Malware Analysis using Machine Learning

Wessly Soronellas
Kennesaw State University
Marietta, Georgia, United States
wsoronel@students.kennesaw.edu

Abiola Oni
Kennesaw State University
Marietta, Georgia, United States
aoni7@students.kennesaw.edu

Walter Ivy
Kennesaw State University
Marietta, Georgia, United States
wivy@students.kennesaw.edu

Ginette Messanga
Kennesaw State University
Marietta, Georgia, United States
gmessang@students.kennesaw.edu

Nicholas McLaren
Kennesaw State University
Marietta, Georgia, United States
nmclaren@students.kennesaw.edu

***Abstract*—This paper discusses an overview of malware and uses a dataset to create different machine learning models. The tools used are a combination of Weka and Jupyter with the use of a dataset located at UCI repository.**

Keywords—malware, machine learning, neural networks, data science

I. Introduction

The exponential growth of malware over the past decade, as well as its complexity, have been a constant never-ending challenge to security analysts and the anti-malware community. Malware is defined as any software that is intentionally designed to cause damage to a computer, file, server, client and/or computer network. There are several types of malware including viruses, worms, trojan, ransomware, worms, viruses and many others. Researchers are constantly seeking new and various alternative approaches to detecting malware [8]. Looking back at the history of malware, we cannot but agree that malware threats have been in existence since the creation of computing technology. Creeper worm was the first documented virus in the 1970's and it was a self-replicating malicious software that displayed the message "I'm the creeper, catch me if you can", on the remote systems it has copied itself to. From then on, an industry that would later torment the world was born and researchers as well as security analysts have been fighting this never-ending battle against malware. But as they find ways to boost their defenses, malware developers get more innovative in their obfuscation techniques. Their innovations continue to expand to more sophisticated malware based on the advantages derived from technological advancements including, but not limited to, IOT devices, the internet, smartphones, social media or social networks. However, the more creative they get, the more proactive researchers and security analyst must become to improve their defenses.

The lack of cybersecurity skills and the daunting number of malware attacks organizations face in recent times have led to the realization that traditional malware analysis and detection cannot keep pace with new malware variants. These challenges are an opportunity for machine learning to bring about compelling change to the cybersecurity field due to the large amounts of data it can handle and its velocity [4]. This is why this research paper is very important. There is urgent need to develop faster and more efficient malware detection techniques and tools. Our work focuses specifically on creating a training set of equal instances of malicious files and benign files to create an accurate predictive model using supervised learning.

The goal for this research paper is to provide a multi-step approach to identify malware. It will be based on a supervised learning approach that tries to create an accurate model to classify files/executables. In addition, it will help identify whether a file is corrupted. It is a systemic review of how ML algorithms can be used for malware detection, due to the ability of ML to keep pace with the swift growth of malware. During the experiment, our mission will be to investigate malware behavior and identify file hashes of known malware types, hexadecimal data to identify malware content, and supervised learning techniques to predict malicious behavior.

This paper reviews various instance classifiers and how they perform in malware analysis. Those classifiers explained include: Naïve Bayes, J48, K-Nearest Neighbor (KNN), Support Vector Machines (SVM) and Neural Networks. The tools and datasets used in this paper include: ASM, Metame, VirusTotal, X86 Assembly Guide, Chadwick Malware Detection, Chronical Virus Total, MaleX, Obidump, Cuckoo Sandbox, GNU Binutis Obiectdump and VxHeaven.

This research paper is organized as follows. Section 2 presents a summary of the research literature review. Section 3 provides the malware analysis. Section 4 provides a brief review of the research classification including the origin, efficiency, performance and comments. Finally, Section 5 provides the concluding remarks of this research paper.

II. Literature Review

A. Malware Overview

Information is an extremely valuable and lucrative tool used constantly in society today. The introduction of the internet and cloud computing has allowed information to be created, stored, and shared all around the world in seconds. This is possible thanks to the continuous development of programming languages that can convert coded files into bytecode that is then read by a computer who performs the instructions coded on the file. This amazing invention of computer code/software has also created a new opportunity for developers to generate code for malicious behavior. This malicious code/malware is a continuous NP problem of which there is no method to eliminate the threat/risk completely as of the date of writing this paper; however, new methods are being researched and developed frequently to further advance malware detection. One of the more recent methods used in malware detection is Machine Learning (ML)[9].

ML is trained on data that already exists against a certain classifier, coupled with coded preferences with the aim to

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

accurately classify new unobserved instances. The dataset will contain instances, a single row in a dataset, that contain different characteristics called features. If the learning behavior is supervised, the dataset will have a classifier feature which tells the algorithm what the instance should be classified as. The algorithm will use most of the set to generate a model (training), and the remainder of the dataset is used to test the model (testing) for accuracy. ML in malware uses features obtained through different types of analysis to create models that can accurately classify malicious code. Some of the classifiers used in malware analysis are discussed in this paper along with some examples of malware analysis using different tools and datasets. To distinguish between benign software and malware, programs are analyzed through dynamic and static analysis. These are not the only analysis techniques, but they are by far the most popular among researchers. As perhaps inferred by the names, static and dynamic analysis perform analysis based on a certain state of a program. Dynamic analysis uses a live environment to document how a program behaves in real-time, whereas static analysis relies on program structure to document behavior. There are two types of static analysis: code and non-code based. Code-based can disassemble the program and documents the flow of behavior performed called a control flow graph (CFG) where non-code based disassembles the program into bytecode or raw binary and documents the assembly code function calls where different types of techniques are used to aid in malware classification [19].

While both dynamic and static analysis are effective in classifying malware efficiently, adversaries have used techniques to prevent detection. Adversaries have been able to find different approaches to respond to newer detection methods discussed earlier. For example, to try and limit signature-based detection, obfuscation programs generate different programs with similar functionality but different file hashes; therefore, two programs which might be identical in functionality appear as separate programs based on file signatures. Encryption algorithms are also injected into program code to prevent access to method calls unless decrypted first. Code fluffing also occurs where attackers will put useless functionality in a program to try and evade detection. ML techniques are targeted at dataset poisoning attacks where the dataset contains incorrect classification in the training set leading to inaccurate models. Malware can also identify where it is being run and change behavior based on that information. In addition to analyzing environments, malware can also perform timed processes where no malicious behavior will deploy until a certain point of time. The combination of these techniques in addition to new emerging malware in real-time like zero-day attacks, malware researchers have a difficult task of trying to identify effective ways of accurately detecting malware. [2][18]

B. Machine Learning (ML) Techniques

There are plenty of studies that provide in depth discussion regarding the algorithms and process that classifiers go through in deciding what category an instance is classified as, but for this portion we are going to simply provide brief explanations of the classifiers and how they perform in malware analysis. If you want dive deeper into how a classifier works, please refer to D. Ucci et al. [5] for more info

Naïve Bayes is a simple classifier that assumes that all features/attributes in a dataset are mutually exclusive meaning one attribute does not affect another. While this assumption is almost always incorrect, the classifier still performs surprisingly well in classifying instances. Naïve Bayes looks at the total number of instances in a category given a specific attribute and compares it to the compliment to perform a probability analysis to decide which classification an instance is. For example, in spam classification Naïve Bayes is used to see how many times a specific word is said and compares it to its training data and produces a classification based on that data. Naïve Bayes runs fast but is not as accurate as the other models.

J48 is a decision tree algorithm that uses entropy (how much information a feature produces) to start with the highest entropy first and continues to create nested decision trees until a classification is produced. The best way to think of entropy is by playing 20 questions and trying to use the least number of questions possible to get the correct answers. J48 performs better than Naïve Bayes but is not the most accurate nor the most efficient with larger datasets.

K-Nearest Neighbor (KNN) graphs classifications on a graph based on attributes and of course classification. KNN then is trained to a point where when an unknown instance is introduced, the classifier will analyze the features and classify the instance to the classification that is most like it or its ‘nearest neighbor’. The best way to thin of K-Nearest Neighbor is to try and classify an unknown mammal by using the given characteristics.

Support Vector Machines, like K-Nearest Neighbor, analyze the known instances of the dataset and graph the instances on either a two-dimensional or three-dimensional graph. SVM then calculates a line called a hyperplane to separate the classifications where there is a maximum margin between the nearest instances of both sides. The line is used to classify unknown instances based on which side of the line it is placed. A great way of conceptualizing SVM is to think of the equator around Earth. A country is in either the Northern or Southern hemisphere (sometimes both) depending on which side of the line the country is located. SVM performs some calculations to create a hyperplane that is most efficient in being able to correctly classify instances.

Neural networks are a section of machine learning that takes inputs and outputs and create a model for them using weights and biases in combination with a type of graph called an action function. There are several types of neural networks which are used for different domains of data science, but all of them provide the ability to classify non-linear attributes. This means that a feature will provide a certain amount of influence/bias in a way in which may not be consistent. For example, when studying the efficacy of vitamin gummies, scientists will notice that the vitamins may provide beneficial effects with two gummies but cause harmful effects with five gummies; therefore, there is a point in which the gummies are most effective (two gummies) and then provide an opposite effect or no-effect after. The main structure of neural networks contains hidden layers between the inputs and outputs where the biases and weights are evaluated. The algorithm begins with randomly assigned numbers and uses backpropagation where it adjusts weights and biases based on the observed and expected values. Because all nodes/neurons are interconnected in the hidden layer, calculations may be computationally expensive;

therefore, different parameters can be set to determine when the algorithm should stop whether by reaching a certain number of iterations (epochs) or until a specified accuracy is reached.

C. Tools and Datasets

VirusTotal: VirusTotal is a system that combines over "70 antivirus scanners and URL/domain backlisting services" into one aggregated database that is freely available [20]. VirusTotal is basically a "google" for malware (Google acquired them in September 2012) that has robust reporting abilities that identify common features such as which scanners identified it, file details, how it relates to other malware, and file behavior. This solution can be utilized from their website as well as integrated into other products through application program interface (API) endpoints. [6]

MaleX: This dataset was created by Mohammed et al. [19] in their attempt to generate a large dataset of benign files to attempt statistical generalization. These files will be used to attempt to create models attempting to classify benign code which will be discussed later in the paper. For more information or to view the dataset please visit https://github.com/Mayachitra-Inc/MaleX.

Objdump: This Linux tool is used to identify information about a given file. Objdump will be used in this research as a method of obtaining assembly instructions to identify static behavior of a file which is discussed in the next section.

Cuckoo: Cuckoo is a virtual machine specifically used to analyze and document program behavior. This tool will be used to identify features through dynamic analysis.

VxHeaven: This website is used as an educational site providing historical malware samples for research purposes. These samples will be used to ensure that classifiers are accurate in classifying both old and new malware.

D. Feature Extraction and Explanation

File hashes are groups of data that is placed into a cryptographic hashing algorithm which generates a unique signature that is unique to that specific piece of data. These signatures can be used to validate data used as digital evidence in a criminal investigation as well as performing a checksum on a downloaded program to make sure it downloaded successfully. Because file hashes identify reused code/data, they are extremely useful in identifying malware already discovered and documented; however, file hash functionality in malware detection is limited due to obfuscation efforts performed by adversaries as well as the inability to identify new malicious code introduced (zero-day attack). For this reason, other techniques are combined to create a multilayered approach used for malware detection.

TABLE I. STATIC FEATURES

ASM	Hex Dump	PE Header
Sections Registers OpCodes API calls Keywords	File size Whole binary entropy Entropy variance Entropy range Binary Entropy- median Binary Entropy- maximum Binary Entropy- minimum Binary Entropy- median Binary Entropy- mean	Size of code Number of sections Number of symbols Number of imports Size of code

TABLE II. DYNAMIC FEATURES

Cuckoo
Number of mutex
Number of files read.
Number of files deleted.
Register operations.
Dll libraries loaded.

TABLE III. FEATURE INDEX

Name	Definition
Sections	There are three types of sections in assembly language. The bss section is used for declaring variables. The text section is used for keeping the actual code. The data section is used for declaring initialized data or constants.
Registers	store data elements for processing without having to access the memory.
OpCodes	Instruction machine language.
API Calls	is a process that takes place when you send a request after setting up your API with the correct endpoints.
Keywords	These are standards terms used in the assembly language.
File Size	This is the size of a file usually measured in bytes, kilobytes, megabytes or larger.
Entropy	Randomness collected by operating systems.

Portable Executable	Format is a file format for executables, object code, and DLLs and others used in 32-but and 64-but version of Microsoft Windows OS
Mutex	Object that synchronizes access to a resource.
Register Operations	Store data elements for processing with having to access the memory.
DLL	Dynamic link library that contains code and data that can be used by more than one program at the same time.

III. Analysis

The original dataset obtained from the UCI repository was originally generated from Alberto Redondo and David Rios Insua in their research [2]. While the dataset is located at the UCI Repository [1], Alberto Redondo's GitHub repository contains the code used to acquire files and perform classification. For reproducibility, please visit his GitHub at https://bit.ly/3vMjQlt. For this paper we are only going to analyze the data already gathered as there is limited time and resources available to acquire the necessary amount of data; however, we make suggestions to acquire more data to further research this topic later.

A. Data Preprocessing

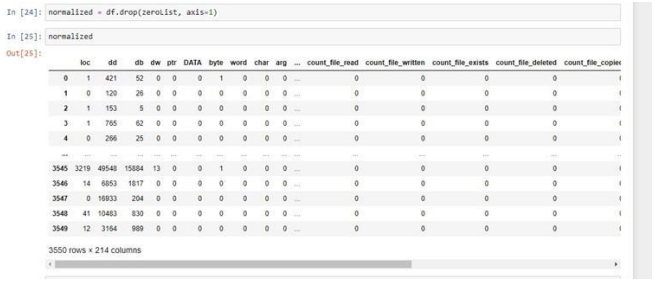
Using the dataset downloaded from UC Irvine Machine Learning Repository we had to perform some preprocessing tasks before building models. After viewing all three files in the directory (VirusTotal, benign, and VxHeaven) we decided to exclude the VxHeaven dataset due to the lack of validation located in the file. Moreover, an untitled column had to be removed from the benign file to correspond to the VirusTotal structure. The filename attribute was also removed due to its ineffectiveness in predicting new instances based on static/dynamic features. Once these changes were applied, the datasets were combined to create a singular dataset of both benign files and VirusTotal files.

After combining the datasets, we wanted to remove any unnecessary columns in the dataset. Because all columns have an integer data type, we decided to perform a sum of all instances in each column; if the sum of the column was equal to zero then that column can be removed. This step was performed in Jupyter Notebook using Python as seen below. After identifying the columns that only contained zeros, we created a new dataset (output.csv) where all columns located in the zeroList are removed. Now the output.csv file has been processed and we can begin creating models.

Print all columns whose sum of all instances equal zero

Append all zero-sum columns to zeroList:

File processed and ready to use.

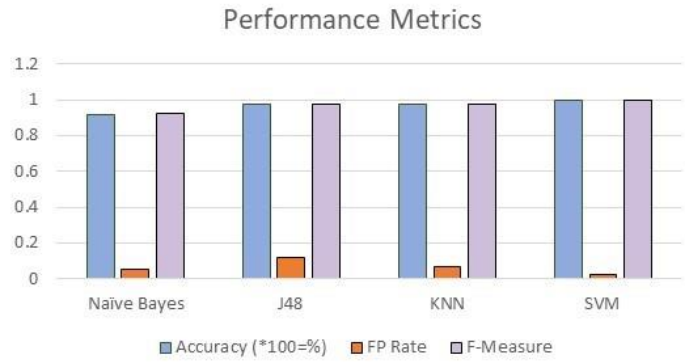


B. Different Classifiers using Weka

After opening Weka and loading the output.csv file, we applied the numeric to nominal filter onto the class label to be able to distinguish between the separate classes. We decided to create four models in Weka: Naïve Bayes, J48, K-Nearest Neighbor, Support Vector Machines. All classifiers use the 10-fold cross-validation testing option selected, and all classifiers are run five separate times and the averages of accuracy, false positive rate, and false negative rate are calculated for each classifier in the table below.

TABLE IV. Performance Metrics

Classifier	Accuracy	FP Rate	F-Measure
Naïve Bayes	91.7%	0.052	0.922
J48	97.4%	0.117	0.973
KNN	97.3%	0.070	0.973
SVM	99.4%	0.022	0.994



C. Building a Neural Network Model using Jupyter

The neural network was built with guidance from a tutorial on freecodecamp's website for reference [11]. The following section builds two models and evaluates their accuracy and loss through graphical representations to help ensure that the models have not been overfitted. The models were built and ran in Jupyter Notebook using Python and various imported ML libraries. The two different models used the same activation functions but differed in the number of total activation functions. In addition, the optimizer algorithm differed between the two models as well. Because there are many different combinations in which to build a neural network, it would not be feasible to configure all of them in the given timeframe; therefore, we used these two models as reference but understand that there may be different configurations which generate more accurate models.

TABLE V. MODEL ACCURACY

Model 1	**Model 2**
96.4%	96.6%

Model 1

Build-

```
from keras.models import Sequential
from keras.layers import Dense
```

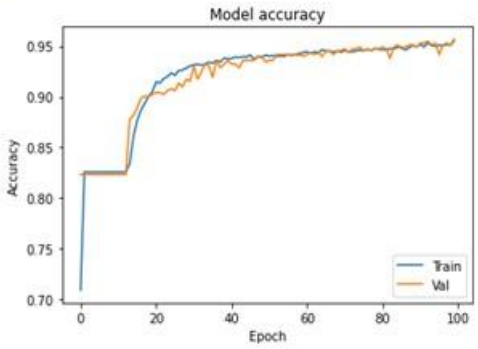
```
model = Sequential([
    Dense(32, activation='relu', input_shape=(213,)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid'),
])
```

```
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
hist = model.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))
```

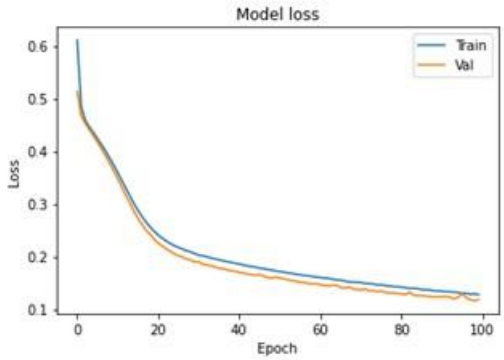
Accuracy-

```
In [28]: plt.plot(hist.history['accuracy'])
         plt.plot(hist.history['val_accuracy'])
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Val'], loc='lower right')
         plt.show()
```



Loss-

```
In [22]: plt.plot(hist.history['loss'])
         plt.plot(hist.history['val_loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Val'], loc='upper right')
         plt.show()
```



Model 2

Build-

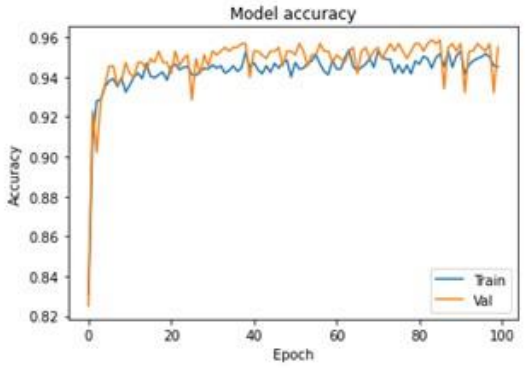
```
model_2 = Sequential([
    Dense(1000, activation='relu', input_shape=(213,)),
    Dense(1000, activation='relu'),
    Dense(1000, activation='relu'),
    Dense(1000, activation='relu'),
    Dense(1, activation='sigmoid'),
])

model_2.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

hist_2 = model_2.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))
```

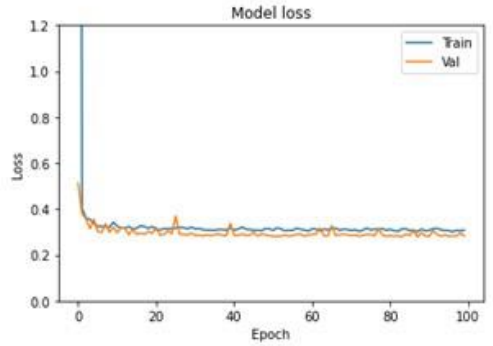
Accuracy-

```
In [36]: plt.plot(hist_3.history['accuracy'])
         plt.plot(hist_3.history['val_accuracy'])
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Val'], loc='lower right')
         plt.show()
```



Loss-

```
In [34]: plt.plot(hist_3.history['loss'])
         plt.plot(hist_3.history['val_loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Val'], loc='upper right')
         plt.ylim(top=1.2, bottom=0)
         plt.show()
```



IV. Review

After performing all the evaluations using the separate classifiers on the dataset there are noticeable differences in performance in each approach. For example, SVM did report the highest overall accuracy; however, the time to build and run the models in Weka using SVM was by far the largest wait time. All other classifiers in Weka ran relatively quick. The neural networks did take slightly longer than the other classifiers in Weka except for SVM. Both neural networks were validated and checked for overfitting whereas the classifiers in Weka were only averaged using 5, 10, and 15 cross-fold validation test options.

In addition to the separate training/testing methods performed there were modifications to the original dataset that was loaded into the UCI Repository to reduce the number of columns to only those whose sum did not equal to zero. This method was aimed to reduce the dataset structure to include only those columns that provide valuable information to the dataset in the effort of increasing efficiency and decreasing computational time.

Due to the limited amount of time and resource, only preliminary analysis was performed on the dataset using the different classifiers. This leaves room for additional research to provide more in-depth analysis of each classifier to recreate and validate these procedures as well as using the revised dataset (output.csv) to create different models.

V. Conclusion

This research provides insight on how to use a combination of malware and benign software analyzed through static and dynamic analysis to obtain sets of features that can be used to correctly classify code as malicious. Although this dataset does provide malware documented and published on VirusTotal, the malicious data obtained from VxHeaven was removed due to its inability to correspond to the benign and malware structures produced from the other datasets; therefore, there can be more research on ways to obtain older viruses such as those located in VxHeaven in a way that corresponds to the other datasets to validate models on older malware examples. Moreover, there is only a limited amount of malware examples produced from VirusTotal which leaves more room for discovering additional malware examples from VirusTotal and other repositories to create a larger dataset.

In addition to the limited number of malware examples available in the dataset, there is also a lack of benign instances as well. While there are other researchers who have successfully gathered large numbers of benign examples such as MaleX, the limited amount of time and resources has prevented this research to be able to use such datasets; therefore, more research can be performed to obtain a larger set of benign code examples in the effort to create a more realistic dataset.

Because malware is becoming increasingly more difficult to identify as discussed earlier, there may be potential in creating a model in which attempts to classify benign software rather than identifying malware. Using a default deny model may be a viable solution to organizations who hold extremely sensitive information. While a default deny model would most likely produce more false negative results (classifying benign software as malware), this method can be implemented as a single layer in a multi-layer approach to malware detection. This way there is a method to which reinforces developers to align with coding standards to try and produce clear concise code that runs efficiently. Should such an approach be implemented and enforced, adversaries would have to increase time, effort, and expenses to try and develop software to act maliciously while simultaneously complying with standards. Though such suggestions are only proposals currently, the current trend of malware evolution leads to a discussion of standardizing developing practices to try and reduce malware risk.

Malware analysis is a cutting-edge field that uses the most recent technological developments in the effort to identify and prevent malware from causing catastrophic damages. Getting hacked is an expensive and lengthy process that most organizations including governing branches will have to deal with at some point their lifetime. Researchers and field experts must think creatively to deliver methods that identify malware before it is too late while also trying to limit the expenses that go into identifying and analyzing malware. For this reason, there should be cooperation throughout the InfoSec community to identify new malware files and behaviors as well as working together to discuss new and innovative techniques to classify malware.

REFERENCES

[1] A. Redondo, "UCI Machine Learning Repository: Malware static and dynamic features VxHeaven and Virus Total Data Set," Malware static and dynamic features VxHeaven and Virus Total Data Set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Malware+static+and+dynamic+features+VxHeaven+and+Virus+Total. [Accessed: 04-May-2021].

[2] A. Redondo and D. R. Insua, "Protecting from Malware Obfuscation Attacks through Adversarial Risk Analysis," arXiv:1911.03653 [cs, stat], Nov. 2019, Accessed: Apr. 16, 2021. [Online]. Available: http://arxiv.org/abs/1911.03653.

[3] A. Shalaginov, S. Banin, A. Dehghantanha, and K. Franke, "Machine Learning Aided Static Malware Analysis: A Survey and Tutorial," arXiv:1808.01201 [cs], vol. 70, pp. 7–45, 2018, doi: 10.1007/978-3-319-73951-9_2.

[4] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," Journal of Network and Computer Applications, 02-Jan-2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804519303868#bbib23. [Accessed: 04-May-2021].

[5] D. Ucci, L. Aniello, and R. Baldoni, "Survey of Machine Learning Techniques for Malware Analysis," Computers & Security, vol. 81, pp. 123–147, Mar. 2019, doi: 10.1016/j.cose.2018.11.001.

[6] F. Lardinois, "Google Acquires Online Virus, Malware and URL Scanner VirusTotal," TechCrunch, 07-Sep-2012. [Online]. Available: https://techcrunch.com/2012/09/07/google-acquires-online-virus-malware-and-url-scanner-virustotal/#:~:text=Google%20Acquires%20Online%20Virus%2C%20Malware%20and%20URL%20Scanner%20VirusTotal,-Frederic%20Lardinois%40fredericl&text=VirusTotal%2C%20an%20online%20malware%20and,was%20just%20acquired%20by%20Google.&text=VirusTotal%20will%20continue%20to%20operate,antivirus%20companies%20and%20security%20experts. [Accessed: 04-May-2021].

[7] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in Android," Journal of Parallel and Distributed Computing, vol. 103, pp. 22–31, May 2017, doi: 10.1016/j.jpdc.2016.10.012.

[8] Heena and B. M. Mehtre, "Advances In Malware Detection- An Overview," arXiv:2104.01835 [cs], Apr. 2021, Accessed: Apr. 16, 2021. [Online]. Available: http://arxiv.org/abs/2104.01835.

[9] I. Firdausi, C. lim, A. Erwin and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection," 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, 2010, pp. 201-203, doi: 10.1109/ACT.2010.33.

[10] "Intel 64 and IA-32 Architectures Software Developer Manual Combined Volumes.pdf." .

[11] J. L. W. En, "How to build your first Neural Network to predict house prices with Keras," freeCodeCamp.org, 17-Mar-2021. [Online]. Available: https://www.freecodecamp.org/news/how-to-build-your-first-neural-network-to-predict-house-prices-with-keras-f8db83049159/. [Accessed: 04-May-2021].

[12] J. M. Spring, A. Galyardt, A. D. Householder, and N. VanHoudnos, "On managing vulnerabilities in AI/ML systems," in New Security Paradigms Workshop 2020, Online USA, Oct. 2020, pp. 111–126, doi: 10.1145/3442167.3442177.

[13] "Malwarebytes-Labs-2019-State-of-Malware-Report-2.pdf." .

[14] M. Ebrahimi, N. Zhang, J. Hu, M. T. Raza, and H. Chen, "Binary Black-box Evasion Attacks Against Deep Learning-based Static Malware Detectors with Adversarial Byte-Level Language Model," arXiv:2012.07994 [cs], Dec. 2020, Accessed: Apr. 16, 2021. [Online]. Available: http://arxiv.org/abs/2012.07994.

[15] R. Naveiro, A. Redondo, D. R. Insua, and F. Ruggeri, "Adversarial classification: An adversarial risk analysis approach," International Journal of Approximate Reasoning, vol. 113, pp. 133–148, Oct. 2019, doi: 10.1016/j.ijar.2019.07.003.

[16] S. Paul and M. Stamp, "Word Embedding Techniques for Malware Evolution Detection," arXiv:2103.05759 [cs], Mar. 2021, Accessed: Apr. 16, 2021. [Online]. Available: http://arxiv.org/abs/2103.05759.

[17] S. Poudyal and D. Dasgupta, "AI-Powered Ransomware Detection Framework," in 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, Dec. 2020, pp. 1154–1161, doi: 10.1109/SSCI47803.2020.9308387.

[18] S. Poudyal, K. P. Subedi, and D. Dasgupta, "A Framework for Analyzing Ransomware using Machine Learning," in 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, Nov. 2018, pp. 1692–1699, doi: 10.1109/SSCI.2018.8628743.

[19] T. M. Mohammed, L. Nataraj, S. Chikkagoudar, S. Chandrasekaran, and B. S. Manjunath, "Malware Detection Using Frequency Domain-Based Image Visualization and Deep Learning," arXiv:2101.10578 [cs], Jan. 2021, Accessed: Apr. 16, 2021. [Online]. Available: http://arxiv.org/abs/2101.10578.

[20] VirusTotal, "How it works," VirusTotal. [Online]. Available: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works. [Accessed: 04-May-2021].

[21] X. Wang and R. Miikkulainen, "MDEA: Malware Detection with Evolutionary Adversarial Learning," arXiv:2002.03331 [cs], Apr. 2020, Accessed: Apr. 16, 2021. [Online]. Available: http://arxiv.org/abs/2002.03331.