Detecting Mac Malware by Implementing Machine Learning Model

Frederick Berberich III

School of Computer Science and Mathematics
Marist University
Poughkeepsie, New York, USA
frederick.berberich1@marist.edu

Daniella Boulos

School of Computer Science and Mathematics
Marist University
Poughkeepsie, New York, USA
daniella.boulos1@marist.edu

Christopher Drisdelle

School of Computer Science and Mathematics
Marist University
Poughkeepsie, New York, USA
christopher.drisdelle1@marist.edu

Dominick Foti

School of Computer Science and Mathematics
Marist University
Poughkeepsie, New York, USA
dominick.foti@marist.edu

Julianna Russo

School of Computer Science and Mathematics
Marist University
Poughkeepsie, New York, USA
julianna.russo3@marist.edu

Abstract—In recent years, the proliferation of Apple Mac malware has raised significant concerns. AV-Atlas, a threat intelligence platform, reports a staggering 122,709,272 new malware and potentially unwanted applications (PUAs) discovered in the past 12 months alone (AV-Atlas, 2024). Additionally, 21 new malware families emerged on MacOS systems in 2023. To address this challenge, one promising solution involves training machine learning (ML) algorithms to detect malware based on behavioral patterns. Unlike traditional signature-based tools, ML models can adapt to the evolving complexity of malware. The research aims to expand upon the earlier work by training ML models specifically on the agent family of malware. This major category includes backdoors, rootkits, and Remote Access Trojans (RATs). Unlike stealer malware, which forwards information to malicious actors, agent-based malware can grant covert system access or siphon sensitive data from target machines. Detecting and mitigating such threats efficiently remains critical for MacOS users. The research group has written a script using *eslogger* (endpoint security logger) to collect data off of a Mac desktop computer, gathering over 20 CSV data files. After successfully parsing and analyzing the data, it was then cleaned and implemented into the K-Nearest Neighbors algorithm (KNN) and XGBoost for the group's machine-learning models.

***Keywords**: Backdoors, Trojans, machine learning, eslogger, SpriteTree, Jupyter Notebook, Google Colab, malware, K-Nearest Neighbors algorithm, XGBoost, malicious, non-malicious, benign*

1 INTRODUCTION

This research project focuses on malware, particularly Backdoor viruses, a type of Trojan, and the minimal ability to detect them on Apple computers. Throughout the project, we ran samples of Backdoor malware which were then collected and analyzed using SpriteTree. This data is used to train a machine learning (ML) model that we designed to differentiate between benign and malicious files and identify logs that may pose a risk to devices. This is done by training the ML algorithm with lesser amounts of data and then assessing it to see if it can predict whether the file is malicious.

1.1 MACOS MALWARE

Malware specific to Apple has existed since the 1980s, with the emergence of "Elk Cloner", the first virus to affect Apple II computers. It displayed the potential for software to spread malicious code, clearing a path for future developments in computer security [1]. Since then, there has been a rise in Apple-specific malware creating a need for improved attack and defense strategies.

In the past, Apple systems were thought of as more secure than Windows for several reasons. One reason is fewer security researchers specialize in macOS than in Windows, resulting in more issues being found on Windows machines. With that, malware writers focus on Windows because there is a higher number of potential systems to compromise. As of April 2024, Windows had a market share of about 72% compared to about 15% for Apple [2]. Apple's share is higher in the small to medium-sized enterprise (SME) vertical, which was about 22.4% as of February 2024 [2]. In 2023, Apple's desktop and laptop operating system, macOS, represents a 31% share of US desktop operating systems, and roughly 25% of all businesses utilize Mac devices somewhere in their networks [3].

Over the years, most of the top 5 computer brands reported a decline in sales, Apple reported a 40% increase [4]. This is a considerable rise in macOS adoption, which will, unfortunately, result in an influx of malware being created for Mac [4]. Being that there are fewer Apple-specific malware researchers, it will allow hackers to take advantage of Apple systems and infect them with a lower chance of them being detected quickly.

Backdoor Malware: Backdoors are a type of trojan malware that disguises itself as a legitimate file or program to access the device. A backdoor grants access to a system by passing the usual authentication measures. They can allow hackers to gain remote access to a Mac and send the data back to the Command-and-Control (C2) server, which the hacker can use to send secondary payloads such as spyware and keyloggers [4]. As with most malware, early detection of backdoors is critical and will increase the probability of being able to protect the system before the attacker gains too much information.

1.2 MACHINE LEARNING

Artificial Intelligence (AI) has become a significant part of the world today, and machine learning is a sub-field of AI that, just like AI, gradually learns. Generally, machine learning algorithms are used to make predictions or classifications, but they must be trained using given data sets. However, there are advantages and disadvantages to machine learning. Machine learning models can identify patterns and trends within large data sets that humans might not necessarily be able to spot on their own. The more data that is fed into the model, the more it can be refined and the more accurate it will be in its predictions. Since the models function better with larger data sets, it needs to be ensured that the data set is accurate and unbiased while also having enough data for the machine learning model to generate accurate predictions [5].

Machine learning algorithms are taught using statistical techniques to produce classifications or predictions [6]. While there are various machine learning algorithms, there are also different machine learning types such as supervised, unsupervised, semi-supervised, and reinforcement learning. This project utilizes two Algorithms for its models. The first model is K-Nearest Neighbors (KNN), a supervised learning algorithm which uses labeled datasets to train algorithms to predict output based on the training. The second algorithm being utilized is a decision tree algorithm called XGBoost. After training the machine with input and output, it uses the test dataset to see if the machine can predict the output [6].

2 PREVIOUS WORK

There is minimal research done on Apple malware, as people have believed it was a lot more secure than Linux or Windows, and even less involving the implementation of a machine learning algorithm. Due to the little research, it is crucial to continuously investigate Mac malware with the implementation of algorithms that can be used to identify the various malware families.

Authors Omar Aslan and Refik Samet, IEEE members, give an overview of various malware detection approaches and explain that malware is increasingly sophisticated and uses techniques such as obfuscation, polymorphism, and stealth to avoid detection [7]. Many machine learning models are used to detect malware including behavior-, deep learning-, and cloud-based detection algorithms. However, there are limitations to each algorithm and difficulty in designing universally effective methods. The article briefly mentions the k-nearest neighbors (KNN) algorithm, which discusses how it is a simple algorithm that performs well with carefully selected features but struggles when handling large data sets [7].

The article “Malware Detection with Artificial Intelligence: A Systematic Literature Review” covers the advances in AI-powered malware detection and highlights challenges like evasion techniques, dataset quality, and generalizability of AI models. The article analyzes static vs. dynamic analysis and their vulnerabilities to obfuscation and anti-analysis techniques [8]. Through their research with AI models, the authors concluded that deep learning typically outperforms machine learning approaches, but that is not to say machine learning approaches are ineffective.

The article “Mac-A-Mal: macOS Malware Analysis Framework Resistant to Anti-Evasion Techniques” discusses the hybrid malware analysis framework, Mac-A-Mal, which is specifically used to detect malware for macOS. Mac-A-Mal is an open-source framework that offers extensive tools for monitoring system behavior, tracking processes, and analyzing system calls [9]. It supports a wide range of file types, excels in detecting child processes generated from macOS’s XPC services, and is capable of withstanding anti-analysis techniques such as virtual machine detection and debugger avoidance [9]. According to the article, Mac-A-Mal successfully detected 71 unknown adware samples, 2 keyloggers, and 1 trojan within the macOS malware OceanLotus.

The article “Mac Malware Detection via Static File Structure Analysis” explains how static analysis focuses on the structural features of executable files and is cheaper and effective against obfuscation. For this study, there were 450 malware samples and 1000 benign samples that were collected from sites like VirusTotal and OS X Mavericks [10]. Five models were assessed including Rotation Forest, Random Forest, PART, Logistic Model Tree (LMT), and a k-nearest neighbors variant called IBk. This study determined that static analysis is effective but not yet production-ready for antivirus purposes.

The article “Effective and Efficient Malware Detection at the End Host” discusses a malware detection approach that combines behavior analysis and efficiency, overcoming the limitations of traditional anti-virus software. It analyzes malware in a controlled environment to create behavior models, which are then used for runtime detection [11]. This method captures characteristic actions of malware to ensure evasion resistance and demonstrates effective detection with low-performance impacts.

The dissertation “An Application of Machine Learning to Analysis of Packed Mac Malware” examines the implementation of supervised machine learning for analyzing packed malware targeting macOS systems. It addresses the growing popularity of malware for macOS, challenges such as obfuscation techniques, and the limited research and resources compared to Windows malware [12]. With a streamlined analysis process, analysts’ mental load can be decreased by automating detection.

The thesis “Machine Learning Methods for Malware Detection and Classification” explores the application of machine learning techniques to improve the detection and classification of malware rather than traditional methods. This research utilizes a dataset of 1,156 malicious files across the nine malware families and 984 benign files, using the Cuckoo Sandbox, with the goal of developing a proof-of-concept machine learning model to classify malware [13].

The article “Enhancing Mac OS Malware Detection through Machine Learning and Mach-O File Analysis” addresses machine learning techniques and proposes an enhanced detection system that expands the scope of analysis. The study tested various algorithms including Random Forest, Gradient Boosting, Support Vector Machines, and Multi-Layer Perception [14]. This research broadened the detection scope to diverse Mach-O file types and identified Random Forest as the top-performing classifier for Mac malware detection.

The chapter “Mac OS X Malware Detection with Supervised Machine Learning Algorithms” from “Handbook of Big Data Analytics and Forensics” discusses the impact of preprocessing, feature engineering, and machine learning algorithms on malware detection performance. The data set included 459 benign samples and 152 malware samples which were tested in 21 algorithms from categories like Decision Tree, SVM, KNN, Ensemble, and Logistic Regression [15]. It concluded that, excluding library-based features, Subspace KNN generated a 90.5% accuracy while including all library features increased the accuracy to 94.7%.

The article “Malware Analysis and Detection Using Machine Learning Algorithms” introduces a machine-learning-based framework to identify malicious software effectively. It evaluates Decision Trees (DT), Convolutional Neural Networks (CNN), and Support Vector Machines (SVM). The dataset involved the use of 17,394 samples resulting in DT outperforming the other algorithms with a 99% accuracy and the lowest false positive rate of 2.01% demonstrating that machine learning algorithms have the potential for highly accurate and efficient malware detection [23].

The paper “Malware Detection System Using Cloud Sandbox and Machine Learning” examines an approach for malware detection by utilizing cloud sandboxing to simulate a secure environment for testing software and employs machine learning classifiers for malware identification. Cloud sandboxing is used because it isolates threats before they reach live networks, collects behavioral data from potentially malicious software, and adds an extra layer of security for online interactions [24]. This research applies to personal devices, organizational systems, and industrial networks and provides insights for real-time detection and mitigation of new malware.

The book “The Art of Mac Malware: The Guide to Analyzing Malicious Software” by Patrick Wardle is dedicated to understanding and interpreting malicious software targeting macOS. It examines various persistence techniques that malware uses to remain on the infected system as well as it explores Mac malware objectives and capabilities including data exfiltration and ransomware activity [25]. The book also discusses anti-analysis techniques that malware authors use and provides an in-depth analysis of EvilQuest malware.

3 METHODOLOGY

To ensure that this project can be replicated, this section describes the methods used to prepare the environment and collect data. It explains how the system was prepared using a dedicated environment, the tools used, and the procedure to generate and collect data.

3.1 LAB ENVIRONMENT

To ensure the proper execution of Mac malware, the project used a dedicated environment involving an Apple Mac Mini 2018 with macOS Sonoma, an Ubiquiti Dream Machine Pro, and Faronics Deep Freeze as well as utilizing various analytical software. A dedicated environment was utilized because it is more reliable when executing malware and collecting benign event logs, considering it is an actual machine and not a virtual one. Along with this, the two previous implementations of this project were done on this dedicated machine, leading us to make our project synonymous with the others. The Ubiquiti Dream Machine Pro consists of VPN support, firewall rules, and a segregated VLAN. Faronics Deep Freeze is a software program that allows for the creation of a safe state to test malware since it allows for information to be redirected rather than written to the hard drive, leaving the original data intact [17]. This testbed allows for the deployment of various MacOS Backdoors so that malicious logs can be collected once the malware is executed without infecting the machine or the network.

Malware databases were used to gain access to backdoor trojans for our research. Four databases were used: Objective See, TheZoo Git Hub, MalwareBazaar, and VirusTotal. Objective-See is a non-profit foundation formed by Patrick Wardle that developed open-source malware detection and prevention tools for the Mac OS. Along with their dedication to creating tools, they have an extensive Mac malware database that includes all types of Mac malware. TheZoo GitHub is also an extensive malware database whose purpose of creation was specifically for malware study and testing. It contains malware of all types, including malware for both Mac and Windows-based devices. Along with TheZoo Github, MalwareBazaar is an extensive database that is dedicated to sharing malware samples with cyber professionals. It contains threat intelligence for each malware sample that is listed. Lastly, VirusTotal is a massive malware database that hosts in-depth analysis about all of the samples of malware listed.

Eslogger was used to collect all the files and folders edited or created by malware and convert this data into JSON files. We analyzed these files using SpriteTree. Once the JSON files were analyzed in SpriteTree, we converted this data into CSV files using a Python script.

SpriteTree was used to analyze the JSON files produced by the malware and eslogger. It allowed us to look at the data logs for each backdoor malware and determine commonalities between them. By looking at each folder and directory, we discovered folders and directories that were edited when the malware was running and learned everything, we could about them.

Jupyter Notebook combined with the Pandas Python library was used to parse our data, combining our CSV files to make up our data frame, which is used in our machine learning code. However, we transitioned to Google Colab, a free, web-based platform, that functions as a cloud-hosted version of Jupyter Notebook. We utilized the sci-kit-learn framework, an open-source machine-learning library for Python, that centers around ease of use, flexibility, and performance. It allows us to create and implement an intelligent ML model, using Python. When the CSV files were combined there were hundreds of thousands of null data generated that had to be removed to allow for more accurate conclusions and predictions in the machine learning model. One-hot encoding was implemented to convert categorical data into a numerical format that can be used in *Singular Value Decomposition* (SVD). It is a method used to decrease the chance of bias in our data, making it easier for the machine learning algorithm to understand [22]. Due to a lack of hardware resources, our research group implemented SVD, because it supports dense and sparse data and provides options for retaining a specific number of components or a target-explained variance.

3.2 DATA COLLECTION

In total, 4 large benign log samples were collected, consisting of 113,989 different events (file 1: 46,008, file 2: 29,666, file 3: 24,561, file 4: 13,754). This is a hefty source of benign events when considering malware samples consist of benign events. Along with the 4 benign logs, 4 raw malicious logs consisting of 2,809 malicious events were captured (file 1: 96, file 2: 60, file 3: 1523, file 4: 1130). Despite this small sample size of benign-to-malicious logs, this mimics the actuality of a system running malware since there are only a few events occurring in comparison to the system.

1) *Capture Command:* The command shown in Figure 1 is the eslogger command used to capture event types such as file renaming, memory mapping, malware detection, process creation, and IPC events. IPC events allow different processes on a computer to run simultaneously. Having "> outputName.json" in

the command allows the captured events to be logged into a JSON file with the specified file name and saved into the user's home directory, following the termination of the command.

```
% sudo eslogger fork exec rename create deleteextattr mmap
profile_add xp_malware_detected btm_launch_item_add
uipc_connect uipc_bind > outputName.json
```

Figure 1. This is the eslogger command utilized to capture security logs on the target Mac device

Table 1: Eslogger Event Types Details

Generic Event Name	Endpoint Security Event Types	
	event_type	*Description*
Fork	es_event_fork_t	Forking of a process
Exec	es_event_exec_t	Execution of a process
Rename	es_event_rename_t	Renaming of a file
Create	es_event_create_t	Creation of a file
Deleteextattr	es_event_deleteextattr_t	Deletion of an extended attribute
Mmap	es_event_mmap_t	Mapping of memory to a file
Profile_add	es_event_profile_add_t	Installing new profiles
xp_malware_detected	es_event_xp_malware_detected_t	XProtect notifications
btm_launch_item_add	es_btm_launch_item_t	Creation of new launch items
uipc_connect	es_event_uipc_connect_t	Connection of a socket
uipc_bind	es_event_uipc_bind_t	Binding of a socket to a path

1) *Procedure:* To collect benign logs, we simulated normal user activity, like running Minecraft or Firefox, to collect logs while normal system functions take place. A variety of activities were conducted on the system by several applications, browser extensions, and games. These applications would mimic normal activity through file system manipulation, system interactions, and regular habits of the average user. This along with the previously mentioned eslogger capture command was utilized to record the related file events and processes that resulted from the activity. To collect malware logs, we used a specific collection procedure.

First, we had to prepare the system for malware execution, which meant disabling System Integrity Protection (SIP) and Gatekeeper. SIP is a security technology designed to prevent potentially malicious code and software from running and protect the modification of file systems [20]. Gatekeeper is a security technology that enforces code signing and verifies downloaded applications before execution ensuring that only trusted software runs on a user's Mac [21]. These tools were disabled to prevent blocking any of the malware's expected activities. They were also disabled so their processes would not appear within the eslogger logs collected.

Next, before executing any malware samples, the Faronics Deep Freeze software was set to its "frozen" state to preserve the system. This step is the most important for our procedure due to its integrity check on our dedicated environment. Then the backdoor malware samples were located and downloaded from the Objective-See website, VirusTotal, MalwareBazaar, and TheZoo GitHub repository. During the execution of the malware samples, there were two terminal windows open, one had the eslogger command capturing the events and outputting them into a JSON file, and the other monitoring the events on the system to make sure that malicious events were occurring from the samples that we ran.

Lastly, once the malware execution was completed, the eslogger tools were terminated and the resulting JSON file was exported to a drive for analysis. In addition to these terminal windows, a third terminal window was employed to bypass other security mechanisms inherent to the macOS system. For example, Mac has a quarantine feature that will display a popup message and block the execution of the malware. To bypass this, a command is executed to disable the quarantine for the specific malware sample being run. This means that this process would have to be conducted for each malware sample. Below is the command that we utilized to bypass this issue.

```
sudo xattr -d com.apple.quarantine <file path>
```

Figure 2. Quarantine removal command bypassing ingrained security function on MacOS

Once the malware sample is executed and the .JSON file is collected, the Mac needs to be reverted to the safe state, which is done through Faronics Deep Freeze software and restarting the Mac. This wipes any system functions or malware that was run and prepares the system for the next malware execution.

4 BACKDOOR MALWARE ANALYSIS

After the logs of the malicious files were captured, we analyzed them in a tool called SpriteTree. SpriteTree is a Mac security tool that takes data generated from, in this case, eslogger and translates it into an easy-to-understand visual format [18]. For any suspicious process we found, we recorded the process ID (PID), the parent process ID (PPID), the responsible PID, and the original PPID. Every process that gets created is assigned a PID, but we look at the PPID when we want to know what process led to the creation of the current process we are examining. The responsible PID refers to the PID of the application responsible for creating or launching a particular process. It is useful for tracking the permissions a given process should have. For example, when a process performs a privileged action, such as taking a screenshot, the operating system looks up the process' responsible PID to see if it belongs to a program that has been granted permission to take a screenshot. The original PPID refers to the initial PPID from which a process was first created. This value remains consistent even if the process' parent changes and it helps track the true origin of a process, making it valuable when conducting malware analysis.

4.1 COMMONALITIES

With all the malware samples being a part of the backdoor malware family, they have many commonalities including creating hidden folders and directories, modifying file and folder locations, modifying timestamps, modifying application permissions, and the AppTranslocation folder. A majority of these are common in most backdoor malware whether it is a persistence method, a hiding technique, or a way to collect and store sensitive data.

As mentioned, one commonality that was found in most, if not all, of our malware samples was the AppTranslocation folder which was found in over 75% of the samples we examined. This folder is part of a security system in macOS known as Gatekeeper Path Randomization or App Translocation. The feature is designed to protect Mac systems by running newly downloaded applications from a randomized path. It helps to protect systems by preventing malicious code from being executed by repacking legitimate applications. Although most of our malware samples were downloaded and run from the AppTranslocation folder, it is not always a signifier of malware, since other applications can be downloaded into this folder.

5 DATA PREQUISITES

Before plugging data into the algorithms associated with machine learning, various steps are required depending on the algorithm you choose. The algorithm our research group chose is K-Nearest Neighbors (KNN). KNN defined by IBM is a "non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point" [19]. Through linear regression and classification, KNN will be able to split our data into two categories: "malicious" and "non-malicious." Malicious will be data that is generated during the execution of a Trojan virus, while non-malicious is data generated from normal activity on the Apple Mac minicomputer. With KNN some prerequisites must be done to the data to make sure it is "clean" and ready to be used for the algorithm. The first step is to convert all our collected logs into CSV files from the collected JSON files. This was done in two separate steps, the first being converting the JSON collected into 'valid' JSON. The files collected do not get formatted correctly for the immediate conversion to CSV due to missing parentheses around the data. The following is a snippet of the code written to reformat the data to 'valid' JSON.

```python
#path to your folder containing JSON files in files on device
folder_path = '/Users/mac_malware_research/Malware_Logs_Collected_Usable'

#iterate through all files in the folder
for filename in os.listdir(folder_path):
    #check if the file is a JSON file
    if filename.endswith('.json'):
        #Construct the full path to the file
        filepath = os.path.join(folder_path, filename)

        #open and read the JSON file
        with open(filepath) as f:
            data = json.loads("[" + f.read().replace("}\n{", "},\n{") + "]")

    #modify the filename to create the new filename
        new_filename= os.path.splitext(filename)[0] +'_valid.json'

    #construct the full path to the new file
        new_filepath = os.path.join(folder_path, new_filename)

    #write the modified data to the new JSON file
        with open(new_filepath, 'w') as new_file:
            json.dump(data, new_file, indent=2)

        print (f"File '{filename}' processed and saved as '{new_filename}'.")

#convert all Malware_Logs_Collected_Usable JSON to csv
```

Figure 3. Reformatting to 'valid' JSON code snippet

After carefully writing code to go through the data and adding parentheses around each data point, the newly formatted JSON files are converted into CSV files. This is also done through a couple of Python scripts. The second step is to combine all the CSV files into one data frame. Generating a large data frame allows us to choose what percentage of the data the algorithm is trained on and what it is supposed to predict. Next, we had to remove all null values generated and saved in the CSV files. Similar to the results of when you divide by zero in a calculator, running machine learning with nulls is impossible and will give you errors.

Then we encoded our data into numerical data through one-hot encoding. Lastly, the data had to be labeled where each event log was marked as malicious or benign, based on the PIDs gathered from the malware analysis phase. This was done by labeling all events with malicious PIDs as one and the remaining logs that are benign as a zero. Code was written to streamline the process of data labeling. A GUI was created to input the file that we would like to label and then the PIDs would be inputted in a comma-separated list for labeling. The program would then create a new column titled "Malicious?" which would then be filled with 1s or 0s depending on if the event was malicious or benign respectively. To do this, it would find column headers with the extension " *_pid " and would search through each event to find if the PID was present.

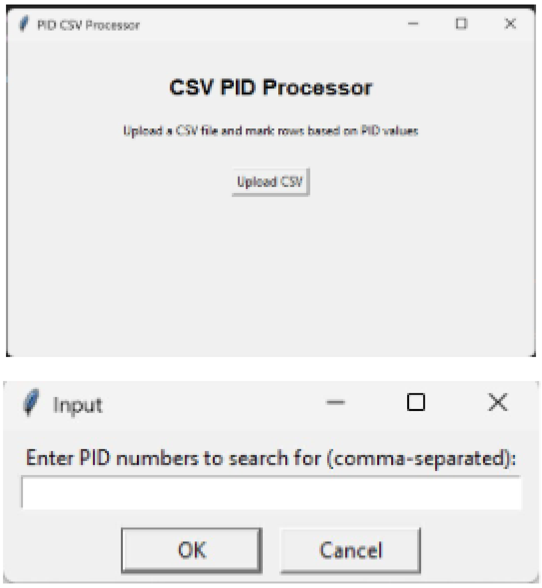

Figure 4. GUI

6 MACHINE LEARNING

Machine learning is defined by IBM as a "branch of artificial intelligence (AI) focused on enabling computers and machines to imitate the way that humans learn, to perform tasks autonomously, and to improve their performance and accuracy through experience and exposure to more data." For our research as mentioned earlier, we are going to implement the data into the KNN and XGBoost algorithms.

6.1 SCI-KIT LEARN FRAMEWORK

For both algorithms, we utilized the Sci-kit learn Python library when creating our machine learning models. The Sci-kit learn framework can be used for classification, regression, and clustering algorithms [16]. It integrates with many other Python libraries and supports vector machines, random forests, gradient boosting, k-means, and DBSCAN [16]. One-hot encoding and PCA are features within the sci-kit learn library and allow us to easily transition from the cleaning of the data to the actual implementation of data to KNN and XGBoost algorithms.

6.2 K-Nearest Neighbors Model

Google Colab was used for both the implementation of our ML algorithm, KNN, and the prerequisites of transformation that our data must go through prior. Google Colab is a cloud system that hosts a *Jupyter Notebook* service, which allows our team to write and edit Python scripts that would impact our data. Before implementing the KNN algorithm, we had to find the "k value." The k value defines how many neighbors will be checked to determine the classification of a specific query point [19]. Not any k value can be used, we had to find the optimal k value which was determined by running the algorithm hundreds of times with different k values.

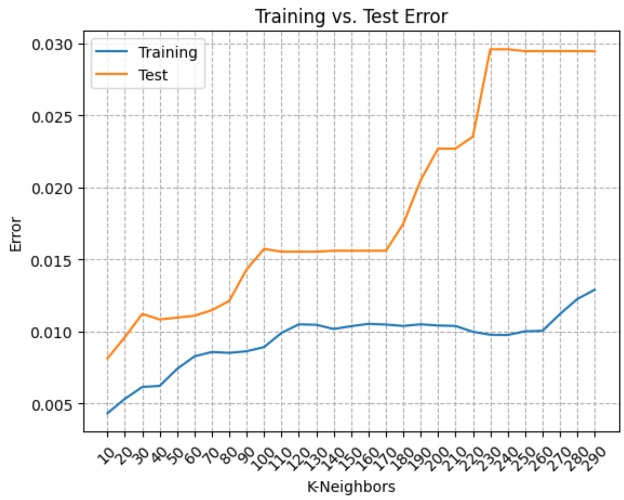

Figure 5. Training vs. Test Error.

Looking at the plot, the optimal k value can be any value where the training line plateaus. A common rule of thumb for estimating a good k value in KNN is the square root of the number of data points in your training set. The code shown in Figure 6 does this calculation, but we find that the K value of 130 might be better since the test and training lines are closer together while plateauing.

```
k_estimate = math.floor(math.sqrt(len(X_train)))
print("We should look around: ", k_estimate)

We should look around:  249
```

Figure 6. Tuning optimal k code.

To train the algorithm, we gave it 25% of the data to learn from and had it predicted the remaining 75%. When examining the chart at Figure 6, we found that we have a data bias of over 50,000 "non-malicious" data compared to the 2,000~ "malicious data".

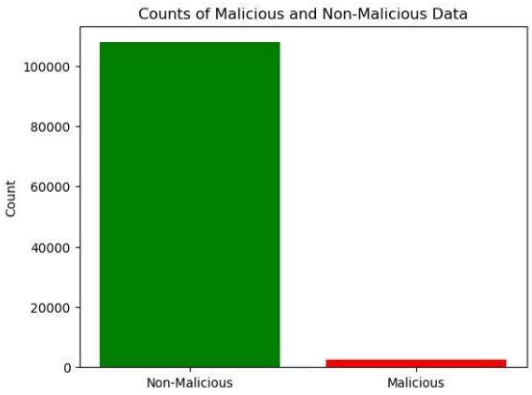

Figure 7. Visual of "non-malicious" data bias.

Our overall model accuracy was 99.25%. Looking at Figure 7, the top box displays the percentage of non-malicious data predicted as 100%, the bottom left box displays that the model missed 22% of malicious data, the top right box displays that the model didn't give any false positives of malicious data and the bottom right box displays that the model predicted 78% of the malicious data.

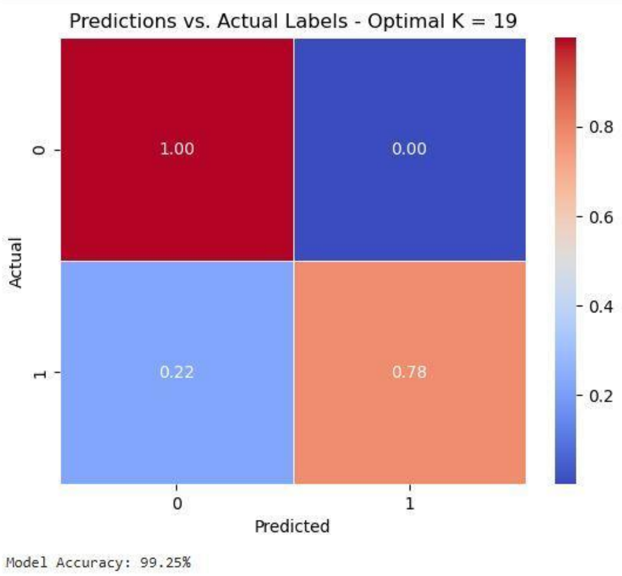

Figure 8. KNN model accuracy.

After seeing how high our model accuracy is, we wanted to test further and see if the data bias has a direct effect on the accuracy. We tested the model again by splitting the data to a perfect 50/50 as shown in Figure 8.

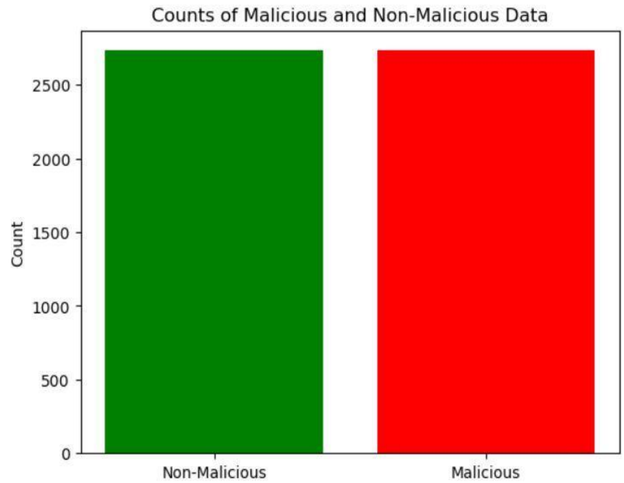

Fig. 9. Equal dataset that was used to train the model.

The new model's accuracy with the equal dataset was 90.85%. Looking at Figure 9, the top left box displays the percentage of non-malicious data predicted as 87%, the bottom left box displays that the model missed 6% of malicious data, the top right box displays 13% of the data as false positive non-malicious, and the bottom right box displays that the model predicted 94% of the malicious data. While this model is less accurate than our initial attempt, it is extremely accurate in predicting malicious data and in general, proves that if we had more "malicious" data we could make our model even more accurate in predicting "malicious" and "non-malicious" data.

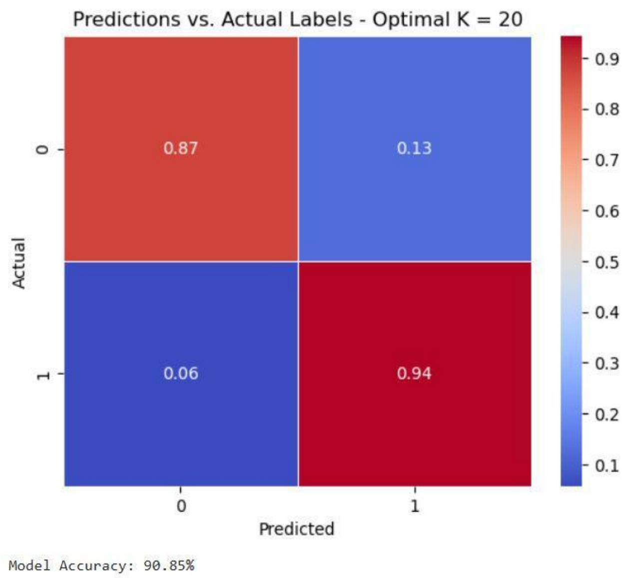

Figure 10. KNN model accuracy with equal dataset.

6.3 XGBoost Model

According to Nvidia, XGBoost stands for Extreme Gradient Boosting, which is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems [26]. XGBoost requires parameters to be set before plugging data into it. We can find these ideal parameters using the code shown in Figure 11.

```
#Run through ALL iterations of parameters to find best performance
xgb = XGBClassifier()
grid_search = GridSearchCV(estimator = xgb, param_grid = param_grid, scoring = 'accuracy', cv = 10)
grid_search.fit(X_train, y_train)

#Print out best parameters
print("Best Hyperparameters: ", grid_search.best_params_)
```

Figure 11. Code that calculates optimal parameters

The values shown in Figure 11 are what we calculated as our ideal parameters for the algorithm, which is calculated by using the data we already prepped so far for the algorithm. This is shown within Figure 12.

```
[ ] #Create a param_grid for hyperparameter tuning
    param_grid = {
        'n_estimators': [50, 100, 150],
        'learning_rate': [0.01, 0.1, 0.3],
        'max_depth': [3, 5, 7],
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0]
    }
```

Figure 12. Code that calculates optimal parameters

After we find our parameters and fulfill all the steps mentioned earlier about what needs to happen to our data prior to plugging it into our algorithms, we can run the model.

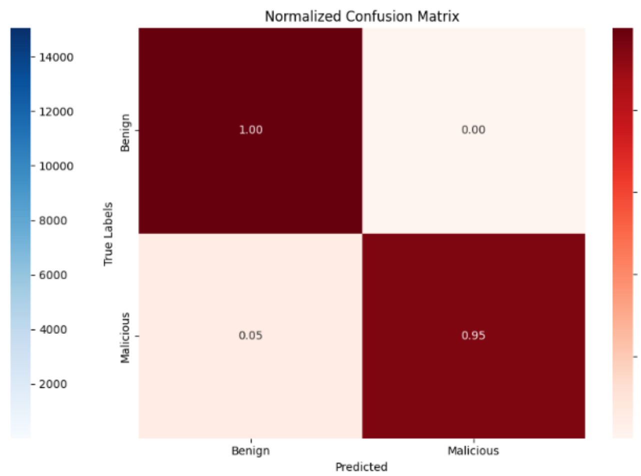

Figure 13. XGBoost model accuracy

After running our model, we can see that the XGBoost model has a higher accuracy than the KNN model. When looking at Figure 13, we can see that the model has a 95% accuracy when predicting malicious samples and a 100% accuracy when predicting non-malicious samples from the model's data frame. While prediction rates vary from each time you run the model, the XGBoost model is consistent with staying above 90% on malicious predictions.

7 CONCLUSIONS

The malware analysis portion of the project exposed valuable insight into some signifiers of malware and commonalities between different backdoor malware files. For example, the AppTranslocation folder, while it is not always a signifier, was found in over half the files analyzed.

While there is still some fine-tuning to be done and more malware samples that can be used to train and test the model, it was exceedingly accurate for the small amount of data used. Both models express high model accuracy with a low false negative rating. Overall, this shows a promising future for the project as these are ideal attributes for a potential malware detection system.

8 FUTURE WORK

Since research on Apple malware and machine-learning implementation is limited, there's room for expansion. In this project, we trained and evaluated our model on a small dataset, though additional collected data was not used. The next step is to assess our model with data that is not part of the model's data frame. Additionally, incorporating more current Trojan samples to refine our dataset is also needed. Lastly, we also plan to fine-tune the model by adjusting test sizes and balancing malicious and benign samples to improve accuracy.

Beyond model optimization, we aim to expand our research to include Spyware, Keyloggers, and Steeler malware. Initial work on Steeler malware was minimal, with few samples and no machine-learning implementation. Moving forward, we will gather a larger dataset and integrate Keylogger data, enabling our model to distinguish between multiple malware types. Ultimately, our goal is to develop a malware detection system tailored for Apple threats.

9 ACKNOWLEDGEMENTS

The following individuals deserve our gratitude for their contributions to this research project. To begin with, we would like to express our appreciation to Professor Dominick Foti for all his support and assistance right from the start of the project. Additionally, we are incredibly grateful to Christian Sarmiento and Nicholas Suchy for their tremendous help with the machine learning aspect and their contributions to developing our models. We want to acknowledge and thank Devin Byrd for his expertise and advice throughout this project. Finally, we want to thank Daisy Kopycienski for her initial research on Steeler malware and for facilitating our further development of the project. We are extremely appreciative of everyone's help and support and look forward to continuing our research and collaborating with other Marist University students in the future.

10 RESOURCES

[1] I. Krastev, "The history of Mac Malware (1982 to 2024)," *SpyHunter for Mac*, Jul. 17, 2024. https://www.spyhunter.com/shm/mac-malware-history/

[2] "MacOS is Increasingly Targeted by Threat Actors," *Intel 471*, Aug. 13, 2024. https://intel471.com/blog/macos-is-increasingly-targeted-by-threat-actors

[3] D. Ruiz, "No 'Apple magic' as 11% of macOS detections last year came from malware," *Malwarebytes*, Mar. 05, 2024. https://www.malwarebytes.com/blog/apple/2024/03/no-apple-magic-as-11-of-macos-detections-last-year-came-from-malware

[4] R. Walsh, P. Bischoff, and R. Walsh, "20+ Mac malware stats and facts for 2024," *Comparitech*, May 20, 2024. https://www.comparitech.com/blog/vpn-privacy/mac-malware-stats-and-facts/

[5] IBM, "Machine learning," *What is Machine Learning (ML)?*, Oct. 28, 2024. https://www.ibm.com/topics/machine-learning

[6] P. Jadav, "Malware Detection Using Machine Learning Techniques," *Einfochips*, Dec. 28, 2022. https://www.einfochips.com/blog/malware-detection-using-machine-learning-techniques/

[7] "A comprehensive review on malware detection approaches," *IEEE Journals & Magazine | IEEE Xplore*, 2020. https://ieeexplore.ieee.org/abstract/document/8949524

[8] M. G. Gaber, M. Ahmed, and H. Janicke, "Malware Detection with Artificial Intelligence: A Systematic Literature Review," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–33, Dec. 2023, doi: 10.1145/3638552.

[9] D.-P. Pham, D.-L. Vu, and F. Massacci, "Mac-A-Mal: macOS malware analysis framework resistant to anti evasion techniques," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 4, pp. 249–257, Jun. 2019, doi: 10.1007/s11416-019-00335-w.

[10] E. Walkup and University of Stanford, "Mac Malware detection via static File Structure analysis." [Online]. Available: https://cs229.stanford.edu/proj2014/Elizabeth%20Walkup,%20MacMalware.pdf

[11] "Effective and efficient malware detection at the end host," USENIX Association, 2009. [Online]. Available: https://www.usenix.org/legacy/event/sec09/tech/full_papers/sec09_malware.pdf

[12] K. Bumanglag, "An application of machine learning to analysis of packed Mac malware," *Beadle Scholar*. https://scholar.dsu.edu/theses/381/

[13] K. Chumachenko, "MACHINE LEARNING METHODS FOR MALWARE DETECTION AND CLASSIFICATION," Bachelor's Thesis, 2017. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/123412/Thesis_final.pdf?sequence=1&isAllowed=y

[14] "Enhancing Mac OS Malware Detection through Machine Learning and Mach-O File Analysis," *IEEE Conference Publication | IEEE Xplore*, Nov. 06, 2023. https://ieeexplore.ieee.org/abstract/document/10478430

[15] S. E. Gharghasheh and S. Hadayeghparast, "Mac OS X Malware Detection with Supervised Machine Learning Algorithms," in *Springer eBooks*, 2021, pp. 193–208. doi: 10.1007/978-3-030-74753-4_13.

[16] "What is sklearn? | Domino Data Lab," *Domino Data Lab*. https://domino.ai/data-science-dictionary/sklearn

[17] Faronics, "About deep freeze." [Online]. Available: https://www.faronics.com/assets/Deep_Freeze_Technical_Datasheet.pdf

[18] Jamf, "Visualizing Endpoint Security Using Apple's 2D Game Framework | JNUC 2023," *Jamf*. [Online]. Available: https://www.jamf.com/resources/videos/visualizing-endpoint-security-using-apples-2d-game-framework-jnuc-2023/

[19] IBM, "KNN," *What is the k-nearest neighbors (KNN) algorithm?*, Oct. 28, 2024. https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20(KNN)%20algorithm%20is%20a%20non,used%20in%20machine%20learning%20today.

[20] "About System Integrity Protection on your Mac - Apple Support," *Apple Support*, Aug. 22, 2023. https://support.apple.com/en-us/102149

[21] "Gatekeeper and runtime protection in macOS," *Apple Support*. https://support.apple.com/guide/security/gatekeeper-and-runtime-protection-sec5599b66df/web

[22] GeeksforGeeks, "One hot encoding in machine learning," *GeeksforGeeks*, Nov. 02, 2024. https://www.geeksforgeeks.org/ml-one-hot-encoding/#

[23] M. S. Akhtar and T. Feng, "Malware analysis and detection using machine learning algorithms," *Symmetry*, vol. 14, no. 11, p. 2304, Nov. 2022, doi: 10.3390/sym14112304.

[24] M. A. E. Mail, M. F. A. Razak, and M. A. Rahman, "Malware detection system using cloud sandbox, machine learning," *International Journal of Computer Systems & Software Engineering*, vol. 8, no. 2, pp. 25–32, Jul. 2022, doi: 10.15282/ijsecs.8.2.2022.3.0100.

[25] "The Art of Mac Malware, Volume 1," *Google Books*. https://books.google.com/books?hl=en&lr=&id=DxZOEAAAQBAJ&oi=fnd&pg=PR17&dq=the+art+of+mac+malware&ots=-GfyFmD5r7&sig=6E0xmyUd4BdSGWbB5trslS5uhVg#v=onepage&q&f=false

[26] "What is XGBoost?," *NVIDIA Data Science Glossary*. Available: https://www.nvidia.com/en-us/glossary/xgboost/