number. After that, by augmenting randomly selecting images either it copies files to reach the target number or it randomly selects images and removes them to reduce the number and reach the target file number. After balancing the dataset, we proceed to Image Processing where all the files are reduced to a general ratio of 75*75, and since the images were in grayscale so for better clarification, they are turned into RGB. We have introduced TensorFlow's Keras API and data augmentation for the processing and generation of image data for training, validation, and testing in machine learning models, especially those involving CNNs. In the Domain Adaptation process, it fine-tunes a pre-trained neural network model for domain adaptation, including which layers to fine-tune, adding a custom final layer, compiling the model, training with early stopping, and so on. We have also applied regularization techniques to prevent overfitting so that it doesn't perform well on only training data, but on unseen data as well. Later on, we experimented with six Convolutional Neural Network (CNN) state-of-the-art architectures on our dataset using Domain Adaptation. Then we implemented XAI to understand model behavior or to know which key areas influence the prediction of the models. We have applied LIME to the test dataset for all six models, and since LIME required a segmentation method to divide the image into superpixels, we used SLIC segmentation to get a better understanding of the model's behavior by highlighting key image features that influence its predictions

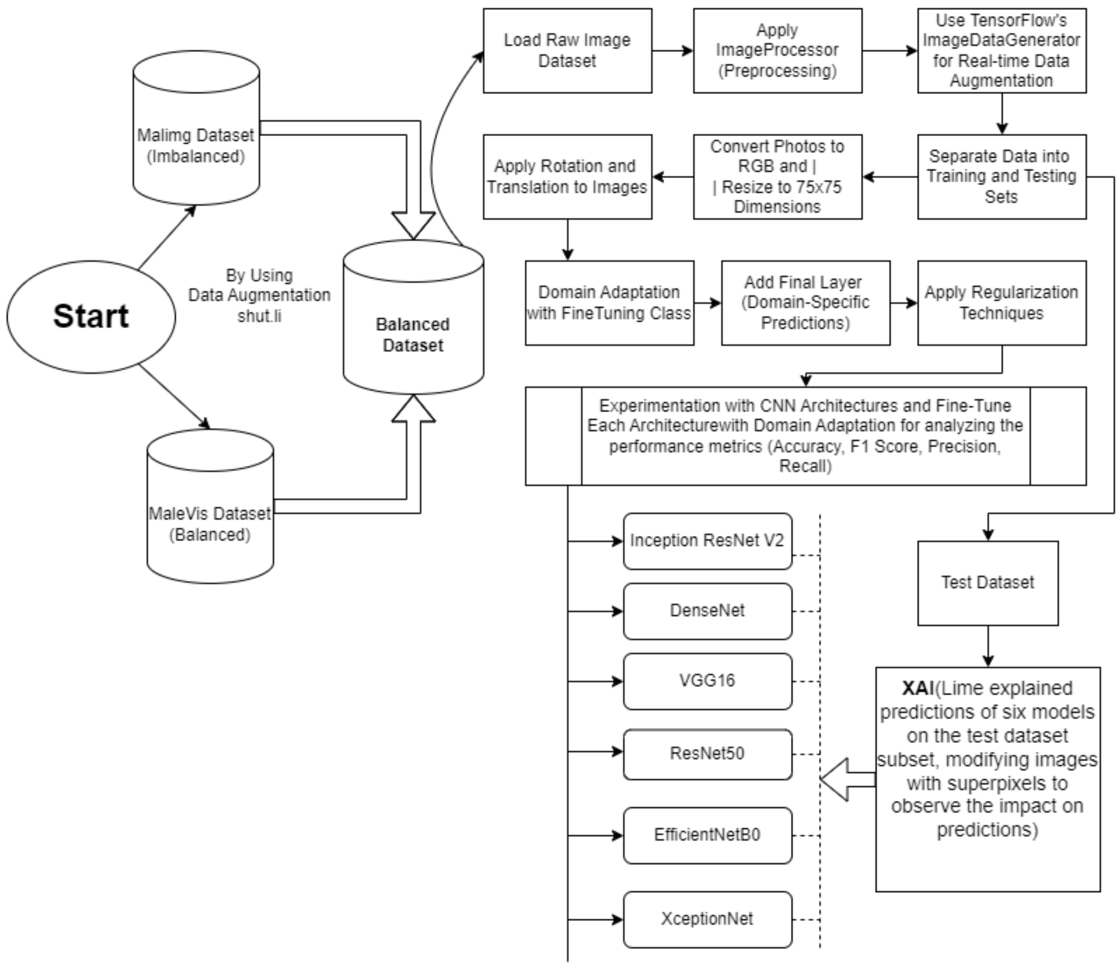

Start
Malimg Dataset (Imbalanced)
MaleVis Dataset (Balanced)
By Using Data Augmentation shut.li
Balanced Dataset
Load Raw Image Dataset
Apply ImageProcessor (Preprocessing)
Use TensorFlow's ImageDataGenerator for Real-time Data Augmentation
Separate Data into Training and Testing Sets
Convert Photos to RGB and | | Resize to 75x75 Dimensions
Apply Rotation and Translation to Images
Domain Adaptation with FineTuning Class
Add Final Layer (Domain-Specific Predictions)
Apply Regularization Techniques
Experimentation with CNN Architectures and Fine-Tune Each Architecturewith Domain Adaptation for analyzing the performance metrics (Accuracy, F1 Score, Precision, Recall)
Inception ResNet V2
DenseNet
VGG16
ResNet50
EfficientNetB0
XceptionNet
Test Dataset
XAI(Lime explained predictions of six models on the test dataset subset, modifying images with superpixels to observe the impact on predictions)

Figure 4.1: Workflow

4.2 Fine-Tuning for Domain Adaption

The Domain Adaptation process is a crucial step in order to adapt the pre-trained CNN models and here in this paper, the FineTuning class is present to facilitate that. This class is used for storing the domain adaptation process and is in charge of organizing how a pre-trained CNN model can be adjusted to operate successfully for the specific needs of a task. Here, individual layers can be marked as trainable or non-trainable using the class which allows for fine-grained control over which components of the model are updated during training. Therefore, we employed the approach to freeze the model's first layers which consequently signals that these layers need to be updated throughout training. On the other hand, the latter layers are free to adapt to the complexity of our dataset.

The model retains knowledge learned from pre-training on a broad dataset by freezing the first layers thus allowing the latter layers to adjust and achieve a compromise between using current knowledge and fitting the distinctive features of our image domain. Then a final layer is added to the model as an additional step. Moreover, regularization techniques can be used to prevent overfitting. This final layer is critical in enabling the model to produce predictions that are uniquely customized to the characteristics of the domain. Finally, the end result is a fine-tuned model that has preserved significant generic knowledge from pre-training while also adapting to the specific unique features of the dataset, allowing it to generate accurate and domain-specific predictions.

4.3 Experimentation with CNN Architectures

The study is to investigate and test multiple convolutional neural network (CNN) architectures, each with its unique set of capabilities in handling categorization tasks. The experiment_model function contains the experiments, where it is intended to run a series of experiments with various CNN architectures. Six distinct architectures are fine-tuned in these experiments. These architectures are Inception ResNet V2, DenseNet, VGG16, ResNet50, EfficientNetB0, and XceptionNet. This experimentation is conducted methodically on a balanced dataset. This dataset has an equal number of samples across different classes, ensuring that every class is fairly represented. Each architecture is fine-tuned by the incorporation of domain adaptation strategies. The primary motivation is to understand or discern which model is most effective in dealing with the intricacies present in the image data or how well different models capture the complexities in the data. The research involves exploring and fine-tuning not just one, but a diverse set of CNN architectures. Each architecture has its own unique design and characteristics.

During the experimental phase, each model, including Inception ResNet V2, DenseNet, VGG16, ResNet50, EfficientNetB0, and XceptionNet, is thoroughly analyzed. The models are fine-tuned by designating individual layers as trainable in order to adapt pre-trained architectures to the specific image classification. The addition of a final layer gives the model the ability to make domain-specific predictions. The Adam optimizer, categorical cross-entropy loss, and a suite of measures for thorough evaluation, including as accuracy, area under the ROC curve (AUC), false positives, precision, and recall, are included in the compilation. To prevent overfitting, the training covers 40 epochs and employs early stopping based on validation loss. This standardised technique ensures consistent comparison across various architectures.

Inception ResNet V2 differentiates it with enhanced inception modules that capture multi-scale features and residual connections that address the vanishing gradient problem. The dense network pattern of DenseNet encourages robust feature propagation and reuse. VGG16 succeeds in hierarchical feature extraction via repeated 3x3 convolutional layers, despite its simplicity. ResNet50 uses residual connections to achieve efficient gradient flow during backpropagation. XceptionNet uses depthwise separable convolutions for efficient feature extraction, while EfficientNetB0 strikes a balance between model size and performance. Despite the architectural differences, the unified training and assessment strategy allows for an in-depth review of each model's capacity to capture the complexity of the heterogeneous image collection.

4.4 Model Evaluation

The ModelEvaluator class is critical in evaluating the performance of each fine-tuned model. Traditional criteria such as accuracy, precision, recall, and the area under the ROC curve (AUC) are used during the evaluation process. The usual accuracy metric, however, falls short in addressing the intricacies of our balanced multiclass classification problem, where support for each class is the same.

Weighted metrics, notably weighted precision, weighted recall, and weighted F1-score, are employed for a more detailed evaluation. These metrics accommodate for the variable quantity of images available for each class during evaluation, ensuring that the model's performance is fairly and comprehensively assessed throughout the whole dataset. Weighted accuracy, recall, and F1-score provide a more in-depth understanding of the model's capacity to handle a wide range of classes. The inclusion of these indicators simplifies the evaluation procedure, enabling a meaningful comparison of model performance across datasets with varying degrees of balanced classes.

Precision

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

In addition to numerical measures, visualization is critical for expressing the models' training history. Plotting different metrics throughout epochs enable to notice trends and potential areas for model performance improvement. The graphical representation of training history makes it easier to investigate how the model evolves over training epochs, which aids in the refining and optimization of model parameters.

The ModelEvaluator class also creates a detailed classification report as well as a confusion matrix. These tools provide deep insights into the model's performance across various classes, including precision, recall, and other critical measures. The classification report contains detailed information for each class, whereas the confusion matrix visualizes the alignment between the model's predictions and the true labels, improving interpretability and allowing model modification.

4.5 XAI

In the study's exploration of Convolutional Neural Networks (CNNs),the performance of six distinct models was diligently explored. Traditionally, a model's quality is quantified through simple and predefined metrics. A pivotal question surfaces: What defines the quality and value of a model, especially when it boasts phenomenal Top-1 accuracy? While high accuracy is great, it doesn't tell us everything about the model's true value. If a model is good at getting the top prediction right, does that mean it's the best model? Not necessarily. There are other important things to consider. By using XAI to identify and correct biases in the models, the fairness and prevention of unexpected results was ensured. Therefore, XAI has been utilized to compare the behavior among various models and distinguish their differences.

By recognizing and emphasizing these critical regions, Lime's technique provides insightful information about how our models make decisions. For XAI, test dataset is used for better results. The test dataset contains cases that were not encountered by the model during training.Assessing the generalization of the models to new and unforeseen events can be achieved through the utilization of this dataset..

In the context of the implemented code, Lime has been used to explain the predictions of all six model's on a subset of images from our test dataset. With the help of superpixels, it modifies the images, tracks how this affects the predictions, and then creates LIME explanations to show which key areas influence the prediction of the model. This process helps identify which Superpixel areas are most important for predicting a specific class.

An initialization of a LimeImage Explainer object (explainer_lime) has been incorporated within our model's code. This object plays a crucial role in generating local and interpretable explanations for image classification models. Moreover, LIME requires a segmentation method to divide the image into superpixels. That's why the chosen method is SLIC (Simple Linear Iterative Clustering). SLIC played a significant part in dividing the images into superpixels, which allowed for a deeper understanding of the model's behavior. Setting parameters such as n_segments=100, compactness=1, and sigma=1 allowed us to balance the segmentation granularity, superpixel shape regularity, and the smoothing applied during the process.

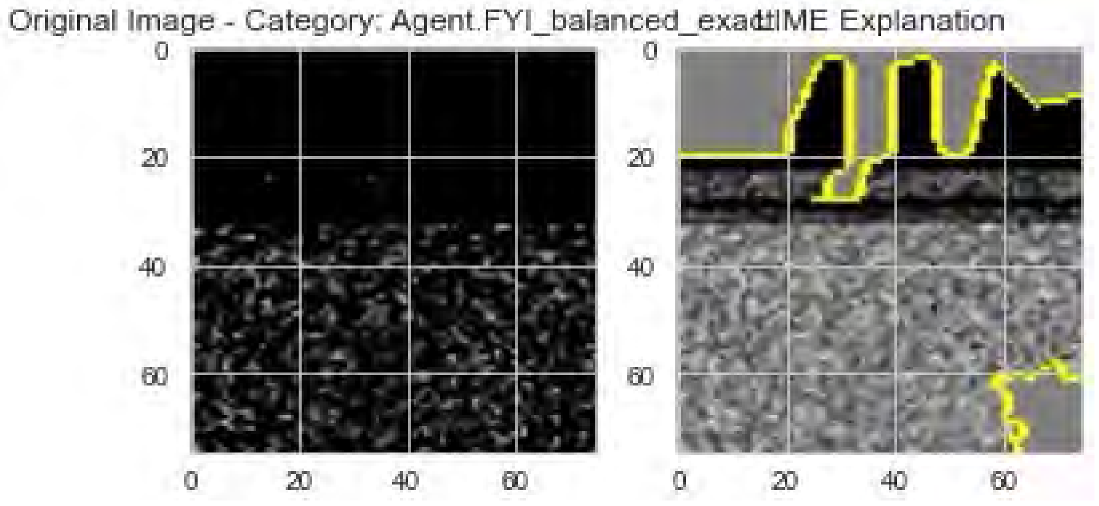


Figure 4.2: XAI Analysis

In the visual analysis pipeline, a comparison is executed between the original image and its LIME interpretation. This aims to provide a comprehensive understanding of the decision-making process employed by the XceptionNet model. The original image is displayed prominently on the left side of each subplot, emphasizing the raw visual input that the model processes. On the right-hand side, the LIME interpretation takes the spotlight. This visual representation is critical in identifying the influential regions, or superpixels, that contribute significantly to the XceptionNet model's prediction for the corresponding original image. By isolating and highlighting these important regions, the LIME interpretation acts as a spotlight on the specific features that contribute to the model's decision. This dual presentation improves the transparency and interpretability of our XceptionNet-based image classification system by presenting both the original image and its LIME interpretation.

On the right-hand side, the LIME interpretation takes the spotlight. This visual representation is critical in identifying the influential regions, or superpixels, that contribute significantly to the XceptionNet model's prediction for the corresponding original image. By isolating and highlighting these important regions, the LIME interpretation acts as a spotlight on the specific features that contribute to the model's

decision. This dual presentation improves the transparency and interpretability of our XceptionNet-based image classification system by presenting both the original image and its LIME interpretation. This allows for a more nuanced and understandable analysis of the complex interactions between image features and model predictions.

Evaluating the LIME explanations revealed important insights into the decision-making process as well as patterns. Certain superpixels were found to be visible in all images and categories, implying that these characteristics were shared and had a significant impact on predictions. Furthermore, the visualisations demonstrated how sensitive the model was to particular aspects of the images, offering insightful information on the elements that went into classifying the images.

Similarly, Local Interpretable Model-agnostic Explanations (LIME) has been implemented across a diverse set of six prominent neural network architectures: Inception ResNet V2, DenseNet, VGG16, ResNet50, EfficientNetB0, and XceptionNet. Each model's predictions are accompanied by LIME interpretations, where significant superpixels are highlighted, offering insights into the critical features influencing their outputs. The purpose of this systematic application of LIME across a diverse set of architectures is to provide a comprehensive and comparative analysis.

4.6 Experimentation

The balanced dataset model performance comparison included an in-depth examination of six state-of-the-art deep learning models: Inception ResNet V2, DenseNet, VGG16, ResNet50, EfficientNetB0, and XceptionNet. For each model, the evaluation included visualizations of training history as well as classification data.

4.6.1 Xception

The Xception architecture across 15 epochs, with each epoch lasting between 800 and 1064 seconds(average 932 seconds) is run on GPU Geforce GTX 1050. The model's performance measures during training demonstrate a significant improvement from the first to the fifteenth epoch, with reduced loss, increased accuracy (from 24.04% to 86.11%), and improved precision and recall. There is an odd validation precision of 0.0 in the first epoch on the validation set, prompting more examination. However, by the 15th epoch, the model has achieved an admirable accuracy of 79.80% on the validation set, with good trends in precision, recall, and AUC metrics.

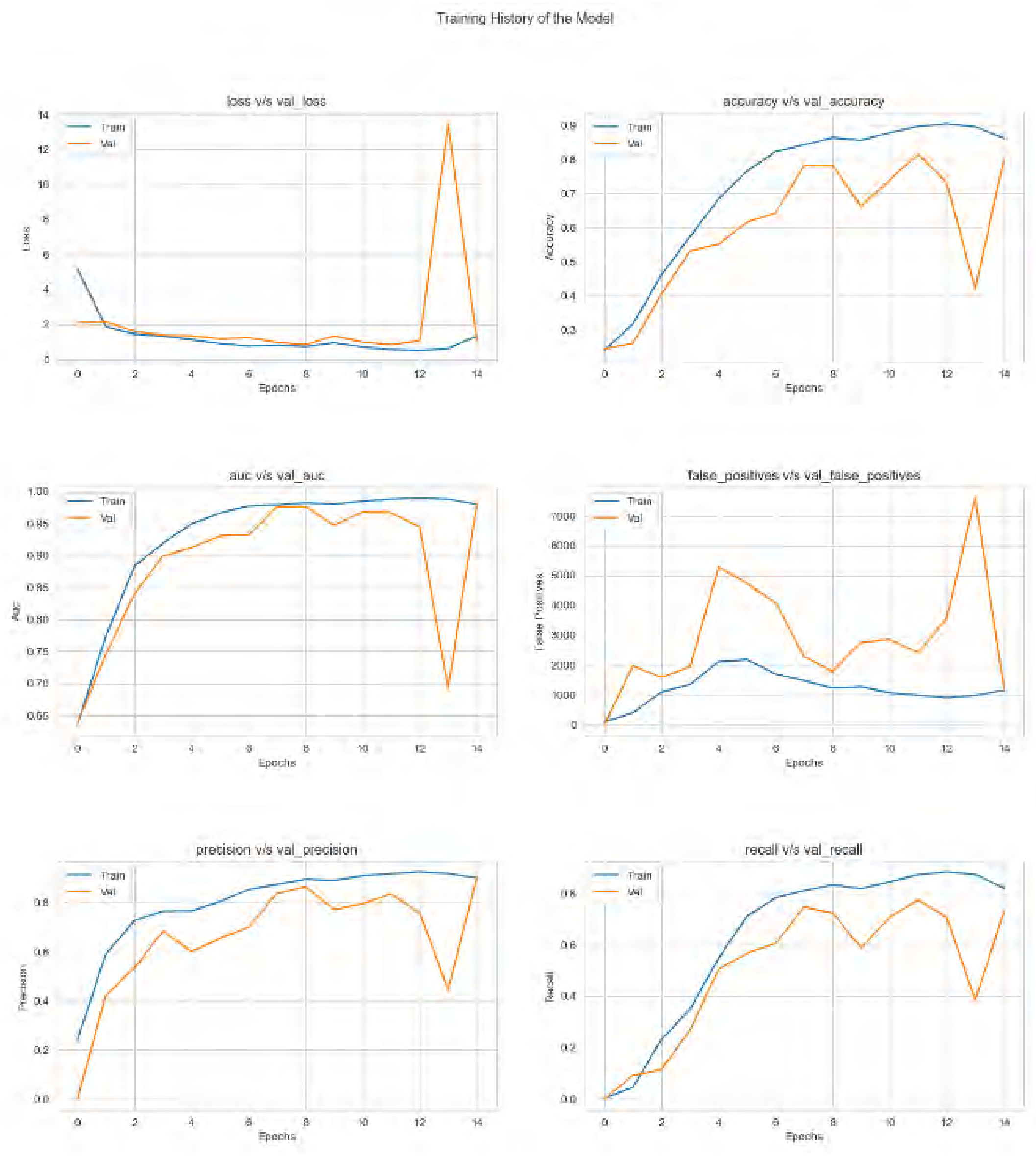
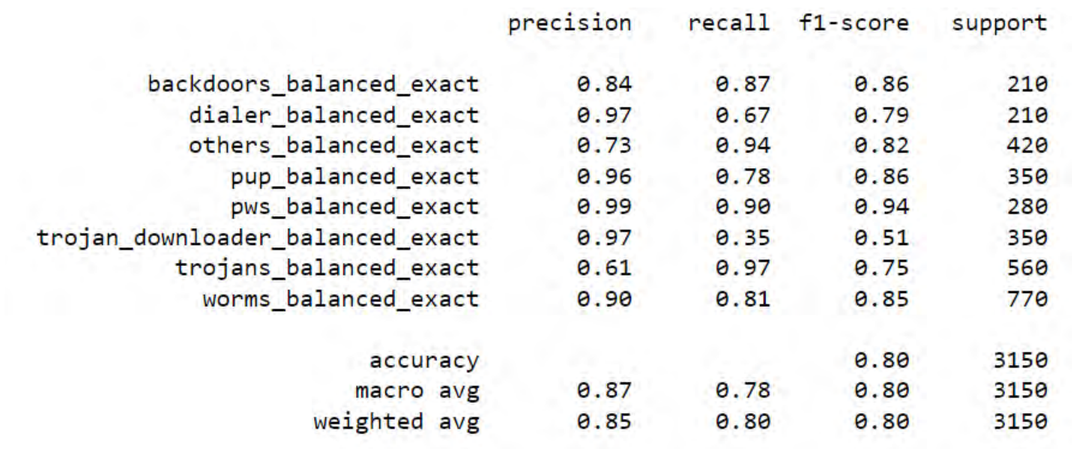


Figure 4.3: Training history of Xception Model



```
                                precision    recall  f1-score   support

        backdoors_balanced_exact       0.84      0.87      0.86       210
           dialer_balanced_exact       0.97      0.67      0.79       210
           others_balanced_exact       0.73      0.94      0.82       420
              pup_balanced_exact       0.96      0.78      0.86       350
              pws_balanced_exact       0.99      0.90      0.94       280
trojan_downloader_balanced_exact       0.97      0.35      0.51       350
          trojans_balanced_exact       0.61      0.97      0.75       560
            worms_balanced_exact       0.90      0.81      0.85       770

                        accuracy                           0.80      3150
                       macro avg       0.87      0.78      0.80      3150
                    weighted avg       0.85      0.80      0.80      3150
```

Figure 4.4: Xception Performance Metrics

4.6.2 EfficientNet

The EfficientNet architecture shows progress over ten epochs, with each epoch averaging 768 to 1009 seconds(average 888.5 seconds) on the same GPU. Throughout the training period, the model shows significant improvements, as seen by a decrease in loss from 10.3237 to 0.3990 and an increase in accuracy from 59.57% to 92.60%. Precision, recall, and AUC metrics show favorable trends as well, highlighting the model's ability to learn and adapt throughout training. The model obtains an outstanding accuracy of 87.92% in the 9th epoch on the validation set. Precision, recall, and AUC measures all regularly show strong performance.

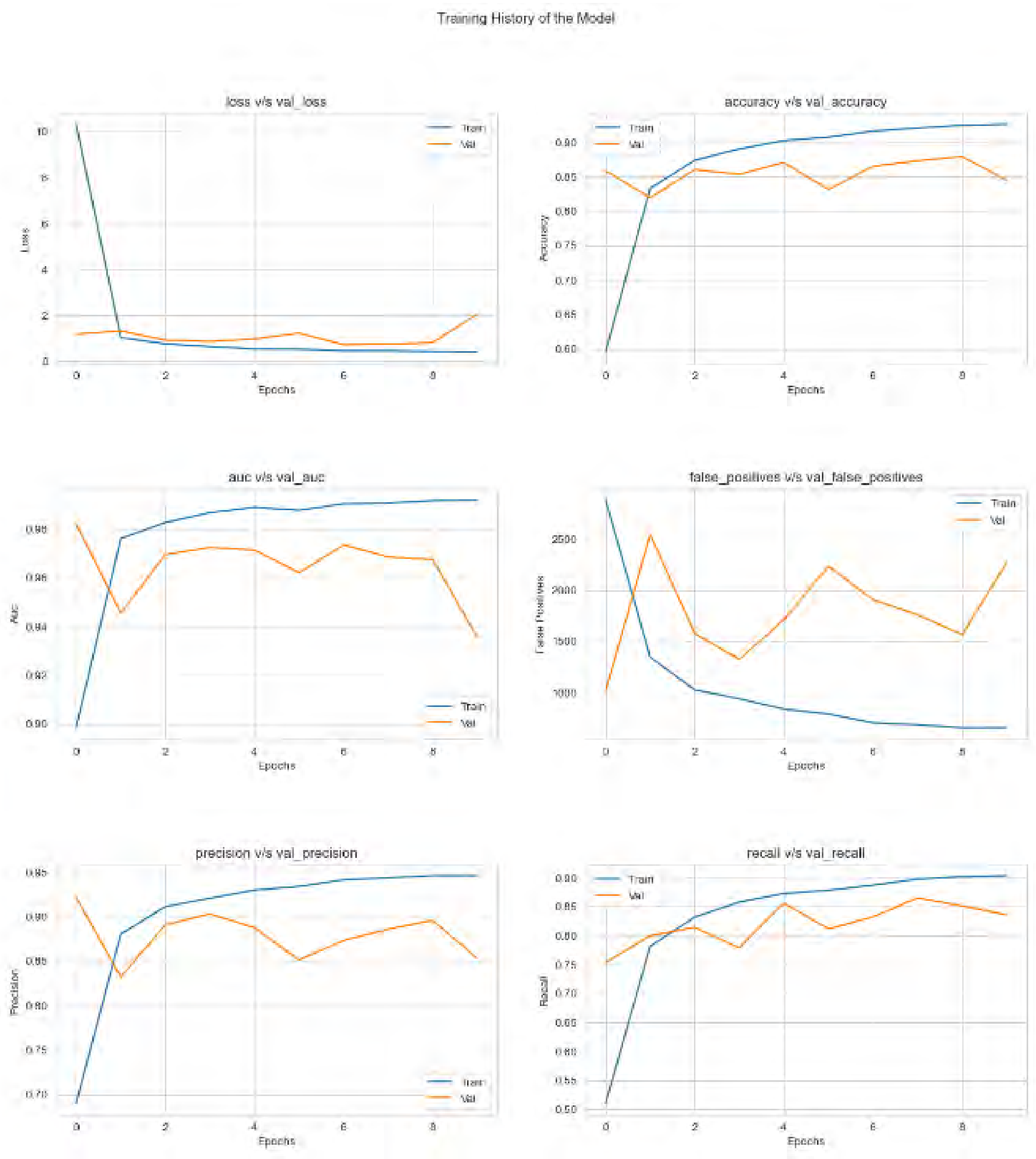
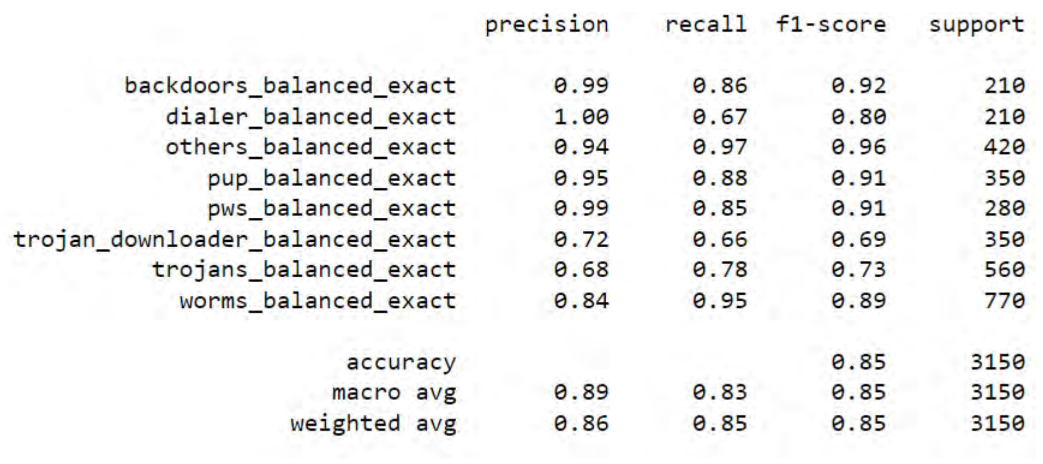


Figure 4.5: Training history of EfficientNet Model



```
                                   precision    recall  f1-score   support

        backdoors_balanced_exact        0.99      0.86      0.92       210
           dialer_balanced_exact        1.00      0.67      0.80       210
           others_balanced_exact        0.94      0.97      0.96       420
              pup_balanced_exact        0.95      0.88      0.91       350
              pws_balanced_exact        0.99      0.85      0.91       280
trojan_downloader_balanced_exact        0.72      0.66      0.69       350
          trojans_balanced_exact        0.68      0.78      0.73       560
            worms_balanced_exact        0.84      0.95      0.89       770

                        accuracy                            0.85      3150
                       macro avg        0.89      0.83      0.85      3150
                    weighted avg        0.86      0.85      0.85      3150
```

Figure 4.6: EfficientNetPerformance Metrics

4.6.3 ResNet

Over 11 epochs, the ResNet architecture demonstrates the model's growth in terms of loss, accuracy, and numerous metrics. Each epoch took an average duration ranging from 951 to 1386 seconds(average 1168.5 seconds). Over the epochs, the model shows a considerable reduction in loss from 21.1246 to 0.5474 and an improvement in accuracy from 50.70% to 91.42%. Precision, recall, and AUC indicators all exhibit positive trends during training, highlighting the model's ability to learn and adapt to new data. The model obtains an accuracy of 80.00% in the 11th epoch on the validation set.

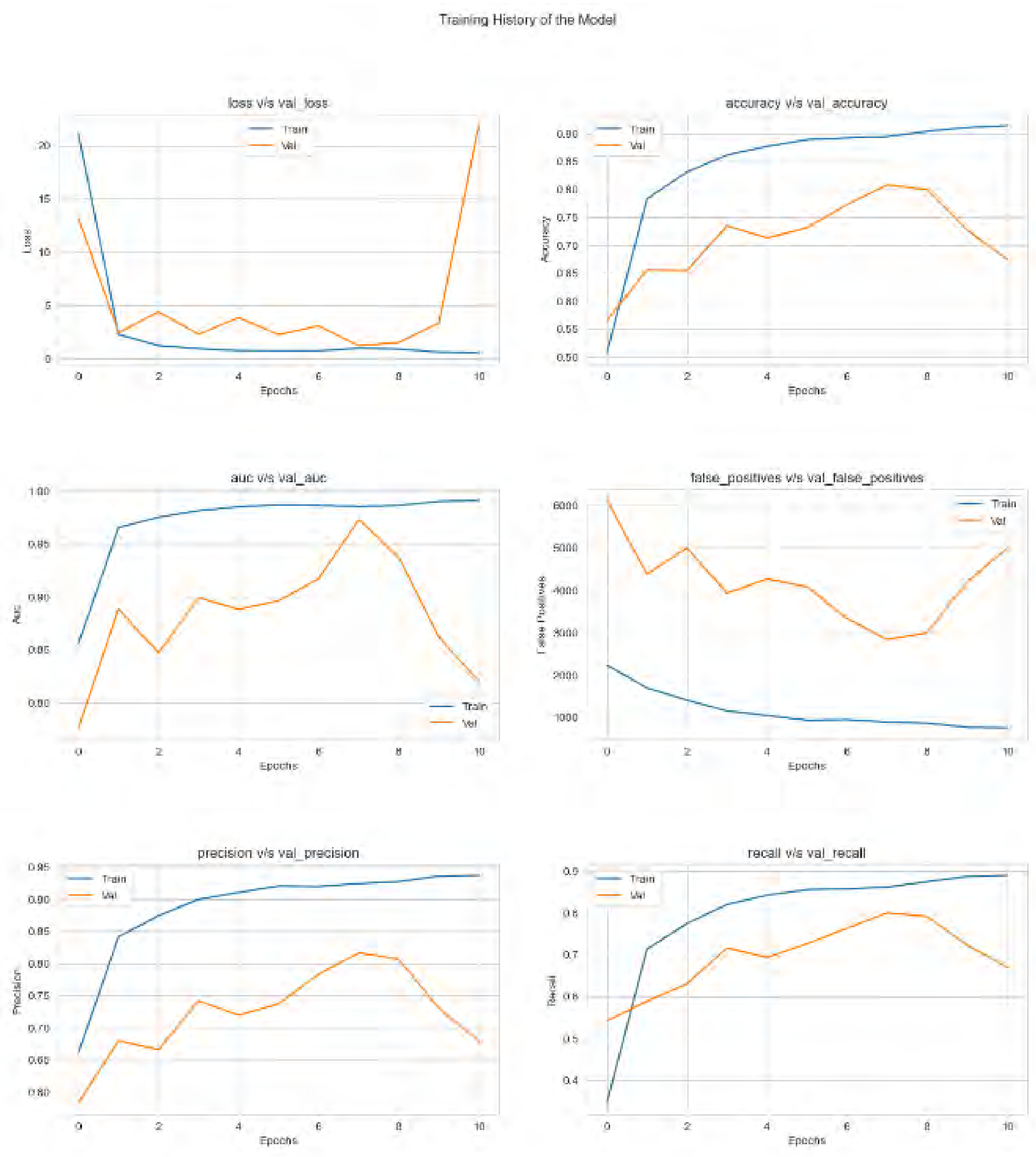
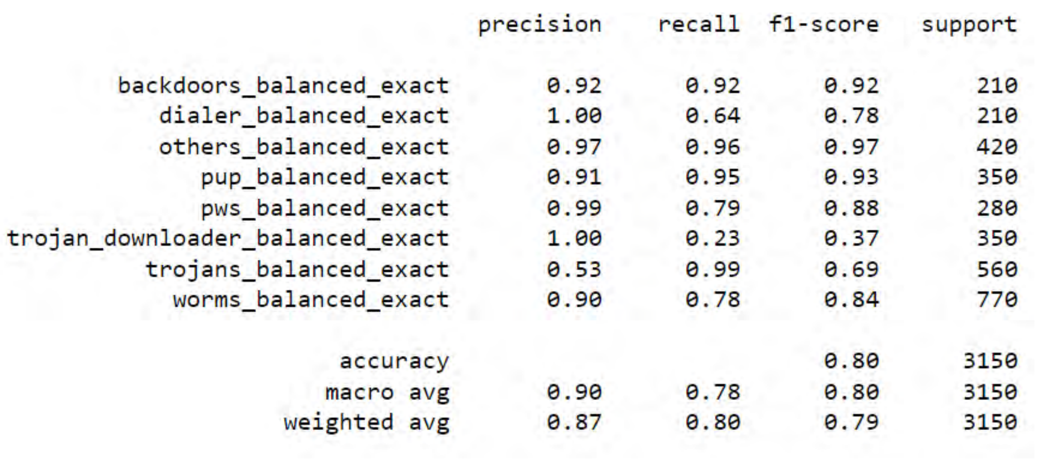


Figure 4.7: Training history of ResNet Model



```
                                   precision    recall  f1-score   support

          backdoors_balanced_exact       0.92      0.92      0.92       210
             dialer_balanced_exact       1.00      0.64      0.78       210
             others_balanced_exact       0.97      0.96      0.97       420
                pup_balanced_exact       0.91      0.95      0.93       350
                pws_balanced_exact       0.99      0.79      0.88       280
trojan_downloader_balanced_exact       1.00      0.23      0.37       350
            trojans_balanced_exact       0.53      0.99      0.69       560
              worms_balanced_exact       0.90      0.78      0.84       770

                          accuracy                           0.80      3150
                         macro avg       0.90      0.78      0.80      3150
                      weighted avg       0.87      0.80      0.79      3150
```

Figure 4.8: ResNet Performance Metrics

4.6.4 VGG16

The VGG16 model's training across 33 epochs (average 930.5 seconds) shows a gradual increase in both loss reduction and accuracy on the validation set. The model starts with a significant loss of 40.2224, but by the 30th epoch, this has been drastically reduced to 0.5649. Simultaneously, the accuracy rises from 30.48% to 83%, reflecting the model's improved ability to accurately categorize occurrences. Precision, recall, and AUC measurements show favorable trends throughout training, highlighting the model's efficacy over multiple classes. The confusion matrix displays excellent results, particularly in classes such as 'pws_balanced_exact' and 'others_balanced_exact.' The weighted average F1-score of 0.83 obtained indicates a balanced trade-off between precision and recall. With an accuracy of 83% on the validation set, this model demonstrates its robustness in discriminating between several malware classes, indicating its potential practical utility.

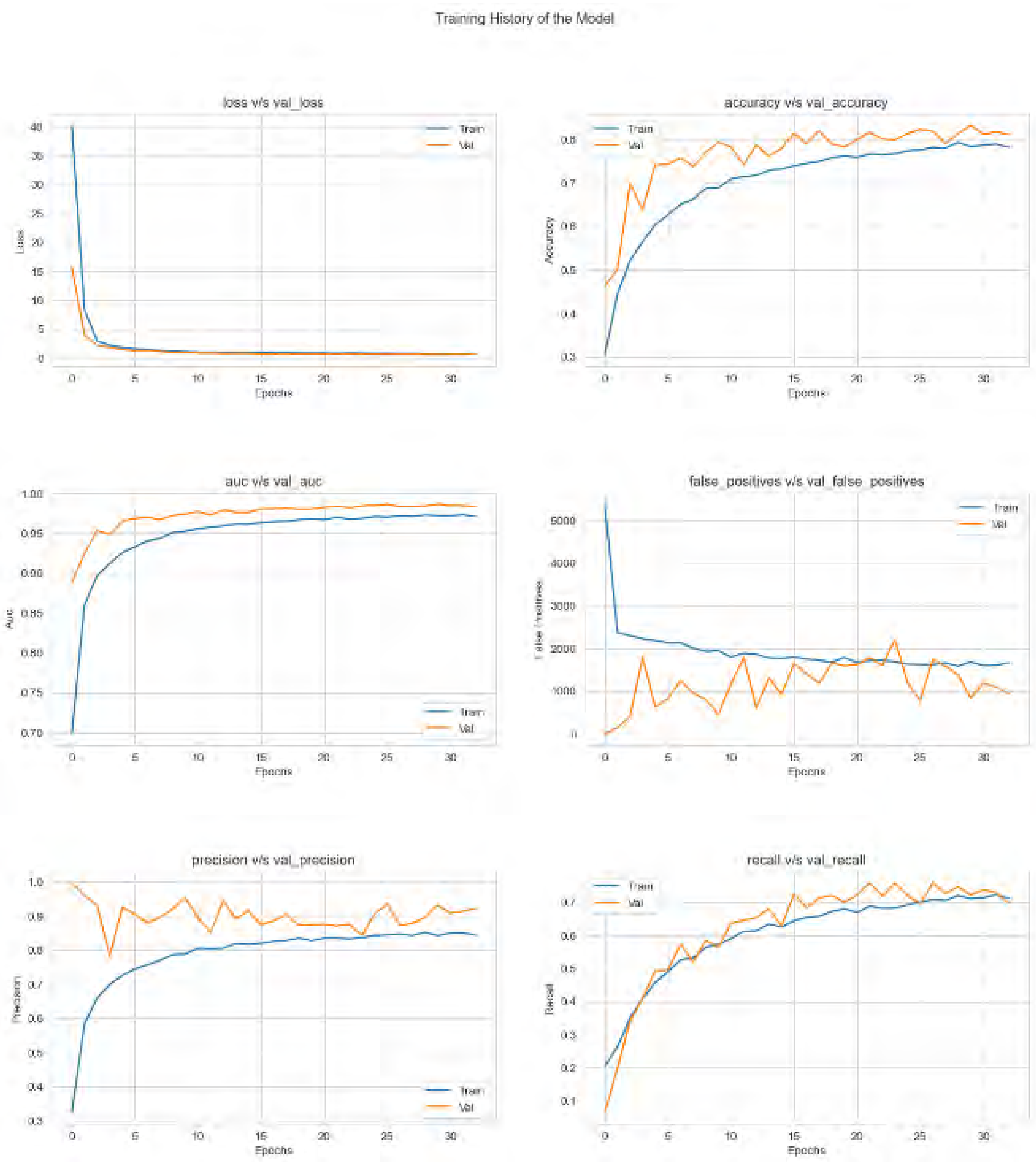
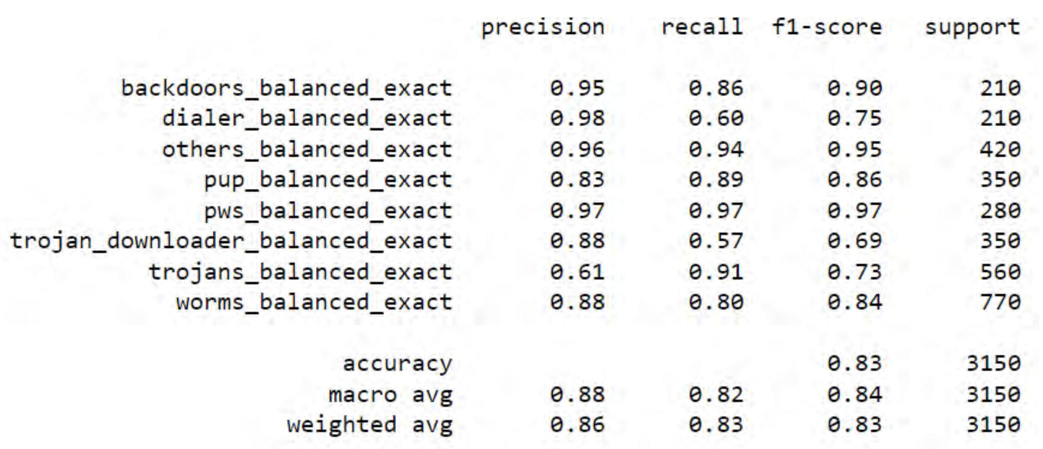


Figure 4.9: Training history of VGG16 Model



```
                                   precision    recall  f1-score   support

         backdoors_balanced_exact       0.95      0.86      0.90       210
            dialer_balanced_exact       0.98      0.60      0.75       210
            others_balanced_exact       0.96      0.94      0.95       420
               pup_balanced_exact       0.83      0.89      0.86       350
               pws_balanced_exact       0.97      0.97      0.97       280
trojan_downloader_balanced_exact       0.88      0.57      0.69       350
           trojans_balanced_exact       0.61      0.91      0.73       560
             worms_balanced_exact       0.88      0.80      0.84       770

                         accuracy                           0.83      3150
                        macro avg       0.88      0.82      0.84      3150
                     weighted avg       0.86      0.83      0.83      3150
```

Figure 4.10: VGG16 Performance Metrics

4.6.5 DenseNet

The DenseNet model shows significant growth in terms of loss, accuracy, and other metrics throughout 11 epochs. Each epoch takes a range of 863 to 1060 seconds(average 961.5) to complete. The model improves in accuracy from 23.72% to 73.81% while decreasing in loss from 20.1405 to 0.8734. While the model achieves relatively high precision and recall for some classes, it struggles to reliably categorize others, such as "dialer_balanced_exact," "pws_balanced_exact," and "trojan_downloader_balanced_exact." After 11 epochs, the overall accuracy on the validation set is 66%.

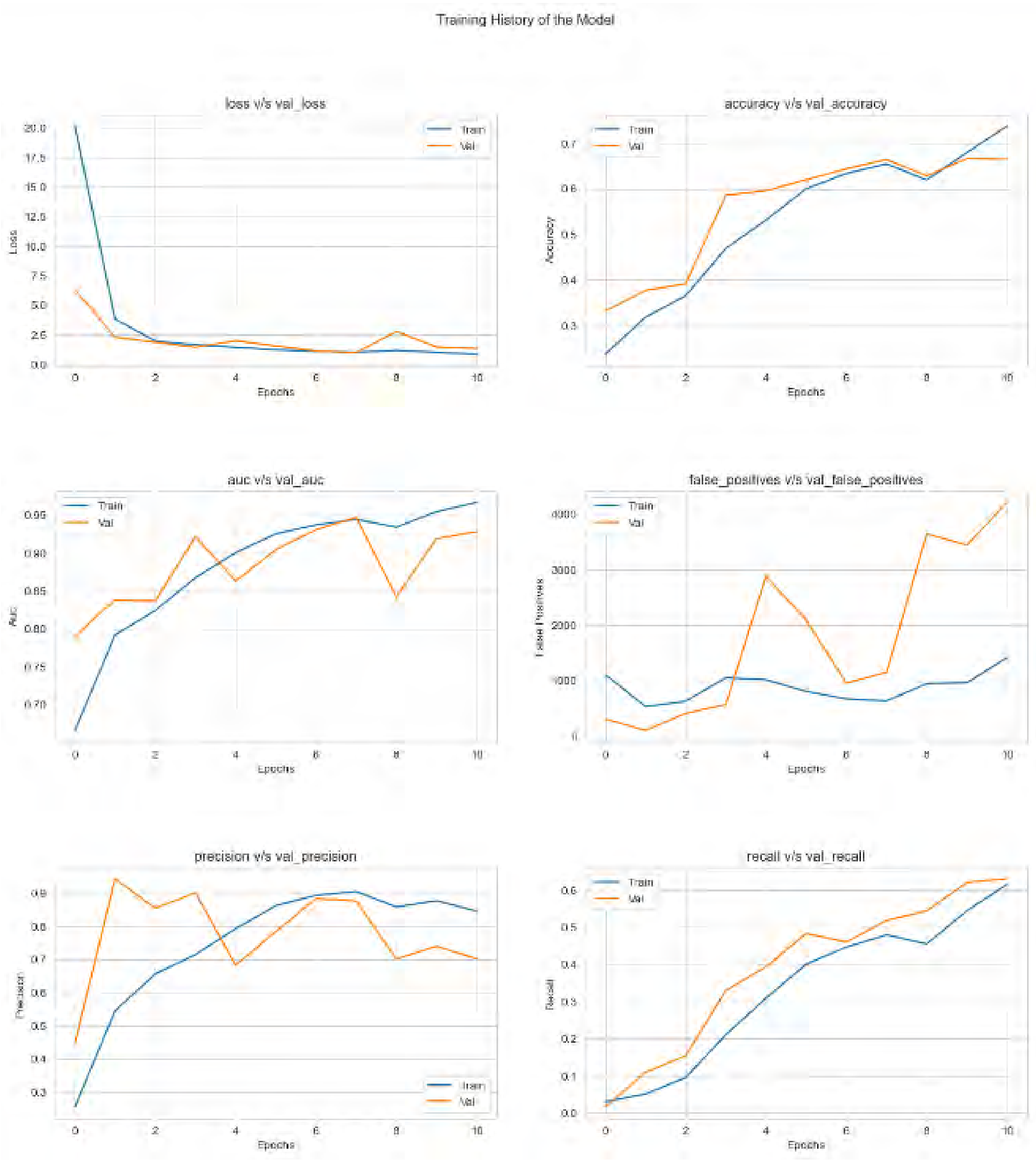


Figure 4.11: Training history of DenseNet Model

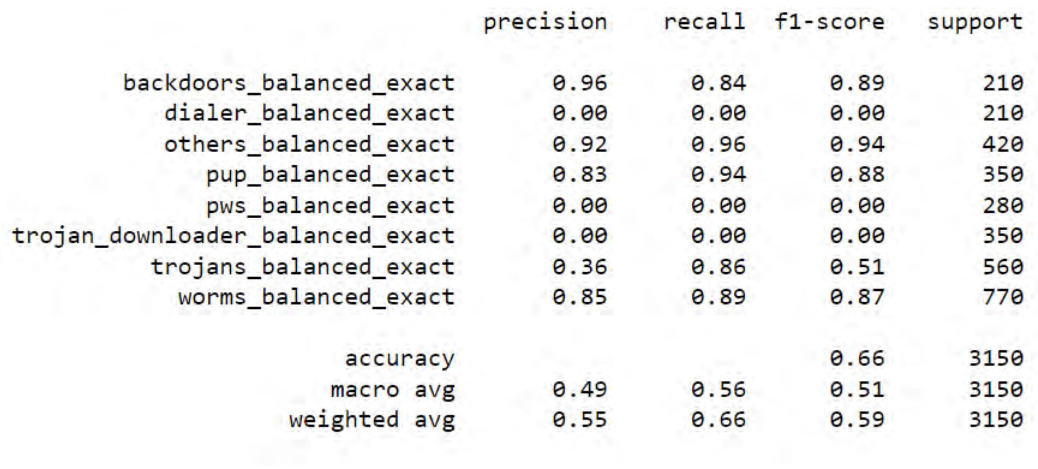

```
                                  precision    recall  f1-score   support

         backdoors_balanced_exact       0.96      0.84      0.89       210
            dialer_balanced_exact       0.00      0.00      0.00       210
            others_balanced_exact       0.92      0.96      0.94       420
               pup_balanced_exact       0.83      0.94      0.88       350
               pws_balanced_exact       0.00      0.00      0.00       280
trojan_downloader_balanced_exact       0.00      0.00      0.00       350
           trojans_balanced_exact       0.36      0.86      0.51       560
             worms_balanced_exact       0.85      0.89      0.87       770

                         accuracy                           0.66      3150
                        macro avg       0.49      0.56      0.51      3150
                     weighted avg       0.55      0.66      0.59      3150
```

Figure 4.12: DenseNet Performance Metrics

4.6.6 InceptionResNetV2

In the first epoch, InceptionResNetV2 has a reasonably high loss, moderate accuracy, and precision, with recall levels changing between classes. The period of each epoch ranges between 901 and 1165 seconds(average 1033 seconds). The model's performance improves noticeably as training goes on. Loss is reduced, while accuracy, precision, and recall levels improve. The model achieves an amazing overall accuracy of 91% on the validation set by the 13th epoch. Individual class categorization results are encouraging, with good precision and recall values reported across multiple categories. This indicates that the model is effectively learning the data's properties and patterns.

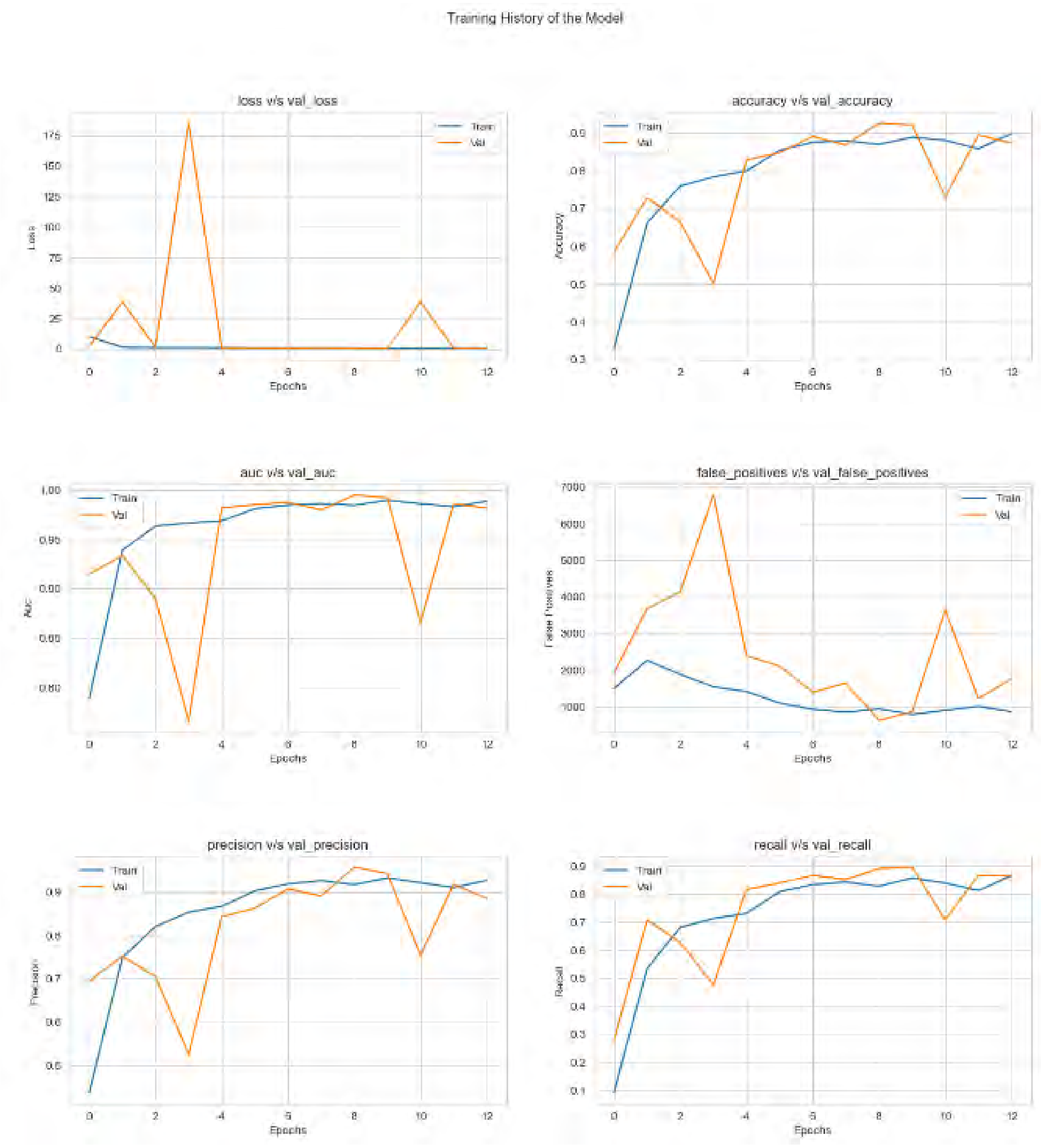
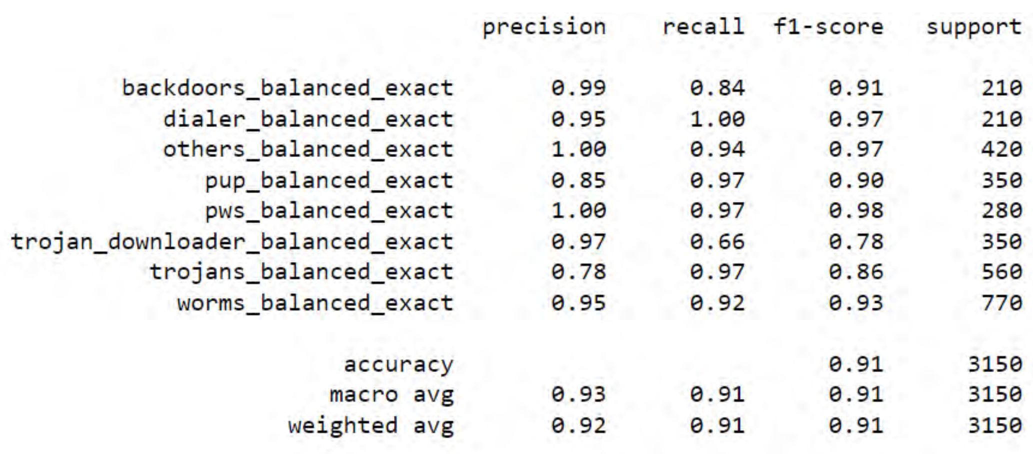


Figure 4.13: Training history of InceptionNetResnetV2 Model



```
                                precision    recall  f1-score   support

        backdoors_balanced_exact       0.99      0.84      0.91       210
           dialer_balanced_exact       0.95      1.00      0.97       210
           others_balanced_exact       1.00      0.94      0.97       420
              pup_balanced_exact       0.85      0.97      0.90       350
              pws_balanced_exact       1.00      0.97      0.98       280
trojan_downloader_balanced_exact       0.97      0.66      0.78       350
          trojans_balanced_exact       0.78      0.97      0.86       560
            worms_balanced_exact       0.95      0.92      0.93       770

                        accuracy                           0.91      3150
                       macro avg       0.93      0.91      0.91      3150
                    weighted avg       0.92      0.91      0.91      3150
```

Figure 4.14: InceptionResNetV2 Performance Metrics

Chapter 5

Result

5.1 Modelwise Comparison

InceptionResNetV2 is the best performer of the six models, with the highest accuracy of 91% and the highest recall values. This means that it is effective at correctly identifying occurrences of positive classes, making it a solid choice, particularly where minimizing false negatives is critical. However, InceptionResNetV2 training times were considerably longer, indicating potentially increased processing requirements throughout the training process.

VGG16 is close behind, with a balanced performance across accuracy, precision, and recall. VGG16 exhibits efficiency in training due to its shorter epoch durations, giving it a dependable alternative for applications that require a good compromise across several metrics.

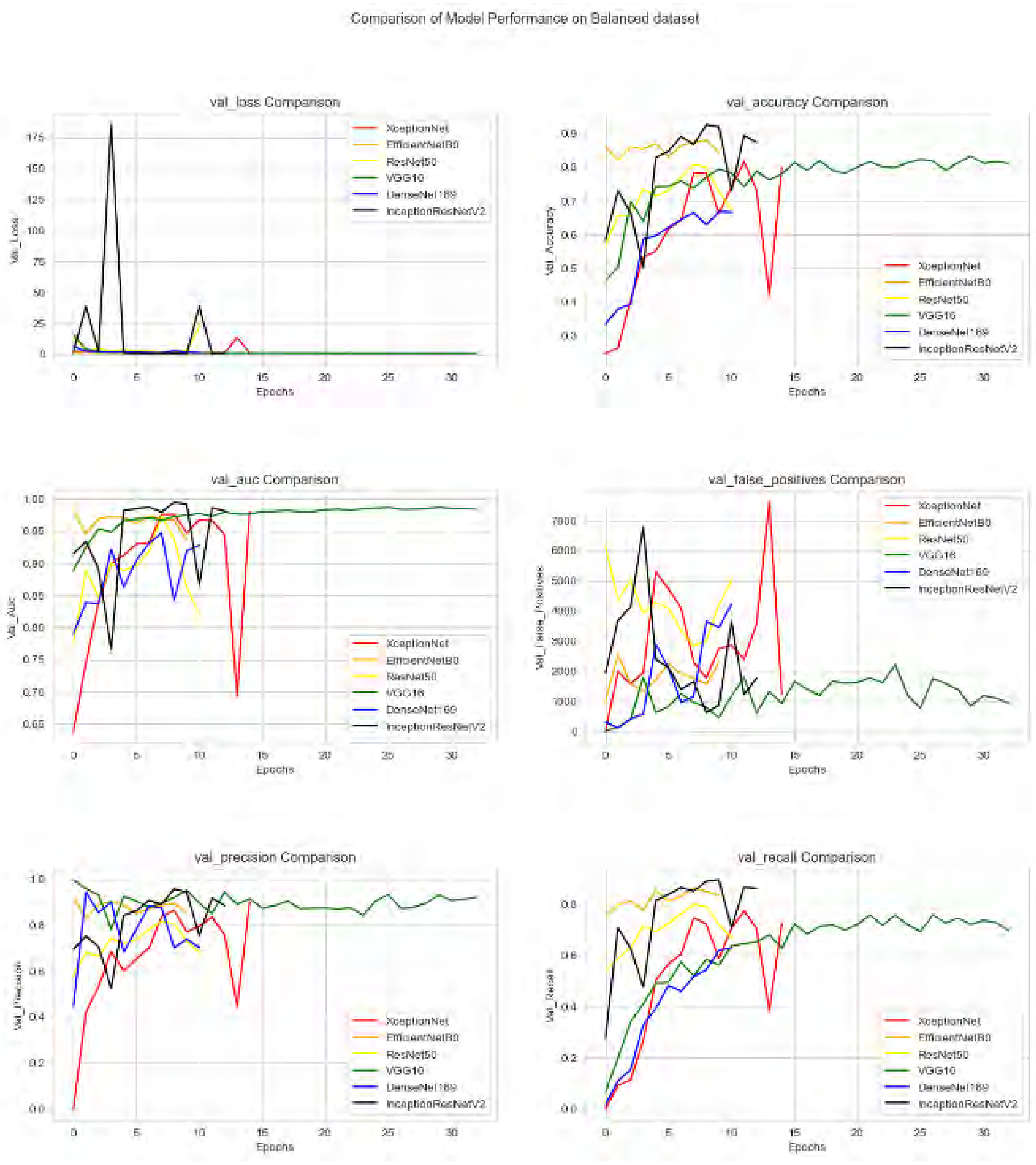


Figure 5.1: Comparision of Model Performance on Balanced Dataset

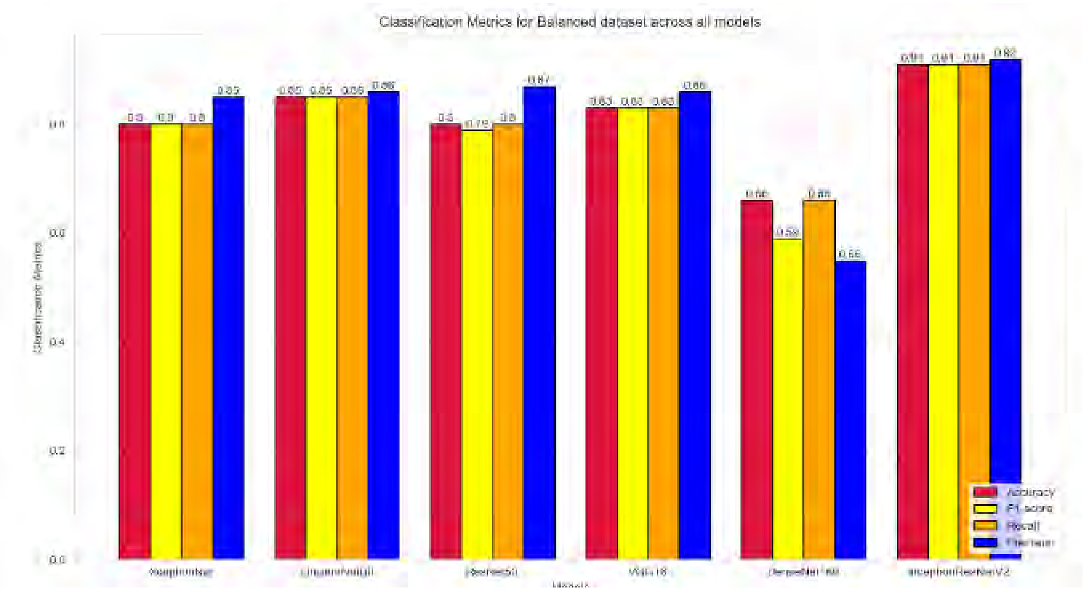


Figure 5.2: Classification Matrics for Balanced datasets across all models

DenseNet, with a precision of 66%, demonstrated a trade-off between precision and recall. While it delivered faster training times, it made a minor sacrifice in its ability to reduce false positives while capturing a considerable number of real positives. The compromise between precision and recall in DenseNet may be acceptable depending on the unique requirements of an application, particularly in cases where computational resources are limited.

5.2 XAI Analysis

The Lime analysis of multiple models revealed different patterns of feature attention, which influenced their prediction accuracy. InceptionResNet demonstrated high accuracy, attributing its success to strong identification and utilization of key image areas. DenseNet, on the other hand, demonstrated lower accuracy, implying difficulties in capturing domain-specific features. For XceptionNet, ResNet, and EfficientNet, Lime consistently highlighted relevant features, contributing to their commendable accuracy.

It was observed that in some cases, Lime did not show any highlighted regions.

Here is an example of Desnet model prediction, where the Lime explanation showed there is no highlight, which indicates that the model struggles to recognize essential details in the image. This lack of highlighted regions in Lime's explanation points

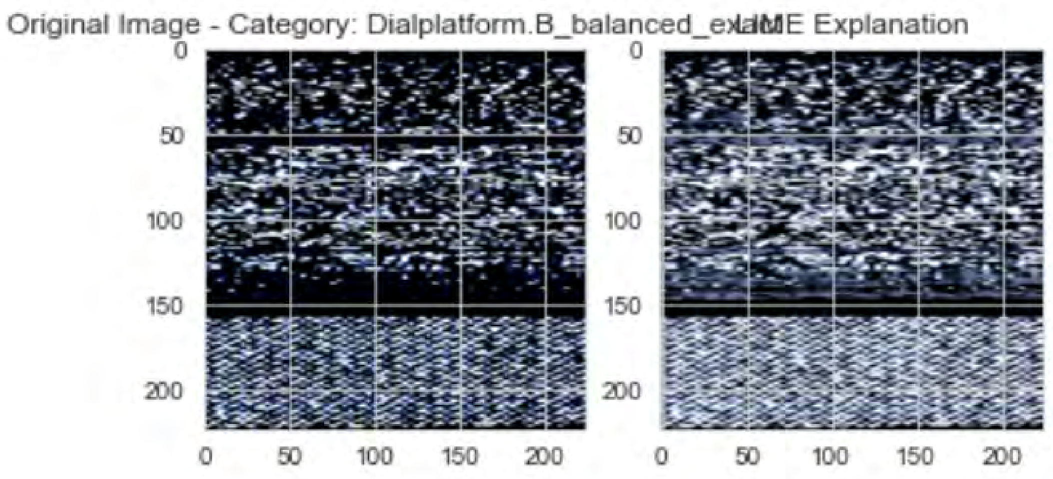


Figure 5.3: XAI Analysis of DenseNet without highlight

to the model's difficulty in identifying key features, giving us valuable insights into its limitations in understanding certain aspects of the image. These findings underscore the crucial role Lime plays in understanding model behavior and identifying strengths and weaknesses in handling diverse image characteristics.

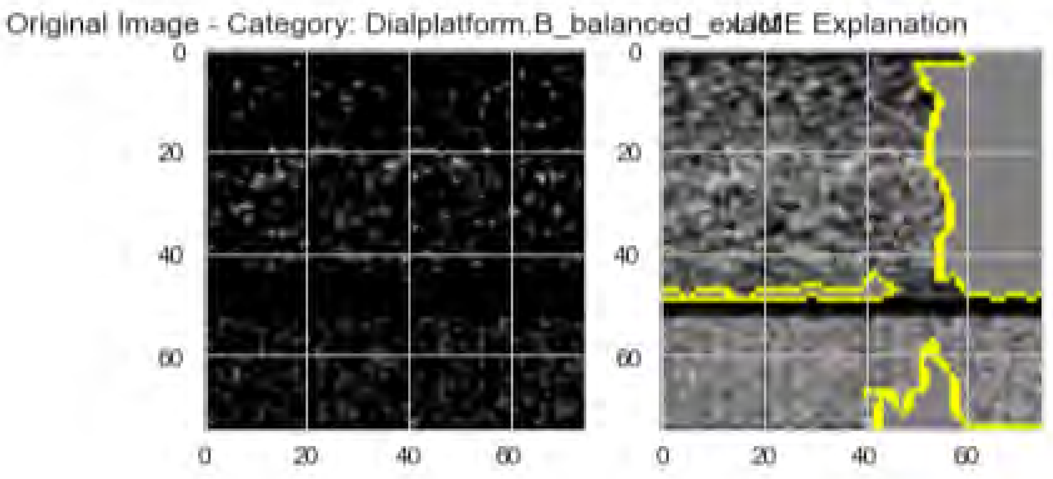


Figure 5.4: XAI Analysis of InseptionResnetv2 with highlight

In another case, it can be seen that distinct features are highlighted while using the Lime explanation in the EfficientNet model.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The evaluation of diverse CNN models—Xception, EfficientNet, ResNet, VGG16, DenseNet, and InceptionResNetV2—for malware image classification exhibited varying performance. InceptionResNetV2 showcased the highest accuracy at 91% on the validation set, closely followed by VGG16, displaying a balanced performance across metrics. However, DenseNet, despite quicker training, demonstrated a trade-off between precision and recall thus impacting its ability to discern false positives while capturing true positives. The Lime analysis explained feature attention patterns, highlighting InceptionResNet's effective identification of key image areas for higher accuracy, while DenseNet struggled to capture essential domain-specific features. Additionally, Lime's explanation underscored instances where models failed to recognize crucial image details, emphasizing their limitations. Overall, InceptionResNetV2 emerged as the most accurate model, with Lime analysis providing insights into feature attention patterns, contributing to a comprehensive understanding of model performance in malware image classification. There is substantial potential for future research and improvement in this domain.

6.2 Future Work

The 45 malware samples were classified into 8 distinct classes of malware families through a theoretical research-based approach. To validate their categorization, future examinations could leverage Virtual Machines or Sandboxes for verification against these predefined families with actual malwares.

In this research, six Convolutional Neural Network (CNN) models were utilized, with InceptionResNetV2 exhibiting the most promising performance. InceptionResNetV2 amalgamates the strengths of two established models: Inception, focused on determining computational cost, and ResNet, prioritizing computational accuracy. The success of this hybrid model in outperforming others encourages the exploration of additional hybrid models to further enhance accuracy like NASNet or AmoebaNet.

Although only Local Interpretable Model-agnostic Explanations (LIME) were implemented for analysis, other eXplainable AI (XAI) methodologies could offer more comprehensive insights. Potential XAI elements like Grad-CAM (Gradient-weighted Class Activation Mapping) or Integrated Gradients can provide more detailed and accurate analysis compared to LIME, offering a deeper understanding of model decisions and performance.

Bibliography

[1] Belcic, I. (2023, December 18). What is malware and how to protect against malware attacks? What Is Malware and How to Protect Against Malware Attacks? https://www.avast.com/c-malware

[2] Efe, Doç. Dr. Ahmet & Hussin, Saleh. (2020). Malware Visualization Techniques. *International Journal of Applied Mathematics Electronics and Computers, 8, 7-20. DOI: 10.18100/ijamec.526813.*

[3] Dipendra Pant and Rabindra Bista. 2021. Image-based Malware Classification using Deep Convolutional Neural Network and Transfer Learning. In AISS 2021, Sanya, China. ACM, New York, NY, USA, 9 Pages. https://doi.org/10.1145/3503047.3503081

[4] Charmet, F., Tanuwidjaja, H.C., Ayoubi, S. et al. Explainable artificial intelligence for cybersecurity: a literature survey. *Ann. Telecommun. 77, 789–812 (2022). DOI: 10.1007/s12243-022-00926-7*

[5] Ksibi, A., Zakariah, M., Almuqren, L. A., & Alluhaidan, A. S. (2023). Deep Convolution Neural Networks and Image Processing for Malware Detection. *Research Square (Research Square). DOI: 10.21203/rs.3.rs-2508967/v1*

[6] Kumar, M. G. R. (2022). IVMCT: Image Visualization based Multiclass Malware Classification using Transfer Learning. *The Philippine Statistician (Quezon City), 71(2). DOI: 10.17762/msea.v71i2.65*

[7] Hota, A., Panja, S., & Nag, A. (2022). Lightweight CNN-based malware image classification for resource-constrained applications. *Innovations in Systems and Software Engineering. DOI: 10.1007/s11334-022-00461-7*

[8] Bensaoud, A., Abudawaood, N., & Kalita, J. (2020). Classifying Malware Images with Convolutional Neural Network Models. *arXiv (Cornell University). DOI: 10.6633/ijns.202011_2*2(6).17

[9] Gibert, D., Mateu, C., Planes, J., & Vicens, R. (2019). Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques, 15(1), 15–28. DOI: 10.1007/s11416-018-0323-0*

[10] Lad, S. S., & Adamuthe, A. C. (2021). Malware Classification with Improved Convolutional Neural Network Model. *International Journal of Computer Network and Information Security, 12(6), 30–43. DOI: 10.5815/ijcnis.2020.06.03*

[11] Venkatraman, S., Alazab, M., & Vinayakumar, R. (2019). A hybrid deep learning image-based analysis for effective malware detection. *Journal of Information Security and Applications, 47, 377–389.
DOI: 10.1016/j.jisa.2019.06.006*

[12] Dipendra Pant and Rabindra Bista. 2021. Image-based Malware Classification using Deep Convolutional Neural Network and Transfer Learning. In AISS 2021, Sanya, China. ACM, New York, NY, USA, 9 Pages. https://doi.org/10.1145/3503047.3503081

[13] Y. Liu, C. Tantithamthavorn, L. Li and Y. Liu, "Explainable AI for Android Malware Detection: Towards Understanding Why the Models Perform So Well?," 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), Charlotte, NC, USA, 2022, pp. 169-180, doi: 10.1109/ISSRE55969.2022.00026.

[14] Deepa, K., Adithyakumar, K. S., & Vinod, P. (2022). Malware Image Classification using VGG16. *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS). DOI: 10.1109/ic3sis54991.2022.9885587*

[15] W. W. Lo, X. Yang and Y. Wang. An Xception Convolutional Neural Network for Malware Classification with Transfer Learning. NTMS 2019, Canary Islands, Spain. DOI: 10.1109/NTMS.2019.8763852.

[16] Hemalatha, J., Roseline, S. A., Geetha, S., Kadry, S., & Damaševičius, R. (2021). An Efficient DenseNet-Based Deep Learning Model for Malware Detection. *Entropy, 23(3), 344. DOI: 10.3390/e23030344*

[17] Ravi, V., & Chaganti, R. (2022). EfficientNet deep learning meta-classifier approach for image-based android malware detection. *Multimedia Tools and Applications, 82(16), 24891–24917. DOI: 10.1007/s11042-022-14236-6*

[18] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B.S. (2011). Malware images: visualization and automatic classification. *Visualization for Computer Security.*

[19] MaleVis Dataset Home page. (n.d.). https://web.cs.hacettepe.edu.tr/~selman/malevis/

[20] Kenton, W. (2023, January 16). Adware: What it is, history, malicious use. Investopedia. https://www.investopedia.com/terms/a/adware.asp

[21] Adware.Win32.Amonetize.G - Threat Encyclopedia. (n.d.). https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/Adware.Win32.Amonetize.G/

[22] Adware.Win32.BrowseFox.NHUSFDG - Threat Encyclopedia. (n.d.). https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/adware.win32.browsefox.nhusfdg/

[23] Adware.Win32.Elex.J - Threat Encyclopedia. (n.d.). https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/Adware.Win32.Elex.J

[24] Bedell, C., Loshin, P., & Hanna, K. T. (2022, September 13). computer worm. Security. https://www.techtarget.com/searchsecurity/definition/worm

[25] Bailey, R. (2020, May 20). Backdoor.Win32.Androm. How to Fix Guide. https://howtofix.guide/backdoor-win32-androm/#BackdoorWin32Androm

[26] Trojan.Win32.ANDROM.AF - Threat Encyclopedia. (n.d.). https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/trojan.win32.androm.af/

[27] Corporation, M. (n.d.). Threat description search results - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/threat-search?query=Win32/fakerean

[28] Trojan-Downloader:W32/Fakerean.gen!A — F-Secure Labs. (n.d.). https://www.f-secure.com/v-descs/trojan-downloader-w32-fakerean-gen!a.shtml

[29] Malwarebytes. (2023, September 28). Virus.Neshta. https://www.malwarebytes.com/blog/detections/virus-neshta

[30] Win32/Neshta.A — ESET Virusradar. (n.d.). https://www.virusradar.com/en/Win32_Neshta.A/description

[31] Malwarebytes. (2023a, August 16). Trojan.Stantiko. https://www.malwarebytes.com/blog/detections/trojan-stantiko

[32] Stantinko Trojan: What it is, how it works and how to prevent it — Malware spotlight — Infosec. (n.d.). https://resources.infosecinstitute.com/topics/malware-analysis/stantinko-trojan-what-it-is-how-it-works-and-how-to-prevent-it-malware-sp
#:~:text=Introduction,%2Dminers%20and%20adware%20botnets%E2%80%9D.

[33] Malwarebytes. (2023b, September 27). HackKMS.HackTool.RiskWare.DDS. https://www.malwarebytes.com/blog/detections/hackkms-hacktool-riskware-dds#:~:text=Short%20bio-,HackKMS.,4%20and%20Malwarebytes%20business%20products.

[34] What is a Trojan horse and what damage can it do? (2023, October 3). usa.kaspersky.com. https://usa.kaspersky.com/resource-center/threats/trojans

[35] Bailey, R. (2021, August 4). Backdoor.Hlux. How to Fix Guide. https://howtofix.guide/backdoor-hlux/#BackdoorHlux

[36] Trojan:W32/Injector — F-Secure Labs. (n.d.). https://www.f-secure.com/v-descs/trojan-w32-injector.shtml

[37] Trojan:W32/Agent — F-Secure Labs. (n.d.). https://www.f-secure.com/v-descs/agent.shtml

[38] What is a trojan horse? Trojan virus and malware explained — Fortinet. (n.d.). Fortinet. https://www.fortinet.com/resources/cyberglossary/trojan-horse-virus

[39] Corporation, M. (2008, May 20). Backdoor:Win32/Rbot.gen threat description - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Rbot.gen

[40] Kaspersky Threats — Vilsel. (n.d.). https://threats.kaspersky.com/en/threat/Trojan.Win32.Vilsel/

[41] Corporation, M. (2008, August 18). Trojan:Win32/Alureon.gen!J threat description - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Alureon.gen!J

[42] Trojan.Win32.DINWOD.B - Threat Encyclopedia. (n.d.). https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/trojan.win32.dinwod.b/

[43] Corporation, M. (2009, July 20). Trojan:Win32/C2Lop.P threat description - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/C2Lop.P

[44] TROJ_MALEX.CS - Threat Encyclopedia. (n.d.). https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/TROJ_MALEX.CS/

[45] Corporation, M. (2008, August 31). Win32/Skintrim threat description - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FSkintrim

[46] Trojan:W32/Skintrim — F-Secure Labs. (n.d.). https://www.f-secure.com/v-descs/trojan-w32-skintrim.shtml

[47] [10] Trojan:W32/Skintrim — F-Secure Labs. (n.d.). https://www.f-secure.com/v-descs/trojan-w32-skintrim.shtml

[48] Trojan-Downloader — F-Secure Labs. (n.d.). https://www.f-secure.com/v-descs/trojan-downloader.shtml

[49] Malware Obfuscation: Techniques, Definition & Detection - ExtraHop. (2022, January 21). https://www.extrahop.com/resources/attacks/malware-obfuscation/

[50] TROJ_SWIZZOR.SPS - Threat Encyclopedia. (n.d.). https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/troj_swizzor.sps

[51] Corporation, M. (2009, March 13). PWS:Win32/Lolyda.AA threat description - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/Lolyda.AA&threatId=-2147345828

[52] Corporation, M. (2009, October 20). PWS:Win32/Lolyda.AT threat description - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/Lolyda.AT

[53] Corporation, M. (2006, December 7). Dialer:Win32/DialPlatform.B threat description - Microsoft Security Intelligence. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Dialer:Win32/DialPlatform.B

[54] Malwarebytes. (2023, October 3). Adware.InstallCore. https://www.malwarebytes.com/blog/detections/adware-installcore

[55] Malwarebytes. (2023, August 16). Adware.Multiplug. https://www.malwarebytes.com/blog/detections/adware-multiplug

[56] Bailey, R. (2021, August 14). Trojan.Win32.RegRun. How to Fix Guide. https://howtofix.guide/trojan-win32-regrun/#Technical_details

[57] Macro malware information. (n.d.). https://success.trendmicro.com/dcx/s/solution/000279049?language=en_US&sfdcIFrameOrigin=null

[58] Trojan:W32/VBKRypt — F-Secure Labs. (n.d.). https://www.f-secure.com/v-descs/trojan-w32-vbkrypt.shtml