RANSOMWARE DETECTION USING
MACHINE LEARNING AND
PHYSICAL SENSOR DATA

Approved by:

Dr. Mitchell A. Thornton
Professor of Computer Science and Engineering

Dr. Jennifer Dworak
Professor of Computer Science and Engineering

Dr. Theodore Manikas
Professor of Computer Science and Engineering

RANSOMWARE DETECTION USING MACHINE LEARNING AND PHYSICAL SENSOR DATA

A Thesis Presented to the Graduate Faculty of the

Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Master of Science

with a

Major in Computer Engineering

by

Michael Taylor

B.S., Computer Engineering, Southern Methodist University

December 16, 2017

Taylor, Michael B.S., Computer Engineering, Southern Methodist University

Ransomware Detection Using
Machine Learning and
Physical Sensor Data

Advisor: Dr. Mitchell A. Thornton
Master of Science degree conferred December 16, 2017
Thesis completed November 16, 2017

A new method for the detection of ransomware in an infected host during the initiation of its payload execution is proposed and evaluated. Data streams from on-board sensors present in modern computing systems are monitored and appropriate criteria are used that enable the sensor data to effectively detect the presence of ransomware infections. Encryption detection depends upon the use of small yet distinguishable changes in the physical state of a system as reported through on-board sensor readings. A feature vector is formulated consisting of various sensor outputs that is coupled with a detection criteria for the binary states of ransomware present versus normal operation. An advantage of this approach is that previously unknown or zero-day versions of ransomware are vulnerable to this detection method since no a priori knowledge of the malware, such as a data signature, is required for this method to be deployed and used. Experimental results from a system which underwent testing with 18 different test configurations comprised of different simulated system loads unknown to the model and different AES encryption methods used during a simulated ransomware attack showed an average precision of 95.27% and an average false positive rate of 1.57% for predictions made once every second about the state of the system under test.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1. What is Ransomware

Malware is a term that is used to refer to malicious software and is used to refer to all forms of software that can be used to compromise computer functions. This compromise causes harm to the victim computer and ultimately to the user or owner of the host computer. There are a large variety of types of malware including, viruses, worms, adware, bots, rootkits, spyware, trojans, and the primary subject of this investigation, ransomware. Ransomware is a form of malware that holds a victim's computer system files hostage while demanding a ransom to release access to those files back to their legitimate owner.

A typical ransomware attack scenario involves infection of victim computer through penetration of an attack vector whereby the malware resulting from the attack contains a payload that, unbeknownst to the victim, engages in rendering important files as unusable, through their encryption with a key that is unknown to the victim. Upon completion of the initial silent encryption phase, the original unencrypted files are deleted and the victim is alerted that their files are now inaccessible and will remain so until a ransom is paid. It is also often the case, that the attacker will demand ransom within some time period or otherwise the encryption key will be destroyed resulting in permanent loss of the victims data. Figure 1.1 contains a high-level diagram of the chain of events characterizing a typical ransomware attack from the point of view of the adversary.

The largest ransomware attack in history, WannaCry, occurred in May 2017 with 230,000 computers in over 150 countries being infected within a few days. The spread of WannaCry was only halted by a web researcher in England who found a "kill-switch" which was engaged by registering a domain name found in the code [4]. Just one month later, in June 2017, a

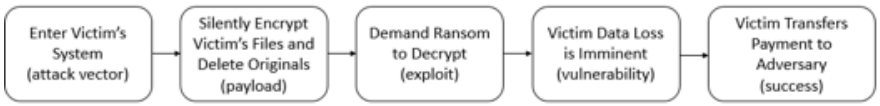

Figure 1.1. Typical Ransomware Attack

ransomware attack known as Petya infected around 16,500 computers globally. One reported instance of the Petya attack requested 100 bitcoins, or about $250,000 dollars, in order to provide the key for decryption [7]. The increase in ransomware attacks has also come with decreased infection times as attackers create new and better methods of file encryption. Testing performed by the cybersecurity company Barkley shows that many ransomware variants complete their encryption phase in under one minute. For instance, Petya finished encryption in 27 seconds and was still slower than Chimera which finished in only 18 seconds. Barkley found that 54% of attacks notified the victim of the ransom within one hour of infection [10]. More detailed information about well-known versions of ransomware, including how they infect computers can be found in [1], [29] and [41].

1.2. How Do Ransomware Attacks Occur

Ransomware attacks can occur through a variety of means. As a more specific example, a common attack vector is the use of email spear phishing where a victim receives an email message that somehow causes the victim to click on an embedded link to a webpage that, in turn, causes the victims browser to display the adversaries webpage to lure the victim into downloading the malware. This malware could be present as a macro contained within an Office document or some other executable. Once the executable is run, it proceeds to encrypt the victims local files in a silent mode and upon completion of the encryption, it then notifies the user that their files are now inaccessible due to the encryption. Next, the exploitation of the victim occurs through a demand of payment. The typical form of ransom payment is through an anonymous transfer of non-traceable funds through the darkweb using electronic currency such as bitcoin. The victim is promised that, if the ransom is received

within a prescribed timeframe, the key will be delivered allowing their encrypted files to be decrypted. While this example scenario is based upon the premise of email phishing as the attack vector penetration method, other means for delivering the malware payload are also possible as well as other varieties of payload activity, victim exploitation, and vulnerability.

1.3. Current Defense Against Ransomware

Effective defense against a ransomware attack is generally considered to comprise a multi-tiered or layered approach [26]. Detection of the malware during the time it is being downloaded to the victim computer is the outer defense, and if possible, can prevent the ransomware from ever entering the system. This defense approach targets prevention of the attack vector from ever penetrating a victims host computer. Packet signature monitoring via an intrusion detection system (IDS) or file signature monitoring via a local antivirus software program can provide this capability, but only if these methods are capable of recognizing the malware through knowledge of the data signatures. While this is a desirable defense, it is notoriously difficult to prevent infection with previously unknown ransomware versions, or so-called zero-day attacks. In the case of zero-day ransomware, data signatures and other corresponding characteristics are unknown by definition. Furthermore, the increasing presence of polymorphic malware is causing signature-based approaches to become less effective than they once were.

If malware penetration is not prevented and the malware manages to be downloaded to the victim machine, the next line of defense is to detect its presence and halt its operation before or at least during the initial stages of victim host file encryption. Recently, an approach has been developed that performs payload detection through monitoring the integrity of victim host file system [37]. This method provides several metrics and indicators that are used to detect the presence of data files that are in the process of being encrypted. One of these metrics is the use of information entropy calculations. Information entropy is a single-valued metric that indicates when a data set has less structure, determinism, and redundancy. The idea underlying the use of an entropy metric is that an encrypted file is one

that closely resembles a file of random data. This is due to the side effect that encryption generally produces data that appear to be random in order to prevent unauthorized decryption through the exploitation of determinism or redundancy in the encrypted file. Therefore an increase in entropy for a given file indicates the high likelihood that the file is being encrypted.

1.4. Utilizing Sensors for Ransomware Detection

Instead of monitoring file system attributes, the victim host system behavior is monitored by taking advantage of the increasingly large number of onboard sensors. In this sense, this new method uses a physical side channel approach where the victims files are not directly monitored, rather the behavior of the victim machine is monitored and onboard sensor provided data is used as side channel information that can indicate when an encryption operation is occurring. This monitoring can be accomplished through a background process that is loaded at boot time and thus continuously monitors the system for suspicious behavior. Once this suspicious behavior is detected, the user can be alerted and the suspicious processes can be suspended. The central difference between this approach and other previous approaches is that this approach uses secondary effects to detect the presence of malware rather than a direct effect, such as measuring increases in file entropy.

Another recent approach for malware detection involves using embedded hardware performance counters that are present in most modern CPU architectures [14] [39]. This approach uses machine learning to create detection models that monitor minor variations in malware execution characteristics. This new approach differs from the use of hardware performance counters in that it uses data being supplied from the suite of embedded sensors that are also present in modern computing platforms rather than performance counter data. Furthermore, this approach is designed to specifically detect ransomware since ransomware uses encryption to enable the victims data files to be held hostage, and hence, allows them to be recoverable when a ransom is supplied in exchange for the decryption key. This approach uses data sources that are secondary to malware execution patterns and it does not rely

upon the presence of performance counters. By targeting a specific class of malware, namely ransomware using encryption in the payload, we can achieve high detection accuracy rates.

It is proposed that this new sensor-based detection methodology be used to complement more traditional signature-based approaches that are intended to prevent attack vector penetration. In contrast to prevention of attack vector penetration, the technique described here is designed to detect the presence of ransomware when penetration has been achieved. The side channel-based or sensor-based approach has an advantage in comparison to antivirus or IDS systems in that zero-day versions of ransomware can be detected since previously captured malware signatures are not required. Furthermore, it is not necessary to monitor individual files and calculate entropy or other metrics that must be continually re-computed and compared with one another as is the case in the solution provided in [37].

An experimental prototype system based on sensor monitoring has been implemented and tested through the use of a variety of scenarios where simulated ransomware is undergoing the silent phase of encrypting victim files. To evaluate this method, five different encryption methods were used from the Python Cryptography Toolkit that have been reported to be commonly used by adversaries during the development of ransomware [6]. Experimental results from a system which underwent testing with 18 different test configurations comprised of different simulated system loads unknown to the model and different AES encryption methods used during a simulated ransomware attack showed an average precision of 95.27% and an average false positive rate of 1.57% for predictions made once every second about the state of the system under test.

Chapter 2

BACKGROUND

2.1. Physical Sensors

Most modern computer systems are comprised of sensors that monitor the state of internal hardware components. These sensors continuously gather and supply information that is communicated with other devices and subsystems within the system for the intended purpose of ensuring that the system stays within specific operating specifications. If sensor data reveals that a system component is approaching a boundary for a recommended value of an operational specification, safety mechanisms will typically be engaged in order to correct the internal environment so that system malfunctions can be prevented. For example, when the data from a temperature sensor of a computers central processing unit begins to increase, a signal is sent to the CPU cooling fan. This signal causes the fan to either become active or to increase the fan speed in order to cool the CPU. Additionally, there are sensors that provide input to other subsystems such as internal power management units, PMUs, to conserve power usage.

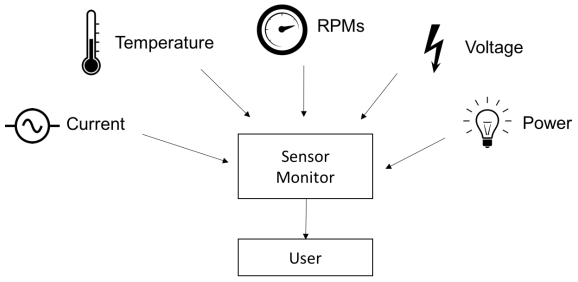

Figure 2.1. Computer Sensor Network

Typically, computer system components are designed to be compact in size through the use of transistors with feature sizing in the nanometer scale. As a direct result, whenever computations become more complex, more stress in placed on a computers hardware components. This increased stress occurs because a large number of transistors are simultaneously switching in a circuit that correspondingly cause an increase in dynamic power consumption resulting in more heat dissipation during heavy computational activity. Thus, monitoring the side channels of a system with embedded sensors that measure temperature, power consumption, and battery voltage levels can give insight into the type of processing that is underway on a computer at a given time. With this thought in mind, it is hypothesized that monitoring a computers side channels through periodic observations of sensor output data could also indicate when a resource-heavy task, such as encryption, is occurring. Since ransomware utilizes encryption in its payload to deny its victims access to their files, analyzing data from a computers side channel sensor data could allow trends to emerge in regard to how a computer behaves while under ransomware attack.

A significant advantage of this approach as compared to other side channel methods is that the sensors and a means for querying them are natively provided. Thus there are fewer concerns in deploying and accessing sensors for the purpose of side channel exploitation. Furthermore, the trend has been that an increasingly diverse number of sensors are provided as integral components in modern computing devices. A typical smart phone has many embedded sensors that could be used to support security applications including power monitors, accelerometers, ambient light sensors, antennas (including GPS receivers), fingerprint scanners, barometers, cameras, touchpad pressure sensors, and others. Even rack-mounted industrial servers contain a significant number of sensors that measure subsystem power consumption, temperature, and other environmental factors. All of these deployed sensors in modern computing devices provide a rich set of data sources that may be used to provide internal side-channel information for the environment in which a computing device is operating. Sensors have been used in other security-related applications in the past. As an example, in [2], sensors present in mobile computing devices have been used to provide a

user demographic classification capability for mobile devices with embedded touchscreens.

2.2. Machine Learning and Statistical Concepts

In the investigation reported here, prediction models were created using machine learning (ML) algorithms. Models are trained using a large amount of data gathered and processed from an experimental environment. It is hypothesized that the sensor data can be used to form a feature vector that differentiates between the binary machine states of normal operation versus ransomware payload execution (i.e., unauthorized encryption activity). The model is trained to weigh the feature vector components with a goal of predicting the machine state with high accuracy. For the implementation described here, several different prediction models are tested which each implement different machine learning algorithms wherein the goal is to discriminate between the binary states of normal operation versus ransomware payload execution.

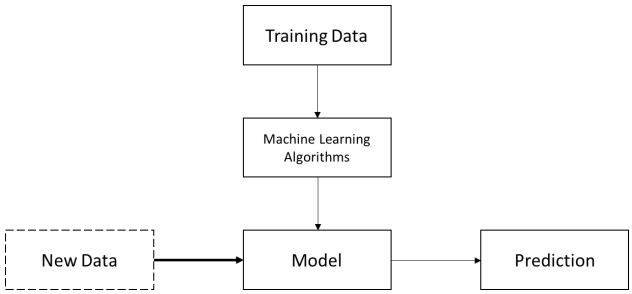

Figure 2.2. Basic Machine Learning Process Flow

The general process for implementing machine learning is displayed in figure 2.2. First, training data is collected which is comprised of independent variable values, and may or may not contain dependent variable values or class membership labels. Machine learning algorithms are applied to the training data which process and fit the training data into a regression or classification model. Once the model is generated, new data is provided which is comprised of values for the same independent variables found in the training data. The machine learning model processes the new input data and provides a real valued prediction

for regression or a class membership label for classification.

2.2.1. Moving Average

Moving average is a calculation to analyze data points by creating a series of averages of different subsets of the full data. One of the most common uses of moving averages is to smooth out short-term changes and emphasize long-term trends in time series data. The simple moving average (SMA) is the simplest implementation of moving average which utilizes the unweighted mean of the previous n data points.

$$SMA_M = \frac{p_M + p_{M-1} + ... + p_{M-(n-1)}}{n}$$

Weighted moving average (WMA) gives different weights to data at different positions in the sample window. WMA allows more recent data to have more impact than previously seen data [15].

$$WMA_M = \frac{np_M + (n-1)p_{(M-1)} + ... + 2p_{(M-n+2)} + p_{(M-n+1)}}{n + (n-1) + ... + 2 + 1}$$

Moving averages are useful when dealing with real time prediction models such as the ones used with sensor data. If a prediction model has been making false predictions for an extended period of time it will take multiple true predictions before the moving average becomes true. Utilizing a moving average may slow the response in reporting real attacks, or true positive predictions. However, the reduction in incorrect attack reports, or false positive predictions, is likely to have more of a positive impact on the prediction accuracy. Moving averages allow the tradeoff between responsiveness and accuracy to be easily adjusted by a user through increasing or decreasing the window size.

2.2.2. Supervised Learning

Supervised learning consists of training data with a set of labeled training examples. Each example is a pair consisting of an input object and an output object. The input object

is the feature vector which is a set of independent measurable properties. The output value is the supervisory signal which is the dependent variable that is being predicted by the feature vector. The supervised learning model generates an inferred function which is used to predict unseen feature vectors. The goal of supervised learning is to make a model from the labeled training set which can accurately predict the class membership of previously unseen feature vectors [27].

2.2.3. Unsupervised Learning

Unsupervised learning consists of determining the hidden structure from a set of unlabeled training data. Each example contains only an input object, or the feature vector, which is a set of independent measurable properties. Unsupervised learning algorithms, such as clustering, attempt to find patterns in the unlabeled data. Clustering creates groups, or clusters, based on the discovered similarities in provided training data. New features vectors can then be introduced into the model and assigned a cluster. This method of grouping similar data may offer insight into useful patterns and relationships which were previously unknown [27].

2.2.4. Model Fitting

Training data is used to fit a prediction model by exploiting relationships between the features. The model should display a high level of accuracy when predicting the same data it was trained with. However, it may be the case that a model is simply unable to capture the relationship between the features in the training data. The model is said to be underfitting when it is unable to perform well with the same data it was trained with. Often underfitting is the product of an oversimplified approach. Conversely, models may be fitted such that they almost perfectly predict the original training data they were presented with, but perform very poorly when presented with new and unseen data. The model is said to be overfitting in this situation. Overfitting is often the product of extreme complexity in an effort to correctly model the original training data. Such complexity can result in highly similar test

data being incorrectly predicted as it isn't perfectly in line with the trends displayed in the training data. Overfitted models essentially "memorize" the training data and offer very little in the way of predictive value for unseen data. Correctly fitting a predictive model involves finding the right balance of complexity which results in relatively high performance with the original training data and also with previously unseen data which is similar [27].

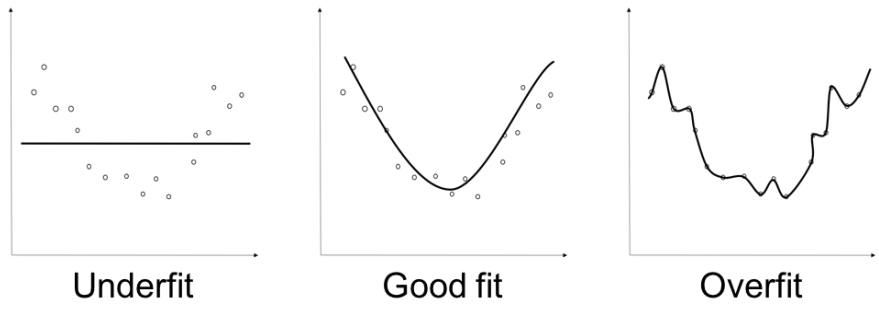

Figure 2.3. Predictive Model Fitting

2.2.5. Dimensionality Reduction

Dimensionality Reduction is the process of reducing the number of independent variables under consideration by obtaining a set of principal variables. Generally, dimensionality reduction is divided into feature selection and feature extraction. Feature selection attempts to find the optimal subset of the original independent variables. Feature extraction transforms the data in the high-dimensional space to a space with fewer dimensions. Dimensionality reduction is advantageous as it reduces the time and storage space requirements, removes multi-collinearity improving the performance of the prediction model, and allows for easier data visualization [23].

2.2.6. Hyperparameters

Hyperparameters are the machine learning algorithm parameters which must be set prior to the learning process. When utilizing machine learning algorithms it is often the case that hyperparameter tuning and optimization is implemented to improve the performance of the

algorithm and reduce overfitting. However, the process of optimization can be expensive both in terms of time and computational resources. When tuning hyperparameters a new prediction model must be generated each time a change is made as changes must occur prior to the learning process. For this reason hyperparameter tuning and optimization should only be carried out on algorithms which have shown the greatest promise for a given problem [18].

2.2.7. Regression

Regression is a method of generating predictions based on the relationship between a set of independent variables, also known as explanatory variables, and one or more dependent variables, also known as the response variable or variables. Regression analysis attempts to model how the dependent variables change when each of the independent variables is changed and the other independent variables remain fixed. The product of regression analysis is a regression function which is comprised of coefficients for each independent variable. Each coefficient represents the mean change in the dependent variable for one unit of change in the independent variable while holding all other independent variables in the model constant. Once the regression equation has been found a feature vector of unseen independent variables can be used to make a prediction for the value of the dependent variable. This is accomplished by multiplying each independent variable measurement by its respective coefficient and adding all of the products together. The resulting prediction value through the use of regression analysis is a real valued number [33].

2.2.8. Classification

Classification is a statistical method of identifying which category an observation belongs based on previously seen training data which contains observations with known category membership. Classification, as it pertains to the application of sensor based prediction, will receive a vector of sensor readings from a system. The sensor vector will be used to determine which category the system currently falls under based on a prediction model. Prediction models can be created using supervised training data which contains many different sensor

vectors and their known category membership. Categories can number as few as two and as many as seen fit to serve the purpose of the application. Therefore, in order for classification to be a useful tool in predicting the state of a system an appropriate machine learning algorithm must be selected, a proper number of categories must be defined, and training data must be collected in a scientific and controlled manner. For the purposes of classifying sensor vectors three different methods of classification will be used. Binary classification, which predicts whether a sensor vector is a member of one of two different categories [33].

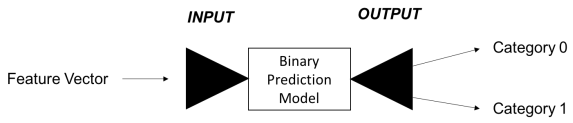

Figure 2.4. Binary Classification Prediction Model

Multiclass classification, which predicts whether a sensor vector is a member of one of three or more different categories.

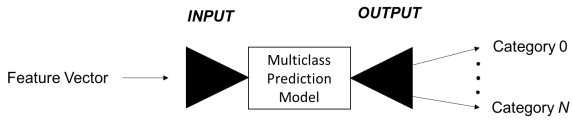

Figure 2.5. Multiclass Classification Prediction Model

Ensemble classification, which utilizes two or more different classification models and determines a single class membership. The final classification prediction is determined by a decision method which accounts for all the individual classification predictions of the prediction models.

2.2.9. Clustering

Clustering is used in unsupervised learning in order to find a structure in a collection of unlabeled data. Clusters can be determined based on a similarity criterion such as distance.

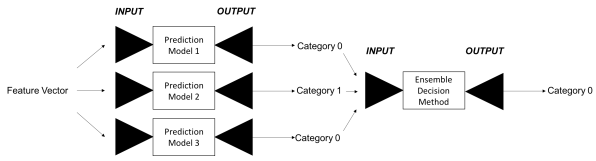

Figure 2.6. Ensemble Classification Prediction Model

In the case of distance multiple objects that are close to one another will naturally form a cluster. Clusters can also be conceptual where clusters are formed based on a common concept among a group of objects. In conceptual clustering, groupings are made according to descriptive concepts rather than simple similarity measures. The ultimate goal of clustering is to determine the intrinsic grouping in the unlabeled data in order to find useful and unusual data [21].

Clustering can be accomplished using several different types of algorithms. Hierarchical clustering is based on the idea that objects located closer to each other are more related than objects located farther apart from each other. Hierarchical algorithms do not generate a single partition and instead generate a hierarchy from which the user still must choose appropriate clusters. Another method of clustering is K-means clustering. In K-means clustering the number of clusters is fixed at k. For each of the k clusters a center point is assigned which is known as the centroid. Each object is assigned to the cluster in which the centroid is closer to the object than any other cluster's centroid [21].

In figure 2.7 the data points are represented in a two-dimensional feature space. The two features, both of which are sensor values, are the independent input. The data points are unlabeled in that they have no known dependent variable value or class associated with them. In the example the number of clusters is set to four resulting in three clusters containing seven data points and the fourth only containing three. Each cluster could potentially represent a system state which was previously unknown or unrecognized.

2.3. Encryption

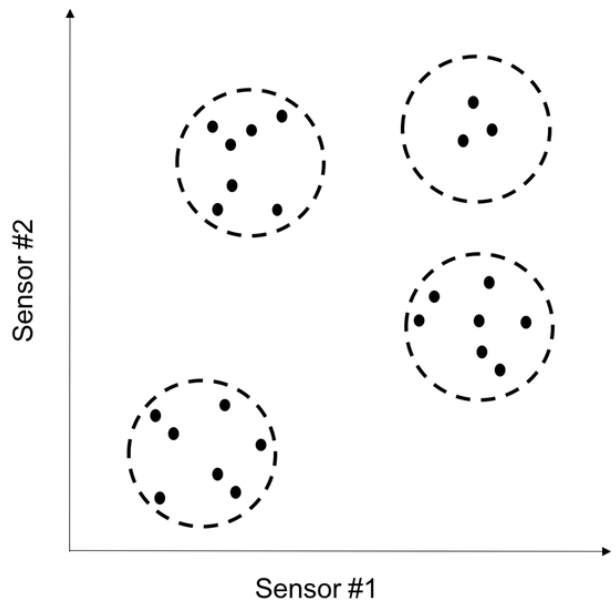

Figure 2.7. Two-Dimensional Clustering Example

Encryption is a process of encoding a message or information in a way that only authorized parties can access it. The message prior to encryption is known as plaintext and the message after encryption is known as ciphertext. Encryption relies on an encryption key which is usually pseudo-randomly generated by an algorithm. While it is possible to decrypt ciphertext without possessing the key, considerable computational resources and a high level of knowledge about encryption schemes are required. Conversely, an authorized recipient can easily decrypt the ciphertext as they possess the encryption key.

Generally, encryption falls into one of two categories, symmetric key or public key. Symmetric key schemes use the same key for both encryption and decryption. The sender and the receiver must both possess the key prior to secure communication. Public key encryption schemes allow anyone to access the encryption key which may be used to encrypt messages. However, only the receiver has access to the decryption key which enables encrypted messages to be read [5].

2.3.1. Advanced Encryption Standard

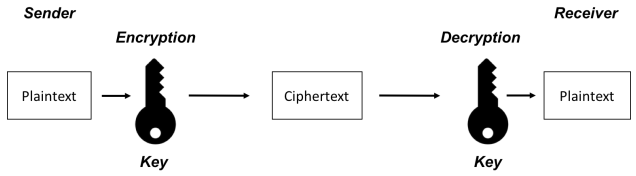

Figure 2.8. Symmetric Key Encryption Scheme

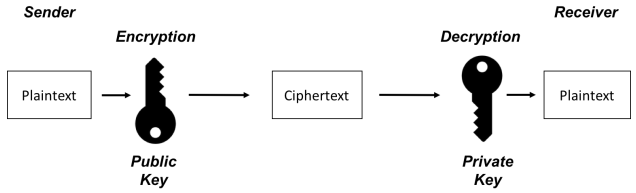

Figure 2.9. Public Key Encryption Scheme

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the United States National Institute of Standards and Technology (NIST). The AES is a subset of the Rijndael cipher which is a family of block ciphers with different key and block sizes. The AES is comprised of three members of the Rijndael family which each have block sizes of 128 bits, but have three different key lengths (128, 192, and 256 bits). AES utilizes a symmetric-key encryption scheme meaning the same key is utilized for encryption and decryption. The United States federal government adopted AES as its standard on May 26, 2002, and it has been approved by the National Security Agency (NSA). Most notably, AES is publicly accessible which allows anyone to utilize the advanced encryption algorithms. AES applies a design principle known as a substitution-permutation network which is a combination of both substitution and permutation [13].

AES operates using a 4x4 column-major matrix of bytes which is called the state. Depending on the size of the key, transformation rounds are repeated a certain number of times which converts the plaintext to ciphertext. Prior to the transformation rounds Rijndaels key schedule is used to expand the key into several different keys called the round keys. If the key is 128 bits 10 rounds are required, if the key is 192 bits 12 rounds are required, and if

the key is 256 bits 14 rounds are required. During each round a number of transformations are performed on the data. First, substitution of data is performed using a fixed 8-bit substitution table. Second, the data rows are shifted cyclically to the left. Third, each column is multiplied with a fixed polynomial. Finally, a simple XOR operation is performed on each column using a different part of the encryption key [28].

Figure 2.10 shows the Southern Methodist University logo after being encrypted with a very simple 16-byte XOR encryption, which is not AES. This encryption scheme is the easiest to implement as it simply performs the XOR operation 16 bytes at a time using a 16-byte key. It is easy to see that the original data on the left could easily be recovered from the encrypted data on the right even without the encryption key.

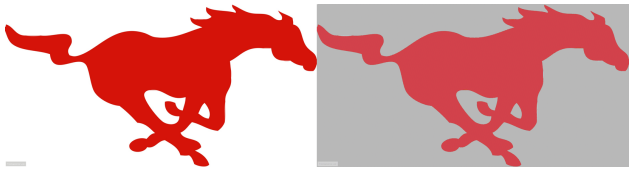

Figure 2.10. XOR Encryption Comparison

The same image encrypted with the Electronic Codebook (ECB) encryption scheme is shown in figure 2.11. ECB encryption is the simplest AES implementation as each block of data is encrypted independently which does little to hide the original pattern of the data. It would still be possible to recover the original data on the left from the encrypted data on the right even without the encryption key.

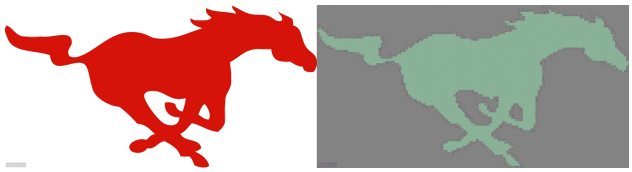

Figure 2.11. Electronic Codebook (ECB) Encryption Comparison

However, more advanced AES implementations like Cipher-Block Chaining (CBC) use techniques which result in encryption that is nearly impossible to recover without the encryption key and a block of bits used to randomize encryption known as the initialization vector. CBC encrypts each block such that it is dependent on the blocks ahead of it. This dependency results in equivalent plaintext blocks becoming different ciphertext blocks which hides the original data pattern as illustrated in figure 2.12.

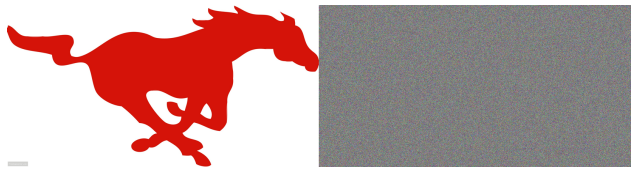

Figure 2.12. Cipher-Block Chain (CBC) Encryption Comparison

There are several AES implementations which result in very sophisticated encryption. Cipher Feedback (CFB) utilizes similar techniques as the ones implemented in CBC to make the block cipher into a self-synchronizing stream cipher. Without the encryption key and the initialization vector recovery of the original data on the left from the encrypted data on the right is almost impossible as can be seen in figure 2.13.

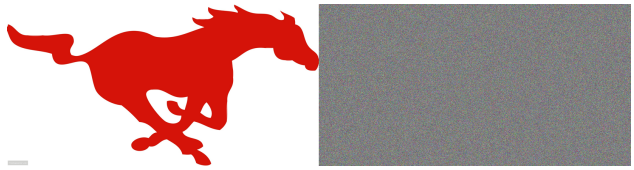

Figure 2.13. Cipher Feedback (CFB) Encryption Comparison

Finally, Output Feedback (OFB) mode makes a block cipher into a synchronous stream cipher by generating keystream blocks which are XORed with the plaintext blocks to make the cipher text. Similar to CBC and CFB, OFB mode block cipher operation depends on all the previous blocks which makes equivalent plaintext blocks into different ciphertext hiding the data pattern as illustrated in figure 2.14.

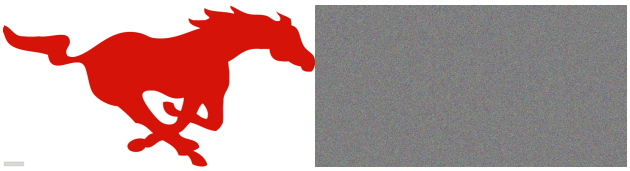

Figure 2.14. Output Feedback (OFB) Encryption Comparison

AES is a popular choice for the encryption stage of ransomware due to its high level of security, lack of known attacks, and wide availability. Many high level packages exist which allow attackers with limited knowledge of encryption to carry out very sophisticated encryption with only a few lines of code. One popular method of encryption is to encrypt a victims files with AES and store the encryption key locally. However, the AES encryption key is encrypted with a separately generated public key. When the victim pays the ransom the attacker releases the private key which decrypts the AES encryption key stored on the victims own computer. It is likely that attackers would implement CBC, CFB, or OFB encryption mode as they are the least likely to be decrypted without the encryption key. However, it is still useful to test simpler methods like ECB and XOR as well due to simple IOT devices likely being encrypted with lightweight encryption methods.

‘

Chapter 3

APPROACH

3.1. Preprocessing

3.1.1. Feature Standardization

Feature standardization is the process of setting each feature of the data to have zero-mean and unit-variance.

$$X' = \frac{X - \frac{\sum X}{N}}{\sqrt{\frac{\sum (X - \bar{X})^2}{N}}} = \frac{X - Mean(X)}{Standard\ Deviation(X)}$$

Every sample of a feature has the mean value of the feature subtracted from it after which it is divided by the standard deviation of the feature. This causes the mean of each feature to be 0 with a standard deviation of 1. It is important to distinguish that feature standardization operates on individual feature columns of a data set [32].

3.1.2. Data Normalization

Data normalization is the process of rescaling each data instance independently such that the L1 or L2 norm is equal to one.

$$L1 = \sum_{i=1}^{n} |y_i - f(x_i)| \qquad L2 = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Scaling the inputs to one, or unit norms, is a common operation when using classification and clustering machine learning algorithms. It is important to distinguish that data normalization operates on the rows of a data set which represent individual data sets of independent variable values [32].

3.1.3. Feature Min-Max Scaling

Feature min-max scaling is a method used to standardize the range of the independent variables or features.

$$X' = \frac{X - Min(X)}{Max(X) - Min(X)}$$

Min-max scaling places all data on the same scale, usually 0 to 1, which in turn allows machine learning algorithms to weigh each feature equally. The standard deviations of the features tend to be smaller with min-max scaling which can suppress the effect of outliers. It is important to distinguish that min-max scaling operates on individual feature columns of a data set [32].

3.1.4. Principal Component Analysis

Principal Component Analysis is a statistical technique utilized to transform a large number of variables, which are possibly correlated, into a smaller number of linearly uncorrelated variables which are then called principal components. Reducing the data set from a large number of variables to a small number of principal components often allows a user to find trends and patterns which may have otherwise not been found [17]. The main goals of Principal Component Analysis are to extract the most important information from a data set, reduce the size of the data by only keeping the most significant information, simplify the description of the data set, and analyze the structure of the observations and variables [35].

The variables in the data set are first standardized so they all utilize the same scale. The covariance matrix is calculated for each variable pairing. The covariance of two variables describes how the two variables move together. When two variables tend to fluctuate in the same manner where their highest values and lowest values fall at the same points they show a positive covariance. Conversely, when the fluctuation of one variable is opposite of another variable where its highest values tend to occur at the other variables lowest values and its lowest values occur at the other variables highest values the variables display a

negative covariance. Eigenvalues and eigenvectors are found utilizing the covariance matrix. The eigenvectors indicate the direction of the principal components thus the original data is multiplied by the eigenvectors to reorient the data onto new axes transforming the data according to the principal components [3].

3.1.5. Feature Selection

Feature selection involves selecting a subset of variables for use in predictive model construction. Feature selection is generally performed in order to decrease the number of features in order to reduce overfitting of the predictive model. Univariate feature selection determines the strength of each individual feature with respect to the response variable. The strongest features are kept for use in constructing the predictive model while the weaker features are discarded. In this way the complexity of the predictive model is reduced and only the strongest variables are used for predictions [34].

Univariate feature selection can be accomplished using several different methods. The simplest method is the Pearson correlation coefficient which is used to measure linear correlation between two variables. Pearson correlation coefficients range between -1 and 1 with -1 being perfect negative correlation, 1 being perfect positive correlation, and 0 being no correlation. The resulting value is found by dividing the covariance of the two variables by the product of their standard deviations. The Pearson correlation coefficient is found for each variable and the response variable. The resulting values are ranked based on their absolute value. Either the top N features are selected or a threshold value is set where only the variables with Pearson correlation coefficient values over the threshold are selected [34].

3.2. Machine Learning Algorithms

The "no free lunch" (NFL) theorem, when applied to machine learning, explains that when no assumptions are made about the data then there is no reason to prefer one algorithm over any other. In other words there is no algorithm which is *a priori* guaranteed to work better than all the rest. It is only possible to determine which algorithm is best by testing

a wide range of algorithms [18]. In this experiment regression, classification, and clustering machine learning algorithms are tested to determine which one most accurately predicts ransomware attacks given sensor training data.

3.2.1. Regression

3.2.1.1. Linear Regression

Linear regression is a statistical method for modeling the relationship between scalar dependent variables and one or more independent variables. The relationship between the dependent and independent variables is modeled using linear predictor functions in which the unknown model parameters are estimated from the data. Linear regression models are most often fitted with the least squares approach which attempts to approximate the solution of a set of equations in which there are more equations than unknowns [9].

Given a data set which contains a dependent variable, y, and a vector of regressors, x_i, linear regression will assume the relationship is linear. Mathematically the model can be expressed as follows:

$$y = \beta_0 + \beta_1 x_0 + \beta_2 x_1 + \ldots + \beta_{p+1} x_p + \varepsilon_i$$

In the above equation y can be labeled the regressand, response variable, or dependent variable. The variable is modeled based on the presumption that the value is directly influenced by the other variables. The vector of independent variables, x_i, can be labeled the regressors, explanatory variables, or predictor variables. The vector of parameters, β_i, can be labeled the effects or regression coefficients. The first regression coefficient, β_0, is the constant offset term after which each element corresponds to a regressor variable. The regression coefficients are the concentration of linear regression and each element can be interpreted as the partial derivative of the regressand with respect to the corresponding regressor. These coefficients are generated from provided training data, prior to making any predictions, in a process which is beyond the scope of this experiment. Finally, the ε_i term is known as the disturbance term which is an unobserved random variable that adds noise to the linear

relationship between the regressand and the regressors. The disturbance term represents all other factors which influence the regressand other than the regressors [16].

Once the model has been fitted and all parameters have been found, predictions in real time are easily accomplished. A vector of regressor values is used as the input and each element is placed in its respective position within the equation. Each coefficient and regressor pair is multiplied and all the product terms in the equation are added together. The result is a real valued prediction for the regressand variable with respect to the regressor vector that was input.

3.2.1.2. Logistic Regression

Logistic regression is a statistical method for modeling the relationship between a binary dependent variable and one or more independent variables. Logistic regression utilizes a binary logistic model which estimates the probability of a binary response based on one or more regressor variables. Logistic regression is used to represent situations in which the dependent variable can have only two possible outcomes. It is useful when making predictions of pass/fail, win/lose, or any other binary relationship. Logistic regression predicts the odds of the outcome being true based on a vector of regressor variable values. The odds are defined as the probability that an outcome is true divided by the probability that the outcome is false.

Logistic regression can be viewed as a special case of linear regression. However, logistic regression makes different assumptions about the relationship between the dependent variable and the independent variables. In logistic regression the conditional distribution is a Bernoulli distribution rather than a Gaussian distribution due to the dependent variable being binary. Additionally, the values being predicted are probabilities and are therefore restricted to a range between 0 and 1. The restriction in range occurs through the logistic distribution function [17].

The logistic function is used in logistic regression for taking any real input, t, and always outputting between 0 and 1.

$$\sigma(t) = \frac{e^t}{e^t+1} = \frac{1}{1+e^{-t}}$$

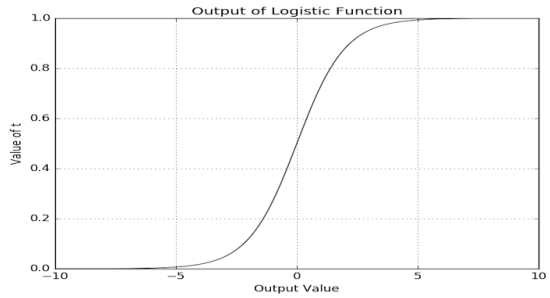

Figure 3.1. Logistic Function Output

The graph in figure 3.1 shows the outputs of the logistic function from -10 to 10. It can clearly be seen that there is an asymptotic boundary at both y=0 and y=1. It is important to note that once the value of t is greater than 5 it will be very close to an output of 1. Conversely, once the value of t is less than -5 it will be very close to an output of 0.

The function for performing linear regression was previously established.

$$y = \beta_0 + \beta_1 x_0 + \beta_2 x_1 + \ldots + \beta_{p+1} x_p + \varepsilon_i$$

Utilizing this linear function with the logistic function, the function for determining the probability of success with logistic regression, F, can be found.

$$F(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-(\beta_0+\beta_1 x_0+\beta_2 x_1+\ldots+\beta_{p+1} x_p+\varepsilon_i)}}$$

The input to the function, x, is a linear combination of multiple regressor variable values. The output of the function can be interpreted as the probability of the dependent variable equaling a success. The odds function can then be found by taking the logistic function and dividing it by its complement. The result can be reduced so that it can be represented as e raised to the power of the linear regression function.

$$ODDS(x) = \frac{F(x)}{1-F(x)} = \frac{\frac{1}{1+e^{-x}}}{1-\frac{1}{1+e^{-x}}} = \frac{\frac{1}{1+e^{-x}}}{\frac{e^{-x}}{1+e^{-x}}} = \frac{1}{e^{-x}} = e^x = e^{(\beta_0+\beta_1 x_0+\beta_2 x_1+\ldots+\beta_{p+1} x_p+\varepsilon_i)}$$

When the odds ratio is greater than 1 the relationship is positive and a prediction of true is most likely to be made. Odds ratio values often exceed 1 which can be interpreted as the strength of the positive relationship where higher values indicated a greater strength. Conversely, odds ratio values that are less than 1 indicate a negative relationship in which case a prediction of false is most likely to be made. The strength of the negative relationship is stronger the closer the odds ratio value is to 0.

3.2.2. Classification

3.2.2.1. Decision Tree Classification

Decision tree classification utilizes a decision tree structure to make predictions about the class membership of a vector of independent variable values. The decision tree contains leaves which represent class labels and branches which represent conjunctions of features. When presented with an input vector of independent variable values, classification occurs by following the branches based on what the respective independent variable values are until a terminal classification leaf is reached [35].

In figure 3.2 a decision tree has been created which predicts whether a student will go to school on a given day based on four binary features. The first branch asks whether the date in question is a weekend. If the date is a weekend the student is most likely not to go to school. If the date in question is not a weekend the decision reaches a second branch. If the date in question is a holiday the student will most likely not go to school. If the date is not a holiday the decision reaches a third branch. If the student is not sick the student is most likely to go to school. If the student is sick on the date in question a fourth branch is reached. If there is a test on the date in which a student is sick they will still most likely go to school. However, if there is not a test on a date in which a student is sick they will mostly likely not go to school.

It can be easily seen that the quality of a decision tree classifier is determined by the training data which is provided. Decision trees are not likely to predict outcomes accurately

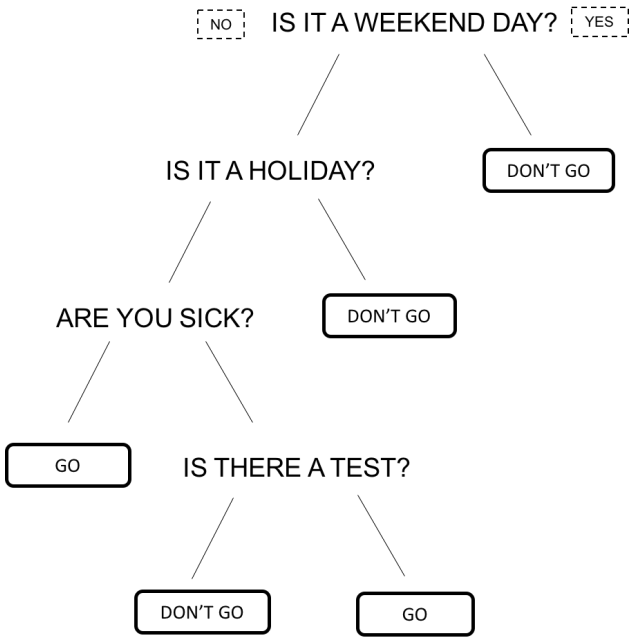

Figure 3.2. Decision Tree Classifier Example

if test data has not previously been seen in the training data. However, if the training data contains all or most of the behavior which will be seen in the testing data it is likely that the decision tree classifier will be very accurate.

3.2.2.2. K-Nearest Neighbor Classification

The k-nearest neighbor algorithm is a non-parametric pattern recognition method used for classification. The algorithm utilizes an n-dimensional feature space where n is the number of features, or independent variables, being utilized. The provided training data is used to determine a classification prediction based on the k nearest training data points to the input vector of feature values. The k nearest data points are polled for their class membership and the class with the majority of neighbors becomes the prediction [3].

In figure 3.3 circles, triangles, and crosses can all be seen. Each of the shapes represents a distinct class which has been plotted in a three-dimensional feature space based on the training data. Generally, the circles are on the bottom, the crosses are in the middle, and

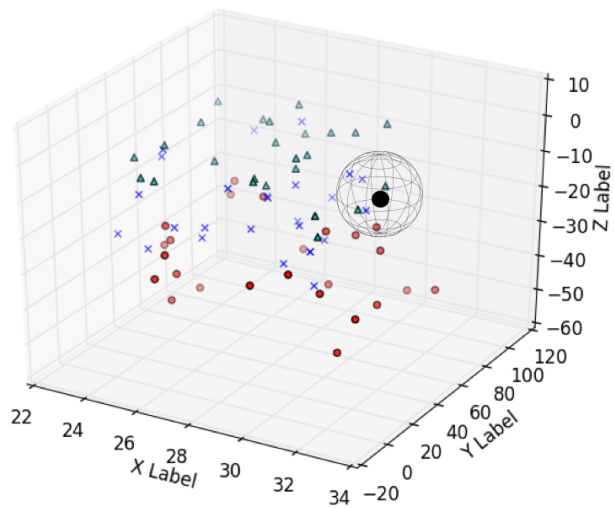

Figure 3.3. Three Class K-Nearest Neighbor Classification Example

the triangles are on top. There is a large black dot representing a test vector of three feature values. In this instance the 6 closest neighbors are found and polled. Within the sphere of nearest neighbors are one circle, two triangles, and three crosses. Based on the k-nearest neighbor algorithm, when k is equal to 6, the classification prediction for the test point will be the same class as the data points represented by crosses.

The k-nearest neighbor algorithm can utilize several different distance metrics for determining the k-nearest neighbors. The most commonly used distance metric is the Euclidean distance.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

3.2.2.3. One-Vs-One Classification

The one-vs-one algorithm is an ensemble classification algorithm which utilizes multiple

binary classification models to determine class membership of an input vector. Every possible pairing of classes has a binary classification model which is trained to distinguish between the two classes and select the one which most closely matches an input vector. After each classification model has made a prediction the class with the most selections is determined to be the classification prediction [32].

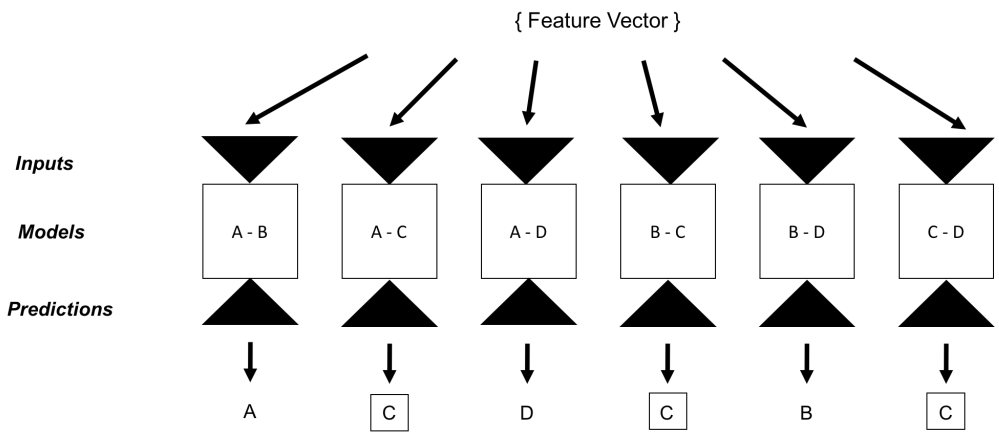

Figure 3.4. Four Class One-Vs-One Classifier Example

In figure 3.4 there are four distinct classes (A, B, C, and D). There exists a prediction model for each combination of classes. There are exactly three models which could result in each distinct class. Each model is provided the input feature vector which consists of independent variable values. The models are trained to determine whether the input vector most closely matches one class or the other. All of the models provide their prediction after analyzing the input feature vector. It can be seen that the class C was positively predicted at each of its prediction models and therefore has the most positive predictions overall. Class C having the most positive predictions makes the final prediction class C.

3.2.2.4. One-Vs-Rest Classification

The one-vs-rest, or one-vs-all, algorithm is an ensemble classification algorithm which trains a binary classification model for each class. When the classification model for a class is trained all samples which are labeled as members of that class are labeled positive and

all samples which are not members of that class are labeled negative. Rather than the individual classification models producing a classification prediction they instead produce a real-valued confidence score. The classification model which reports the highest confidence score is determined to be the prediction for the input vector [32].

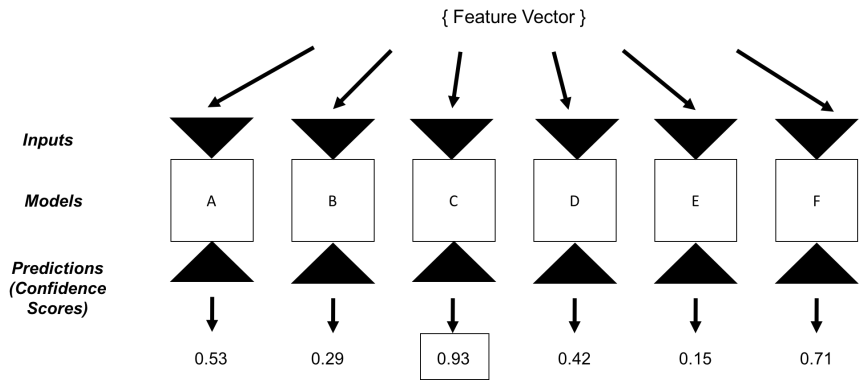

Figure 3.5. Six Class One-Vs-Rest Classifier Example

In figure 3.5 there are six distinct classes (A, B, C, D, E, and F). There exists a separate prediction model for all six classes. For a given model every training data point which belongs to that class is labeled true and every training data point which does not belong to that class is labeled false. Every prediction model is provided the same input feature vector. After analysis each prediction model outputs a real valued confidence score which represents how likely it is that the input feature vector belongs to that particular class. It can be seen that the prediction model for class C provided a confidence score of 0.93 which is the largest of all the models. Therefore, the final class prediction for the input feature vector is class C.

3.2.2.5. Support Vector Machines

Support vector machine (SVMs) are supervised learning models which are capable of performing classification and regression analysis. SVMs, when provided with a labeled training data set, attempt to find a hyperplane in the n-dimensional feature space which maximizes the margin between two classes. Test data is classified based on which side of the classification hyperplane it is on which makes SVMs a non-parametric binary linear classifier [11].

Figure 3.6 shows a two-dimensional feature space with two distinct groups. The black boxes are located in the bottom left while the white boxes are located in the top right. The dotted line shows how a hyperplane can be found that best separates the two groups. If new data points were added to the feature space they could easily be classified based on which side of the hyperplane they were on, bottom left or top right.

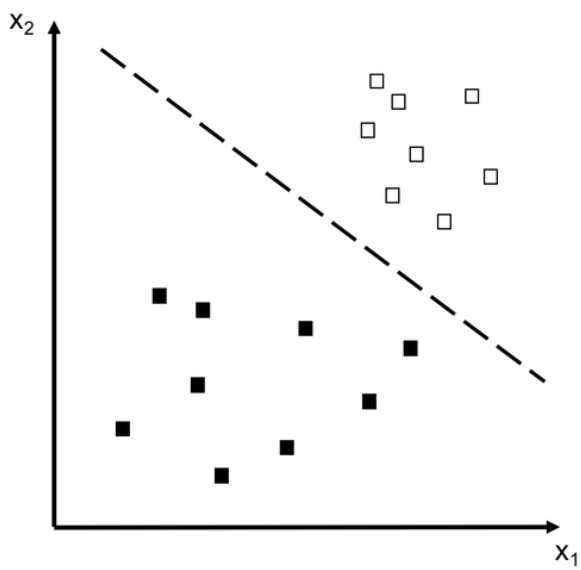

Figure 3.6. Binary Support Vector Machine Classification Example

SVMs can also be used for multiclass classification using techniques such as the one-vs-one classification technique. In the one-vs-one classification technique an SVM will be created for each class pairing. The hyperplane will be found for each SVM which optimally separates the two classes. The class with the most positive predictions among all the SVM models will be the final classification prediction.

3.2.2.6. Naive Bayes Classification

Naive Bayes is a supervised learning algorithm which utilizes Bayes Theorem and posterior probability. Posterior probability is the probability of the parameter θ given the evidence X and is expressed as $p(\theta|X)$.

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

In the above equation $p(c|x)$ is the posterior probability of the class c given the predictor feature x. The class c prior probability is represented by $p(c)$ and the prior probability of the predictor feature x is represented by $p(x)$. The posterior probability of the predictor x given the class c is represented by $p(x|c)$. Naive Bayes operates under the assumption that features are independent of one another.

The naive Bayes model is a conditional probability model. When an input vector is provided with n different independent variable values, the naive Bayes model assigns probabilities for each of the k possible class predictions in C_k.

$$p(C_k|x_1, x_2, ..., x_n)$$

Using Bayes theorem, the conditional probability can be decomposed as follows where the variable x represents the vector of feature values.

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$$

The numerator of the function is the focus of the model due to the fact that the denominator does not depend on C_k and the feature vector, x, is essentially constant. The numerator in the above equation is equivalent to the joint probability model.

$$p(C_k, x_1, x_2, ..., x_n)$$

Utilizing the chain rule, the joint probability model can be rewritten.

$$p(C_k, x_1, x_2, ..., x_n) = p(x_1|x_2, ..., x_n, C_k)p(x_2|x_3, ..., x_n, C_k)...p(x_{n-1}|x_n, C_k)p(x_n|C_k)p(C_k)$$

Naive Bayes classification is naive due to the fact that each feature is considered condi-

tionally independent of every other feature.

$$p(x_i|C_k) = p(x_{i+1}, ..., x_n, C_k)$$

With the application of conditional independence of the features the joint probability model can once again be rewritten.

$$p(C_k|x_1, ..., x_n) = p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

Finally, by taking the above joint probability model and combining it with a decision rule, the naive Bayes classification model can be constructed. The most common decision rule is to select the class which has the highest posterior probability.

$$\hat{y} = \max_{k \in (1,...,K)} p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

Naive Bayes is a relatively fast classification technique which makes it ideal for real time predictions, such as those being made in this experiment [34].

3.2.2.7. Multilayer Perceptron Classification

The multilayer perceptron (MLP) is a type of artificial neural network (ANN). ANNs are a computing system inspired by biological neural networks that comprise organic brains. They are a collection of connected units called artificial neurons which are equivalent to axons in the biological brain. Each connection between neurons, or synapse, can transmit a signal to another neuron. The neuron that receives the signal, or the postsynapse, can process the signal and then signal downstream neurons connected to it. Each neuron may have a real value, typically between 0 and 1, which is called the neurons state. Each neuron and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Additionally, neurons may have a threshold such that the aggregate signal must be either above or below, depending on the implementation, in order for the neuron to propagate the signal downstream [36].

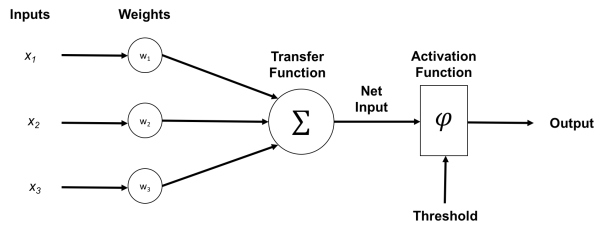

Figure 3.7. Basic Artificial Neuron

The basic neuron in figure 3.7 consists of weights which are multiplied by their respective inputs. All of the weighted inputs are added together at the transfer function, or summing junction, to create the net input. The net input is then subjected to the activation function which is essentially a simple mapping of the net input to the output of the neuron. The activation function governs the threshold at which the neuron is activated and controls the strength of the output signal [36].

Typically, neural networks are organized in layers. The input layer is where patterns are presented which communicate to one or more hidden layers. Hidden layers are where the actual processing is done through a system of weighted connections. Finally, the hidden layers link to the output layer where the final classification is provided. Generally, ANNs contain some kind of learning rule which modifies the weights of the connections.

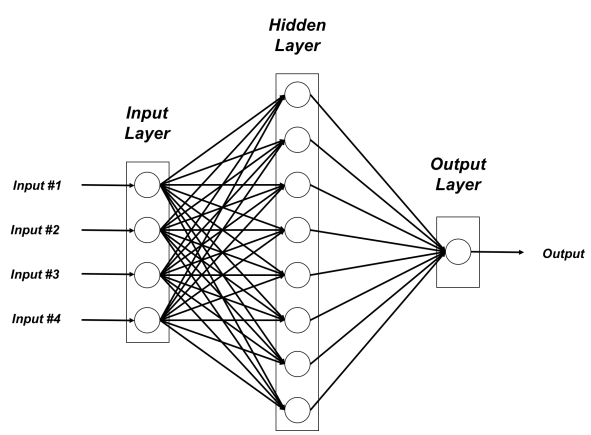

Figure 3.8. Simple Artificial Neural Network

MLPs are a feedforward ANN. All nodes in the model, except for the input nodes, use a nonlinear activation function. Training is supervised and utilizes a technique called backpropagation. In backpropagation the errors propagate backwards from the output nodes to the hidden nodes which are then used to calculate the gradient of the error of the network with respect to the networks modifiable weights. The gradient is then used to find weights that minimize the error, often using simple stochastic gradient descent algorithms. This process allows MLPs to learn from training data as they are constructed [38].

While building and training the MLP may take longer than many of the other prediction models, once the neural network has been constructed and trained it can easily be used for predictions. Inputting a feature vector and performing a forward-pass allows the MLP to generate an output prediction. The MLP can easily be stored for future use by saving the network topology and the final set of weights.

3.2.2.8. Random Forest Classification

Random forest classification is an ensemble classification method which uses multiple instances of a decision tree classifier. The algorithm gets the prediction from each of the decision tree models and selects the mode as the final output prediction. Random forest classification is useful in combating the tendency of decision trees to overfit based on the provided training data. Decision trees that grow very deep have a tendency to learn highly irregular patterns. Random forests average multiple deep decision trees which have been trained on different parts of the same training set. The goal of the random forest is to reduce the variance which comes at the expense of a small increase in the bias and some loss of interpretability [21].

Figure 3.9 shows a small random forest which contains three separate decision diagrams. The decision diagrams are each different due to their being trained with a different subset of the training data. Each decision tree is provided the input feature vector and arrives at a classification prediction. The mode of the classification predictions is provided as the final classification prediction.

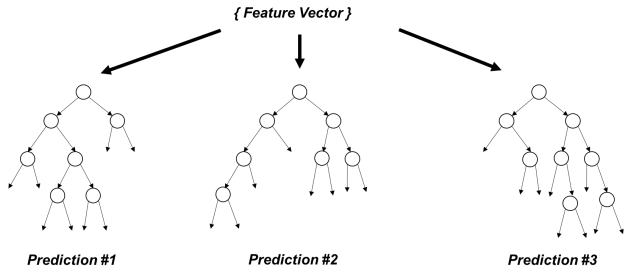

Figure 3.9. Random Forest Classifier Example

3.2.2.9. Extra-Trees Classification

Extra-trees stands for extremely randomized trees or alternately extremely randomized forest. The extra trees method works very similar to a random forest in that it is an ensemble method which utilizes multiple decision trees. However, when splits are determined in the decision trees the thresholds are drawn at random for each candidate feature and the best randomly generated threshold is picked as the splitting rule. Usually this leads to a reduction in the variance of the model over the random forest model at the cost of a slight increase in the bias [19].

3.2.3. Clustering

3.2.3.1. K-Means Clustering

K-means clustering processes unlabeled training data and separates it into K different clusters. Once the previously unlabeled data is assigned a cluster and cluster centers are found new data points can be classified based on their proximity to the cluster centers. This technique is known as nearest centroid classification. The problem is computationally difficult (NP-hard), but different heuristic algorithms are capable of quickly converging to a local optimum [24].

3.3. Performance Evaluation

In order to determine which machine learning algorithms most accurately predict ransomware attacks, metrics must be established and utilized. The desired outcome of a prediction is a binary state which when false represent "normal operating state" and when true represents "under attack". Therefore, basic binary classification metrics are the simplest method for expressing the effectiveness of each machine learning algorithm. Furthermore, as the predictions are made at set time intervals time series plots and metrics offer additional insight into the effectiveness of each algorithm.

3.3.1. Cross Validation

Cross validation is a technique which assesses how accurately a predictive model will perform when presented with an independent data set. In machine learning, a model is given a set of known data which is used for training and a set of unknown data which is used for testing. Cross validation is used in an attempt to test the model during the training phase in order to avoid problems such as overfitting. Cross validation will be performed on each algorithm prior to presenting them with the testing data in order to provide an initial idea of each algorithms effectiveness [36].

3.3.1.1. Holdout Method

The holdout method separates a data set out into two separate sets. One set is the training set and the other set is the testing set. The predictive model is fit with the training data and then the testing data is used to assess the accuracy of the model. This method is the simplest form of cross validation and offers some insight into how well the model performs, but is susceptible to inaccurate results based on how well the data is divided.

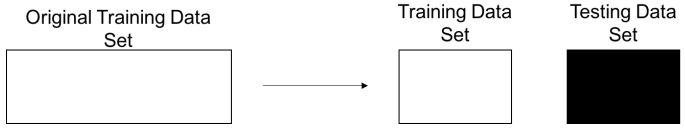

Figure 3.10. Holdout Cross Validation Method

3.3.1.2. K-Folding Method

The k-fold method divides a data set into k subsets where during one of k different trials each subset of data will serve as the training data while the other k-1 subsets are combined together to form the test data set. When working with a classifier, stratified k-fold cross validation, where each fold contains roughly the same proportions of the class labels, is generally preferred. The average error across all k trials is computed and used to assess the accuracy of the model. This method relies less on how well the data is divided as each subset acts as a training set exactly once and acts as part of the test set exactly k-1 times. However, it is computationally expensive due to the fact that k trials must be run.

Figure 3.11. K-Fold Cross Validation Method

3.3.1.3. Leave-One-Out Method

The most extreme method of cross validation is the leave-one-out method. In this method if there are N data points in a data set there will be N different trials. During each trial a training data set is comprised of every data point except for one. The resulting model is then used to make a prediction for the left out data point. The average error across all N trials is used to evaluate the effectiveness of the model [38].

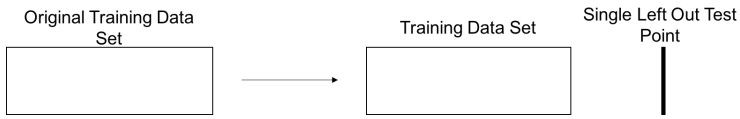

Figure 3.12. Leave-One-Out Cross Validation Method

3.3.2. Binary Classification Metrics

Binary classification predictions can either be positive or negative. Additionally, predictions are either true or false. Classification predictions fall into one of four different categories. True positive predictions are positive predictions that are correctly predicted, and false positive predictions are positive predictions which are incorrectly predicted. True negative predictions are negative predictions that are correctly predicted, and false negative predictions are negative predictions which are incorrectly predicted. These four category values are capable of generating metrics which express the effectiveness of a binary classifier.

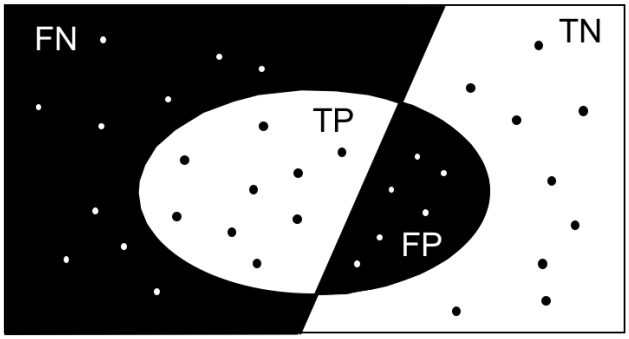

Figure 3.13. Binary Classification Evaluation

In figure 3.13 all of the data points to the left of the diagonal line are positive and all of the data points to the right of the diagonal line are negative. All data points which are located inside of the circle have been positively predicted. Data which is to the left of the line and in the circle is positive and positively predicted making them true positive. Data which is left of the line and out of the circle is positive and negatively predicted making them false negative. Data which is to the right of the line and in the circle is negative and positively predicted making them false positive. Data which is right of the line and out of the circle is negative and negatively predicted making them true negative.

In the context of this study regression, classification, and clustering machine learning algorithms are each used to make a prediction about the binary ransomware attack status of a system. True positive predictions equate to periods of attack which are correctly identi-

fied as being under attack. False negative predictions equate to periods of attack which are incorrectly identified as not being under attack. True negative predictions equate to periods of no attack which are correctly identified as not being under attack. False positive predictions equate to periods of no attack which are incorrectly identified as being under attack. This study focuses mainly on the true positive and false positive predictions. Models which predict a high rate of true positive predictions while maintaining a low rate of false positive predictions are ideal. This would equate to a method which accurately flags instances of ransomware attacks while minimizing flagging instances where no ransomware attack exists.

3.3.2.1. Sensitivity (Sen)

$$\frac{TP}{TP + FN} = \frac{TP}{Positive\ Population}$$

Sensitivity is also known as the true positive rate (TPR). This metric expresses the ability of a binary prediction model to correctly identify as much positive conditioned data as possible in a data set. Sensitivity alone is not a valid measure of a binary prediction model's effectiveness at positive prediction. Simply by predicting every instance as positive, the model will achieve a sensitivity of 100%. Sensitivity must be weighed in conjunction with a model's ability to differentiate positive instances from negative instances, such as precision.

In the context of this study sensitivity equates to the percentage of time intervals that occurred during an attack which were correctly identified as being under attack. This is one of the most important metrics as a high sensitivity means the ransomware attacks are being caught and flagged. However, it is important to show that the model is not simply predicting the system is always under attack in order to catch all instances.

3.3.2.2. Precision (Prec)

$$\frac{TP}{TP + FP} = \frac{TP}{Predicted\ Condition\ Positive}$$

Precision is also known as the positive predictive value (PPV). This metric expresses the ability of a binary prediction model to differentiate data of a positive condition from that of a negative condition. Precision alone is not a valid measure of a binary prediction model's effectiveness at positive prediction. Precision can be very high by only positively predicting instances with a very high confidence score. However, the model may miss a very high number of positive instances which possess a lower, but still sufficient, confidence score. Precision must be weighed in conjunction with a model's ability to correctly predict the largest number of positive instances in the data set, such as sensitivity.

In the context of this study precision equates to the percentage of time intervals that were correctly predicted as being under attack out of all the time intervals which were predicted as being under attack. High precision means the ransomware attacks are being caught and flagged only at times when the system is actually under attack. However, it is important to show that the model is not missing attacks by being too selective with positive predictions.

3.3.2.3. Specificity (Spec)

$$\frac{TN}{TN + FP} = \frac{TN}{Negative\ Population}$$

Specificity is also known as the true negative rate (TNR). This metric expresses the ability of a binary prediction model to correctly identify as much negative conditioned data as possible in a data set. Specificity alone is not a valid measure of a binary prediction model's effectiveness at negative prediction. Simply by predicting every instance is negative the model will achieve a specificity of 100%. Specificity must be weighed in conjunction with a model's ability to correctly predict positive instances, such as the sensitivity value.

In the context of this study specificity equates to the percentage of time intervals that occurred without an attack which were correctly identified as not being under attack. High specificity means that periods of no attack are being correctly recognized. However, it is important to show that the model is not simply predicting that the system is never under attack in order to keep from issuing any false alarms.

3.3.2.4. Fallout (Fall)

$$\frac{FP}{TN + FP} = \frac{FP}{Negative\ Population}$$

Fallout is also known as the false positive rate (FPR). This metric expresses the rate at which a binary prediction model incorrectly identifies negative conditioned data as positive conditioned data. Fallout alone is not a valid measure of a binary prediction model's inefficiency at positive prediction. Simply by predicting every instance is negative the model will achieve a fallout of 0%. Fallout must be weighed in conjunction with a model's ability to correctly predict positive instances, such as the the sensitivity value.

In the context of this study fallout equates to the percentage of time intervals that occurred without attacks which were incorrectly identified as being under attack. This is one of the most important metrics as a low fallout means the periods of no attack are being kept free from false alarms. However, it is important to show that the model is not simply predicting the system is never under attack in order to keep from issuing any false alarms.

3.3.2.5. Accuracy (Acc)

$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{Total\ Population}$$

Accuracy is a metric which expresses the ability of a binary prediction model to correctly identify as much data as possible in a data set. Accuracy may not always be a true representation of the ability of the model to make correct classifications. For instance, if the data set was negatively conditioned 95% of the time then always predicting negatively will results in a high accuracy of 95%. However, the model has no predictive value as it is incapable of making positive predictions. Accuracy should be weighed together with sensitivity and specificity in order to show that both classes are being accurately predicted.

In the context of this study accuracy equates to the percentage of time intervals which were correctly predicted as being under attack or not under attack. High accuracy means that

the prediction model is correctly identifying the state of the system based on the provided sensor data. However, it is important to ensure that the prediction model is not always predicting the system is not under attack as attacks account for far less of the testing time.

3.3.2.6. F_1 Score (F_1)

$$2 * \frac{1}{\frac{1}{sensitivity} + \frac{1}{Precision}}$$

The F_1 score is an alternative measure of a binary classification model's accuracy. Rather than finding the percentage of true predictions over the total population of the data, the harmonic mean of the precision and sensitivity are found. sensitivity measures the amount of positive conditioned data correctly identified and precision measures the percentage of positive predictions which were correct. Therefore, the F_1 score weighs both the amount of correct predictions and the predictive success of correct predictions in order to determine the accuracy of the model.

In the context of this study the F_1 score is a valuable measurement of the prediction models ability to correctly predict as many intervals occurring during an attack while also maintaining a high percentage of correct positive prediction. The F_1 score, being a single value, offers quick insight into the effectiveness of a prediction model.

3.3.2.7. Matthews Correlation Coefficient (MCC)

$$\frac{(TP * TN) - (FP * FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The Matthews correlation coefficient (MCC) takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes such as the data in this experiment. The MCC is a correlation coefficient between the observed and predicted binary classifications. Values range between -1 and +1. A coefficient of +1 represents a perfect predictor, 0 represents the

same as random prediction, and -1 indicates total disagreement. The MCC is regarded as being one of the best measures for describing the relationship between the four possible binary classification outcomes in a single value. Thus, the MCC will be the most heavily weighed metric in determining the optimal machine learning algorithm for the problem presented in this experiment.

3.3.3. Time Series Metrics

3.3.3.1. Rate of Attack Recognition (RAR)

When testing, an actual attack time series exists which defines the actual time periods during which there is an attack. Figure 3.14 displays an example time series plot with the actual attack time series located on the far left. For each time interval occuring during an attack in the actual attack time series, or every time interval after the rising edge and before the falling edge, is checked in the corresponding prediction time series for positive predictions. If a positive prediction exists then the attack instance is considered detected. Ideally there should exist at least one positive prediction during each attack instance which would result in a 1.0 or perfect rate of attack recognition. In the worst case scenario there would exist no positive predictions during any attack instance which would result in a 0. Unlike traditional binary classification metrics, such as sensitivity, rate of detection is not concerned with the volume of correct positive predictions during an attack instance. Instead it considers the application of the binary classifier in which one positive prediction would perform the same as a high volume of positive predictions as the attack only has to be flagged once. It should be noted that the rate of attack recognition will always be 1 if all predictions are positive. Therefore, rate of attack recognition must be weighed in conjunction with a binary classifiers ability to make correct positive predictions such as precision.

3.3.3.2. Mean Time to Attack Recognition (MTAR)

The rising edge of each attack instance in the actual attack time series represents the

time interval at which the attack began. The first instance of a positive prediction at or after the rising edge and before the falling edge in the corresponding prediction time series represents the initial attack recognition. Ideally the rising edge itself would be a positive prediction, but in practice it is more likely that the sensors would need a small amount of time to reach the values at which positive prediction occurs. The number of time intervals until the first positive prediction is recorded for every attack instance which was successfully recognized. Afterwards all values are averaged to determine the mean time to attack recognition. It should be noted that attack instances which were not successfully recognized do not weigh negatively in this metric. Therefore, mean time to attack recognition should be considered in conjunction with the rate of attack recognition to describe the effectiveness of detecting attack instances both quickly and consistently. Furthermore, the mean time to attack recognition will always be 0 if all predictions are positive. Thus, mean time to attack recognition must also be weighed in conjunction with a binary classifiers ability to make correct positive predictions such as precision.

3.3.4. Graphs and Diagrams

3.3.4.1. Time Series Plot

Predictions in this experiment are made at intervals in time. During the time interval the system may either be under attack or not under attack. The machine learning algorithm must make a prediction for each time interval based on the sensor data provided at that time.

In figure 3.14 there is an example time series plot with the left most plot displaying the actual periods of attack, the middle plot displaying the periods of predicted attack, and the right most plot displaying the periods of predicted attack with a moving average applied. When comparing the middle plot to the left plot there is obvious noise which is most often single incorrect predictions. The right most graph removes most of the noise and is a very close match to the plot on the left. However, simply looking at the statistics show that the

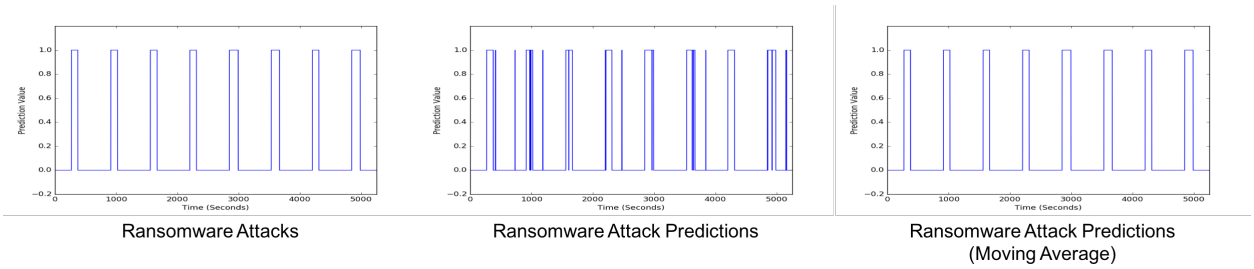

Figure 3.14. Example Time Series Plot

middle plot has an accuracy of 99.4% and the right plot has an accuracy of 99.3%. The moving average causes a small delay in the initial positive prediction and also in the initial negative prediction of each attack instance. The time interval plots show that having more inaccurate predictions can result in a better model when they are located strategically. This conclusion would be much more difficult to draw from the statistics alone.

Chapter 4

IMPLEMENTATION

4.1. Test Equipment

4.1.1. Hewlett Packard ENVY m4-1015dx

The Hewlett Packard m4-1015dx is the first system which will be analyzed in this experiment. It can be seen in table 4.2 that when using Open Hardware Monitor the HP ENVY laptop returns twenty two sensor values. The types of sensor values which are returned include clock, data, load, power, and temperature. Desired sensor types where outlined in chapter 2 and do not include clock, data, or load. Thus the HP ENVY laptop only contains nine sensor values which are of use in this experiment. Furthermore, of the nine sensors that are of use, eight measure aspects of the CPU making the predictive model nearly entirely dependent on the CPU. Therefore, the HP ENVY laptop can be seen as a poor implementation of the sensor based ransomware detection method presented in this experiment.

4.1.2. MacBook Air 13-Inch Mid 2013

The MacBook Air 13-inch mid-2013 model is the second system analyzed in this experiment. It can be seen in table 4.4 that when using Hardware Monitor for Mac the MacBook Air laptop returns sixty nine sensor values. There is only two sensor values reported which do not fall under the desired sensor types outlined in chapter 2. Thus, the MacBook Air contains sixty seven sensor values which are of use in this experiment. Unlike the Hewlett Packard ENVY m4-1015dx, only seventeen of the available sixty seven sensors measure CPU activity resulting in predictive models which are more robust. Therefore, the MacBook Air laptop can be seen as a good implementation of the sensor based ransomware detection

Product Name	**m4-1015dx**
Product Number	C2M81UA
Operating System	Windows 10
Microprocessor	2.20GHz 3rd Generation Intel Core i7-3632QM
Microprocessor Cache	4MB L2 Cache
Memory	8GB DDR3 SDRAM (2 DIMM)
Video Graphics	Intel HD graphics 4000
Hard Drive	Corsair Force-LX 256GB SATA 3 6Gb/s SSD
Power	65W AC Adapter
Battery	6-cell 62WHr 2.8 Ah Lithium-Ion

Table 4.1. HP ENVY m4-1015dx Product Specifications

	Name	**Type**		**Name**	**Type**
1	BUS_SPEED	Clock	12	CPU_TOTAL	Load
2	CPU_CORE_#1	Clock	13	MEMORY	Load
3	CPU_CORE_#2	Clock	14	CPU_CORES	Power
4	CPU_CORE_#4	Clock	15	CPU_GRAPHICS	Power
5	AVAILABLE_MEMORY	Data	16	CPU_PACKAGE	Power
6	TOTAL_LBA_WRITTEN	Data	17	CPU_CORE_#1	Temp
7	USED_MEMORY	Data	18	CPU_CORE_#2	Temp
8	CPU_CORE_#1	Load	19	CPU_CORE_#3	Temp
9	CPU_CORE_#2	Load	20	CPU_CORE_#4	Temp
10	CPU_CORE_#3	Load	21	CPU_PACKAGE	Temp
11	CPU_CORE_#4	Load	22	TEMPERATURE	Temp

Table 4.2. HP ENVY m4-1015dx Sensors

method presented in this experiment.

Product Name	**MacBook Air**
Operating System	OSX 10.12.5
Microprocessor	1.3GHz Dual-Core Intel Core i5
Microprocessor Cache	3MB Shared L3 Cache
Memory	4GB of 1600MHz LPDDR3
Video Graphics	Intel HD Graphics 5000
Hard Drive	128GB Flash Storage
Power	45W MagSafe2 Power Adapter
Battery	54WHr Lithium Polymer

Table 4.3. MacBook Air 13-Inch Mid 2013 Product Specifications

4.2. Data Acquisition and Model Creation

4.2.1. Sensor Data

Sensor data is the input required to make predictions about the binary state of a system using machine learning algorithms. It is important that the methodology utilized in procuring sensor data is both quick and reliable. Many tools exist which allow users to simply monitor sensor data in a graphical user interface, but the experiment required the automated retrieval and parsing of sensor data at specific intervals in time. Both tools utilized in this experiment can be used to either write data to a file for future analysis or provide a feature vector of real-time sensor data to a prediction model. Writing data to a file allows for the direct comparison of machine learning algorithms in as far as how they would have predicted the state of the system given the same input data.

4.2.1.1. Open Hardware Monitor

	Name	Type		Name	Type
1	SMART_DISK_APPLE_SSD	Temp	36	BATTERY_CURRENT	Current
2	AIR_INLET	Temp	37	CPU_CORE	Current
3	BATTERY	Temp	38	CPU_HIGH_SIDE	Current
4	BATTERY_CHARGER	Temp	39	CPU_SUPPLY_1	Current
5	BATTERY_POSITION_1	Temp	40	CPU/VRM_SUPPLY_2	Current
6	BATTERY_POSITION_2	Temp	41	DC_INPUT	Current
7	BATTERY_POSITION_3	Temp	42	DDR3_MEM_1.35V_LINE	Current
8	BOTTOM_SKIN	Temp	43	DDR3_MEM_S3_LINE	Current
9	CAMERA_PROXIMITY	Temp	44	DISCRETE_BATTERY	Current
10	CHARGER_PROXIMITY	Temp	45	LCD_PANEL	Current
11	CPU_A_PROXIMITY	Temp	46	PWR_SUPPLY/BAT	Current
12	CPU_CORE_1	Temp	47	SSD_SUPPLY	Current
13	CPU_CORE_2	Temp	48	WLAN_CARD	Current
14	LEFT_PALM_REST	Temp	49	5V_S0_LINE	Power
15	MAIN_HEAT_SINK_2	Temp	50	BACKLIGHT	Power
16	MAIN_LOGIC_BOARD	Temp	51	CPU_CORE	Power
17	PLATFORM_CONT_HUB	Temp	52	CPU_HIGH_SIDE	Power
18	SSD_BAY	Temp	53	CPU_PACKAGE_CORE	Power
19	SSD_TEMPERATURE_A	Temp	54	CPU_PACKAGE_GPU	Power
20	SSD_TEMPERATURE_B	Temp	55	CPU_PACKAGE_TOTAL_1	Power
21	WLAN_CARD	Temp	56	CPU_PACKAGE_TOTAL_2	Power
22	SMART_BATTERY_1	Temp	57	CPU_SUPPLY_1	Power
23	SMART_BATTERY_2	Temp	58	CPU/VRM_SUPPLY_2	Power
24	BATTERY_1_CELL_1	Voltage	59	DC_INPUT	Power
25	BATTERY_1_CELL_2	Voltage	60	DDR3_MEM_1.35V_LINE	Power
26	BATTERY_1_VOLTAGE	Voltage	61	DDR3_MEM_S3_LINE	Power
27	CPU_CORE	Voltage	62	LCD_PANEL	Power
28	CPU_SUPPLY_1	Voltage	63	PWR_SUPPLY/BAT	Power
29	DC_INPUT	Voltage	64	SSD_SUPPLY	Power
30	PWR_SUPPLY/BAT	Voltage	65	TOTAL_SYSTEM_SUPPLY	Power
31	SSD_SUPPLY	Voltage	66	WLAN_CARD	Power
32	WLAN_CARD	Voltage	67	BATTERY_1_CURRENT	Capacity
33	BATTERY_1_CURRENT	Current	68	BATTERY_1_TOTAL	Capacity
34	5V_S0_LINE	Current	69	FAN_EXHAUST_RPMS	RPMs
35	BACKLIGHT	Current	–		

Table 4.4. MacBook Air 13-Inch Mid 2013 Sensors

Open Hardware Monitor (OHM) is a free open source project that monitors temperature sensors, fan speeds, voltages, load, and clock speeds of a Windows or Linux computer. OHM is an especially powerful tool, in the context of this experiment, as it publishes all sensor data to Windows Management Instrumentation (WMI) for accessibility from the command line. Initially, OHM is capable of providing a comma separated list of the nomenclature for each sensor and the corresponding sensor category. OHM is continuously given a second command after a set time interval which returns a comma separated list of only sensor values which are in the same order as the initial list of sensor nomenclatures.

4.2.1.2. Hardware Monitor for Mac

Hardware Monitor is an application to read out hardware sensors in Macintosh computers. Hardware Monitor allows sensor data to be accessed from the command line which in turn allows automated scripts to work with real time sensor data [8]. Prior to reading any sensor values a command can be sent to the command line which returns a comma separated list of sensor names and categories which can then act as a header for future sensor data. In order to access the sensor data a command is fed to the command line which returns a string of comma separated sensor values in the same order as the header string.

4.2.2. CPU Load Simulation

Simulating a CPU load is necessary to test the ability of the prediction models to make accurate predictions even when a user is placing additional stress on a system.

4.2.2.1. CPUSTRES

CPUSTRES is a program for the Windows operating system which allows threads to be run creating a load on the CPU. Each thread can be set to one of four activity levels which are low, medium, busy, and maximum. Using the Windows task manager in conjunction with CPUSTRES a combination of threads at various activity levels can be found which creates a desired load between 1% and 100%.

4.2.2.2. Mac Yes Program

The Mac yes program is a terminal command which prints the word "yes" at such a speed that it consumes all the available processor resource available. Each instance of the yes program will consume the resource of a single CPU core or a single hyperthread if the system is hyperthreaded. Running a certain number of yes program instances can result in a desired load on a CPU.

4.2.3. Python Test Packages

The Python programming language is a commonly used high level language which is capable of supporting large projects in a greatly reduced time frame. Python was selected for use in this experiment as several high level packages exist which allow for quick and efficient implementation of the desired operation in both acquiring and analyzing the training and test data. This experiment attempts a "proof of concept" implementation of the sensor based ransomware detection methodology. Python emphasizes high level readability which is especially useful in allowing readers to quickly look at the implementation and grasp the direction of the experiment.

4.2.3.1. Crypto

The Crypto python package is a high level encryption library which includes implementations of all five encryption modes discussed in chapter 2. Prior to encryption a cipher data structure is created and the encryption mode, encryption key, initialization vector, and block size are all defined. Plain text is provided to the cipher data structure in sizes equal to the predefined block size. The cipher data structure then returns the appropriate cipher text. Furthermore, the same cipher data structure can be used to decrypt cipher text. Cipher text is provided in sizes equal to the predefined block size and the appropriate plain text is returned. Entire files are encrypted and decrypted by reading them one block size at a time until the end of the file.

4.2.3.2. Windows Management Instrumentation (WMI)

Windows Management Instrumentation (WMI) consists of a set of extensions to the Windows Driver Model, providing the operating system an interface through which instrumented components can provide information and notification. WMI allows scripting languages to manage Microsoft Windows personal computers and servers. In the context of this experiment, WMI is used to provide real time sensor data to a program used to make predictions about the binary state of a system. The WMI python package is a lightweight wrapper on top of the extensions used to allow the python programming language to communicate with the WMI API [20]. The WMI python package allows quick and simple WMI interface which is used to read real-time sensor data provided by OHM.

4.2.3.3. NumPy

NumPy is a python package which adds support for a powerful N-dimensional array object, tools for integrating C/C++ and Fortran code, and linear algebra capabilities. It is also used as an efficient multi-dimensional container of generic data which allows arbitrarily defined data types. This allows NumPy to seamlessly and quickly integrate with a wide variety of databases [30]. The implementation of this experiment requires data to be stored in both matrix and vector form and various linear algebra operations to be performed. NumPy is a high level python wrapper around low level efficient linear algebra packages such as BLAS and LAPACK. Internally, NumPy is implemented very similarly to MATLAB which is an industry leading linear algebra package. However, NumPy is open-source and a viable solution for this and future implementations.

4.2.3.4. Pandas

Pandas python package provides fast, flexible, and expressive data structures which is aimed at being the fundamental high level building block for performing data analysis in python. Pandas is well suited for tabular data similar to Excel spreadsheets, ordered and unordered time series data, and arbitrary matrix data with row and column labels [40]. Pan-

das is built on top of NumPy, allowing an even higher level implementation of its optimized low level functionality. Pandas most useful functionality, in the context of this experiment, is its ability to deal with time series data. The implementation of the tool will require a vector of sensor data to be read at a fixed polling interval which creates a time series. It has been stated that both the OHM tool and the Hardware Monitor for Mac initially outputs a comma separated list of sensor nomenclatures. When this line is written to a file first it is read by Pandas as the column labels in a time series matrix. Afterwards, each line of sensor data is read as a discrete time interval with the rows being labeled sequentially to represent their relative position in time. Once the time series data is constructed, Pandas allows the data to be easily manipulated in ways such as extracting individual columns or a range of columns, extracting individual rows or a range of rows, deleting individual columns or a range of columns, and deleting individual rows or a range of rows. Pandas also allows separate time series with the same column labels to be combined together effectively combining separate test runs into a single testable time series.

4.2.3.5. Sci-Kit Learn

Sci-Kit Learn (Sklearn) is an open source machine learning library for Python. It features various classification, regression, and clustering algorithms which can be implemented at a very high level. Sklearn is designed to work with both the NumPy and Pandas python packages [12]. Pandas is used to read in the time series data and separate the independent data into a matrix and the dependent data into a vector. Generally, the sklearn models are trained with an independent training matrix and a dependent training vector. After the model is trained an independent matrix is used as an input to generate a dependent vector of predictions. Additionally, Sklearn also includes high level implementations for the various preprocessing techniques covered in chapter 3.

4.2.3.6. Matplotlib

Matplotlib is an open source Python plotting library which produces publication quality figures in a variety of hardcopy formats [22]. Matplotlib works directly with NumPy arrays which are the product of Sklearn model predictions. This easily allows the test vector and the prediction vector to be plotted in a time series for easy visualization.

4.2.3.7. Psutil

Psutil is a cross-platform Python library which is used for the retrieval of information about running processes and system utilization. The library is utilized in this experiment to log the current overall CPU load percentage during simulated load test data collection. The CPU load is provided as a floating point number between 0.0 and 100.0 and represents the collective percentage of CPU utilization for all the CPU cores in a system [25].

4.2.4. Ransomware Simulation Routine

The ransomware simulation routine follows the general ransomware attack phase methodology. The encryption mode is defined prior to running with the encryption key and initialization vector being hard coded rather than randomly generated for safety. The routine only encrypts very specific directories and file extensions which have been documented as targets from previous ransomware attacks in an effort to target important data and complete the encryption phase as fast as possible. When a file is found in the targeted directories and it is determined it should be encrypted it is first opened and quickly scanned to determine if it has already been encrypted by reading the first line of the file which the ransomware simulation routine marks if it has already performed encryption. If it has not already been encrypted a second temporary file is created in the same directory with a header containing the encryption key, initialization vector, and encryption mode for further safety. The original file is read 16 bytes at a time and encrypted. The encrypted data is written into the temporary file. Once the file is completely read and the corresponding encrypted data has been written to the temporary file both files are closed. The original file is cleared and the temporary file data is copied over to it. After the temporary file data has been copied it is removed

from the computer. Once all the files that were found in the targeted directories have been encrypted the ransomware simulation completes. It should be noted that encrypted files can easily be decrypted using the header of the file which contains the encryption mode used, the encryption key, and the initialization vector. Additionally, all encryption methods use the same hard coded encryption key and initialization vector on every file and the ransomware simulation routine checks every file prior to encryption to ensure it is not encrypted twice causing data loss.

4.2.5. Data Acquisition Routines

In this experiment data falls into one of two categories, training or testing. Training data must be collected in a way which captures the behavior of the system in both the states of "normal operation" and "under attack". The training data should have a large enough sample size of both states to generate an accurate model. However, test data should have sample sizes of the states which more accurately reflects how they will be normally seen. Furthermore, this experiment implements five different encryption modes which each require training and testing data to be captured individually.

4.2.5.1. Initial Training Data

The initial training data set is collected with no additional user activity being performed. Training data is collected in roughly two hour blocks. For each two hour block the training routine starts by measuring how long the ransomware simulation takes to fully encrypt the targeted files. After fully encrypting the system it is decrypted and returned to the initial state. The routine then waits two minutes for the sensors to return to a normal state. Once the two minute wait period has ended the routine starts measuring time towards the total two hours of training. First, the system sits unattacked for the duration of the time it takes to encrypt the system. Afterwards, the system is encrypted with the ransomware simulation routine. Finally, the system sits unattacked for the duration of the time it takes to encrypt the system. The routine stops measuring time towards the total two hours of training and

decrypts the system returning it to its original state. The routine then waits two minutes for the sensors to return to a normal state. If the total measured time is more than two hours the routine stops. If the total measured time is less than two hours the three step wait-attack-wait routine is run again. The wait-attack-wait method employed in the training data collection keeps the ratio of "normal operation" to "under attack" at two to one. Generally, if training data was collected in the way one would expect to see attacks there would not be enough "under attack" samples to create an accurate prediction model. Furthermore, the Pandas python package allows time series to be combined together to form a single time series which allows for the creation of a larger training time series out of the smaller ones. For each of the five different encryption modes 12 two hour training sessions are carried out allowing up to 24 hours of training data per encryption mode. Training data is collected in this manner for future measurement of the effect of training time on model accuracy.

4.2.5.2. Initial Test Data

The initial test data is collected with no additional user activity being performed. Test data is collected in roughly one hour blocks. For each one hour block one of the five encryption modes is used during a simulated ransomware attack. One attack occurs every hour resulting in a much larger sample of "normal operation" time intervals than "under attack" time intervals. After the one hour period the system is decrypted and returned to its original state. For accuracy analysis, robustness analysis, and training time analysis 24 hours of test data per encryption mode are recorded.

4.2.5.3. Simulated Load Training Data

Once the initial data has been processed and analyzed a determination will be made for a reduced training duration. It is believed that training time analysis with the initial data set of 24 hours per encryption method will reveal the point in time when there is no longer a significant advantage to additional training. Using this reduced training time requirement, new training data will be collected for the final test, simulated load analysis. Training data

will be collected for all encryption methods with simulated CPU loads of 0%, 25%, 50%, 75%, and 100%.

4.2.5.4. Simulated Load Test Data

Test data will be collected for each encryption method with simulated CPU loads of 0%, 25%, and 50%. Simulated load test data will also include the current overall CPU load percentage at the time of each collection. Each encryption method will have test data collected at each simulated CPU load for a period of 6 hours. During a one hour period the ransomware encryption phase will occur once at a randomly determined time. After each one hour period the data collection will briefly stop while the encrypted data is decrypted and time is allowed for the sensors to return to a normal state.

4.3. Algorithm Analysis

The collected training and testing data is used to assess the performance of the machine learning algorithms previously outlined in chapter 3. Five tests have been created in order to select the highest performing algorithm from the field of twelve, optimize the prediction model to increase performance, and test the final prediction model with different simulated CPU loads as a proof of concept.

Previously in section 3.3.2 it was established that no one metric fully encapsulates the performance of a binary classifier. However, the Matthews correlation coefficient (MCC) was shown to be the closest metric to a single value which expresses the performance of a model, especially when there is a class imbalance like the one present in this case. Therefore, for all testing the MCC value will be used to determine which machine learning algorithms perform the best. Additionally, the sensitivity, precision, specificity, fallout, accuracy, F_1 score, rate of attack recognition, and mean time to attack recognition will all be reported along with the MCC for further analysis and insight.

4.3.1. Stratified 10-Fold Cross Validation

Stratified 10-fold cross validation is performed on each algorithm individually using only CBC training data in order to determine how well various test configurations are likely to perform when presented with new and unseen data. The test parameters include classification method, data scaling method, dimensionality reduction method, and moving average method. The classification method includes two options, binary and multiclass. Binary classification models are trained with a dependent vector of binary values indicating true for "under attack" and false for "normal operation". Multiclass classification is trained with a dependent vector of values with 7 being the highest and representing the highest likelihood of being "under attack" and 0 being the lowest and representing the highest likelihood of being in "normal operation". Whenever the predicted value is greater than 3 a prediction of "under attack" is made. Conversely, whenever the predicted value is less than or equal to 3 a prediction of "normal operation" is made. The data scaling methods include four options, feature standardization, data normalization, feature min-max scaling, and no scaling. Sklearn includes data structures which fit and transform the training data and are also capable of transforming future test data as needed. The dimensionality reduction method includes seven options which are variations of PCA, feature selection, and no dimensionality reduction. Principal component analysis is performed such that the reduced dimension data maintains at least 70%, 80%, or 90% of the original variance. Feature selection is performed such that only the top 50%, 70%, or 90% of features are selected based on feature importance metrics. Finally, the moving average method consists of five options including simple moving average with window sizes of 2 and 4, weighted moving average with window sizes of 2 and 4, and no moving average.

There exists 280 different combinations of the test parameters which must all be tested individually for each algorithm. For each combination the training data is separated into 10 equal sized subsets. The Matthews correlation coefficient (MCC) is used rather than simple accuracy as it provides a better measure of the overall model performance where there is a class imbalance. Ten different MCC values are found by ten different tests in which each subset acts as the sole training data once and is part of the test data 9 times. The ten MCC

values are averaged together to get the average MCC of the algorithm with the specific test parameter combination. The 280 average MCC values are ordered from greatest to least with the highest value belonging to the test parameter combination which is most likely to result in the highest performance during testing. The highest performing test parameter combination is used for the appropriate algorithm for the duration of analysis.

4.3.2. Accuracy Analysis

The accuracy analysis is performed for each of the twelve machine learning algorithms. There exists five different encryption modes with each having separate training and testing data. Every combination of encryption modes of every size from 1 to 5 has a prediction model trained, and the same combination of testing data is used to assess how accurately the model performs when making predictions for data it has been trained with. Accuracy, in the context of this analysis, is used as a general term which refers to the ability of an algorithm to perform well given test data it has been trained to predict. The actual accuracy metric is only one of several metrics which is used to determine the highest performing algorithms.

The accuracy analysis is split into five parts with different combination selection sizes for the models. The test procedure can be outlined as follows:

1. Select combinations of 1 encryption mode from 5 total encryption modes, $\binom{5}{1}$, resulting in 5 models. Test each model with corresponding encryption mode test data.

2. Select combinations of 2 encryption modes from 5 total encryption modes, $\binom{5}{2}$, resulting in 10 models. Test each model with corresponding encryption mode test data.

3. Select combinations of 3 encryption modes from 5 total encryption modes, $\binom{5}{3}$, resulting in 10 models. Test each model with corresponding encryption mode test data.

4. Select combinations of 4 encryption modes from 5 total encryption modes, $\binom{5}{4}$, resulting in 5 models. Test each model with corresponding encryption mode test data.

5. Select combinations of 5 encryption modes from 5 total encryption modes, $\binom{5}{5}$, resulting in 1 model. Test the model with all encryption mode test data.

There are 31 total models and tests for each algorithm during the accuracy analysis. For each algorithm the scores are computed from the averages of the 31 different tests carried out on the 31 different models.

4.3.3. Robustness Analysis

The robustness analysis is performed for each of the twelve machine learning algorithms. The same 31 models generated during the accuracy analysis are then each tested individually with each encryption mode test data. Robustness, in the context of this analysis, is used to refer to the ability of an algorithm to perform well given test data it has been trained to predict as well as test data it has not been directly trained to predict. This analysis will convey whether models that have not been explicitly trained to detect certain encryption modes can still detect them with relative success.

The robustness analysis is split into five parts with different combination selections sizes for the models. The test procedure can be outlined as follows:

1. Select combinations of 1 encryption mode from 5 total encryption modes, $\binom{5}{1}$, resulting in 5 models. Test each model with all five encryption mode test data sets individually.

2. Select combinations of 2 encryption modes from 5 total encryption modes, $\binom{5}{2}$, resulting in 10 models. Test each model with all five encryption mode test data sets individually.

3. Select combinations of 3 encryption modes from 5 total encryption modes, $\binom{5}{3}$, resulting in 10 models. Test each model with all five encryption mode test data sets individually.

4. Select combinations of 4 encryption modes from 5 total encryption modes, $\binom{5}{4}$, resulting in 5 models. Test each model with all five encryption mode test data sets individually.

5. Select combinations of 5 encryption modes from 5 total encryption modes, $\binom{5}{5}$, resulting in 1 model. Test each model with all five encryption mode test data sets individually.

There are 31 total models with each model requiring 5 tests for each encryption mode. Therefore, 155 tests are required for each algorithm during the robustness analysis. The

scores for each algorithm are found based on the averages of the 155 tests carried out on the 31 models. The algorithm with the highest MCC average in both the accuracy analysis and robustness analysis is selected and used in the remainder of the tests.

4.3.4. Training Time Analysis

It was established that training data is collected for each encryption mode two hours at a time up to twenty four hours. Therefore, models can be trained with data ranging from two hours to 24 hours in intervals of two hours. Training time analysis will show how well the selected highest performing algorithm performs when trained for various amounts of time. Testing will require a single model trained with all 5 encryption modes at each training time varying from 2 hours to 24 hours in 2 hour intervals. The 12 MCC scores should offer insight into the effectiveness of the model as it is trained with different training times. The ideal training time can be determined based on which time has the highest MCC score. Furthermore, if the MCC scores are very close it can be concluded that training for only 2 or less hours is a viable option for future models.

4.3.5. Simulated Load Analysis

All other testing conducted on the models is during a time when there is no additional user activity and minimal background processes being performed. Therefore, it is reasonable to assume periods of ransomware activity would be very easily distinguished from periods with no ransomware activity resulting in very high MCC scores. For this reason it is necessary to test with simulated CPU loads in order to determine if the machine learning algorithm determined to provide the best results is capable of accurately predicting attack instances with additional user activity. The simulated load training data is collected with 0%, 25%, 50%, 75%, and 100% simulated CPU loads. The simulated load test data is collected with 0%, 25%, and 50% simulated CPU loads and will additionally log the current overall CPU load at the time of each data collection. The average overall CPU load caused by the encryption process will be determined by analyzing the test data with 0% simulated CPU load during

test intervals labeled "under attack". Separate predictive models will be generated with the simulated load training data for 0%, 25%, 50%, 75%, and 100% simulated CPU loads. Testing will be conducted with three different CPU loads, 0%, 25%, and 50%. For all CPU loads each of the five encryption modes are run for six hours apiece. Each hour there is a single simulated ransomware attack. After the one hour test has concluded the encrypted data is unencrypted which restores the original state of the system.

During the simulated load analysis an ensemble predictive model will be used which is comprised of the five models trained at 0%, 25%, 50%, 75%, and 100% simulated CPU loads. Utilizing the logged overall system CPU load in the simulated load test data, a determination will be made for which models to utilize in the prediction. The two models utilized during prediction will account for the two possible scenarios of the current system state. In scenario one the system can be "operating normally" leading to the model with the closest trained simulated CPU load to the current overall system CPU load to be selected as the model most likely to result in an accurate "normal operation" prediction. In scenario two the system can be "under attack" with the encryption process increasing the overall system CPU load such that one of the models trained with a lower simulated load level than the current load level will be the most likely model to accurately make an "under attack" prediction. The previously determined average CPU load added by the encryption process is subtracted from the current overall system CPU load and the model with the closest trained simulated CPU load to the adjusted value will be selected. Both models will generate a prediction confidence score for both the "normal operation" and "under attack" states. The scenario one model's confidence score for "normal operation" will be compared to the scenario 2 model's confidence score for "under attack" and the highest confidence score will result in the current system state prediction. This method essentially weighs whether the sensor signature more closely resembles scenario 1 or scenario 2 as the reason for the current overall system CPU load is unknown. Results for testing with 0%, 25%, and 50% simulated loads will be provided for each of the individual encryption methods as well as an instance where all encryption methods are combined.

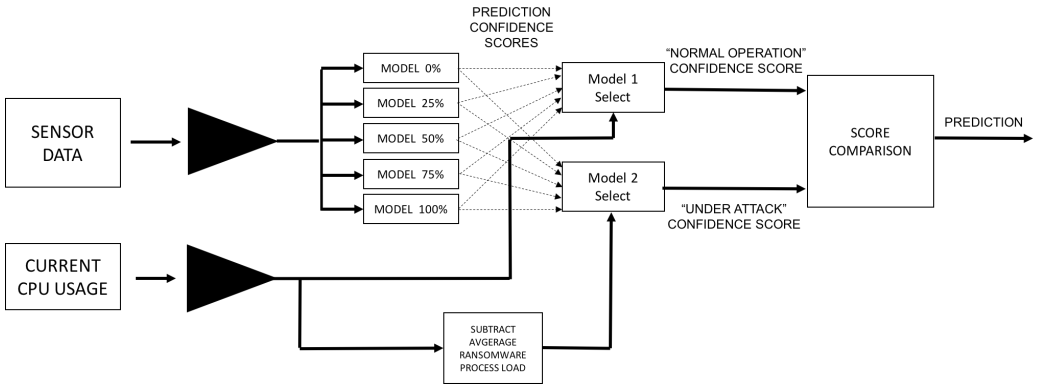

Figure 4.1. Simulated Load Analysis Ensemble Predictive Model

Figure 4.1 illustrates how the ensemble predictive model will be used during the simulated load analysis. One important aspect to note is that the test data is collected first and then analyzed according to the behavior which would have been seen if the predictions were made in real time. Therefore, in the simulated load analysis it is most convenient for all models to generate confidence scores for every prediction instance and then only the selected model scores are used in the actual prediction. However, if the ensemble prediction model were to be implemented in real-time the current system CPU load would be used to select the models prior to confidence score calculation to reduce overall computational overhead by only performing such computations with the two selected models.

Chapter 5

EXPERIMENTAL RESULTS

5.1. HP ENVY m4-1015dx

5.1.1. Stratified 10-Fold Cross Validation

Table 5.1 shows that the logistic regression algorithm and multilayer perceptron algorithm are the most likely algorithms to provide quality predictions with 0.9610 and 0.9600 MCC scores respectively. The logistic regression model trained as a binary classifier with standardized scaling of data, no dimensionality reduction, and a weighted moving average window size of 4 is the best performing configuration tested. The top performing configurations for the one-vs-one, one-vs-rest, and SVC algorithms all resulted in an identical MCC score of 0.9591. This is due to the fact that one-vs-one and one-vs-rest implement SVC predictive models as an ensemble. When the models are trained as binary classifiers they essentially all implement the same method for prediction. Since no configuration of the one-vs-one and one-vs-rest algorithms trained as multiclass classifiers performed better than the SVC model trained as a binary classifier there optimal performance, as determined by cross validation, is equivalent. The three tree based algorithms performed almost equally as well with the more complex random forest and extra tree algorithms performing only slightly better than the simple decision tree algorithm. Linear regression was the only algorithm which cross validation testing determined performed optimally when trained as a multiclass classifier. Additionally, linear regression performed optimally with no data scaling and no dimensionality reduction making its implementation the simplest out of all of the tested algorithms. The k-means clustering method resulted in a significantly lower MCC score than all of the other algorithms at 0.7074. This was expected as the clustering algorithm is the

Algorithm	**Highest MCC**	**Configuration**
Log Reg	0.9610	Binary Classification, Standardized Scaling, None, Weighted Moving Average Window Size 4
MLP	0.9600	Binary Classification, Min-Max Scaling, None, Weighted Moving Average Window Size 4
One-V-One	0.9591	Binary Classification, Min-Max Scaling, None, Weighted Moving Average Window Size 4
One-V-Rest	0.9591	Binary Classification, Min-Max Scaling, None, Weighted Moving Average Window Size 4
SVC	0.9591	Binary Classification, Min-Max Scaling, None, Weighted Moving Average Window Size 4
Rand Forest	0.9570	Binary Classification, Min-Max Scaling, Feature Selection Upper 70%, Weighted Moving Average Window Size 4
Extra Tree	0.9549	Binary Classification, Normalization Scaling, Feature Selection Upper 70%, Weighted Moving Average Window Size 4
Decision Tree	0.9528	Binary Classification, None, Feature Selection Upper 90%, Weighted Moving Average Window Size 4
KNN	0.9311	Binary Classification, Standardized Scaling, None, Weighted Moving Average Window Size 4
Lin Reg	0.9285	Multiclass Classification, None, None, Weighted Moving Average Window Size 2
N Bayes	0.9069	Binary Classification, None, Feature Selection Upper 50%, Simple Moving Average Window Size 2
K-Means	0.7074	Binary Classification, Min-Max Scaling, Feature Selection Upper 70%, Simple Moving Average Window Size 2

Table 5.1. HP ENVY m4-1015dx Stratified 10-Fold Cross Validation Results

only unsupervised algorithm tested.

5.1.2. Accuracy Analysis

Algorithm	**MCC**	**Sen**	**Spec**	**Prec**	**Fall**	**Acc**	**F1**	**RAR**	**MTAR**
Log Reg	0.4928	0.2825	0.9981	0.9220	0.0019	0.9446	0.4319	1.0	61.99
Extra Tree	0.4882	0.2811	0.9977	0.9129	0.0022	0.9442	0.4288	1.0	53.03
One-V-One	0.4882	0.2803	0.9979	0.9129	0.0021	0.9440	0.4286	1.0	69.81
One-V-Rest	0.4882	0.2803	0.9979	0.9129	0.0021	0.9440	0.4286	1.0	69.81
SVC	0.4882	0.2803	0.9979	0.9129	0.0021	0.9440	0.4286	1.0	69.81
MLP	0.4717	0.2657	0.9977	0.9021	0.0023	0.9427	0.4101	1.0	69.47
Rand Forest	0.4707	0.2600	0.9982	0.9194	0.0018	0.9431	0.4038	1.0	59.60
Lin Reg	0.4414	0.2232	0.9988	0.9405	0.0012	0.9405	0.3598	1.0	70.97
KNN	0.4267	0.2689	0.9837	0.7961	0.0163	0.9303	0.3870	1.0	66.41
N Bayes	0.4137	0.2639	0.9598	0.8104	0.0402	0.9084	0.3771	1.0	65.08
Decision Tree	0.3922	0.2547	0.9884	0.7014	0.0116	0.9337	0.3679	1.0	48.10
K-Means	0.1355	0.3818	0.6773	0.5903	0.3226	0.6560	0.2716	0.9962	52.01

Table 5.2. HP ENVY m4-1015dx Accuracy Analysis Results

In general the accuracy analysis resulted in almost the same ranking as cross validation. Logistic regression resulted in the highest MCC score at 0.4928. However, there was some notable differences in results as indicated by the MCC scores and rankings of the extra tree and MLP models. The MLP model resulted in almost the same MCC score as the logistic regression model during cross validation, but resulted in only the sixth highest MCC score during accuracy analysis. Table 5.2 shows exactly how deceptive the accuracy and precision values can be while determining the performance of the different models. Logistic regression resulted in an accuracy of 0.9446 and a precision of 0.9220. However, the sensitivity of the logistic regression model was only 0.2825. These results show that periods of the system being "under attack" occur at a significantly lower frequency than than periods of "normal operation". The model's high precision and accuracy scores, but low sensitivity

score, indicate it predicted "normal operation" the majority of predictions resulting in a high number of correct predictions. However, the model predicted the "under attack" instances incorrectly 71.75% of the time which is the most important aspect of this experiment.

Upon further investigation, figure 5.1 shows a very similar prediction time series to that of the actual attack time series using the logistic regression model. The prediction time series appears to show a small number of erroneous predictions between actual attacks and a much larger number made during actual attacks. However, table 5.2 indicates that periods of "under attack" were only correctly predicted 28.25% of the time. Figure 5.2 shows a single attack instance and gives a much clearer example of how predictions are actually made. The red line represents the entire "under attack" instance. The blue line shows many correct "under attack" predictions were made during this time. It is most likely that the incorrect predictions are occuring as a result of the test system containing only a small number of sensors which are almost entirely comprised of CPU sensors. During instances labeled "under attack" the encryption process is walking through a user's directories and encrypting only files which are of a specific set of extensions. Only when such a file is found does actual encryption occur which in turn greatly increases the load placed on the CPU. This is most clearly indicated by the extended period of correctly made predictions at the end of the "under attack" instance. It is most likely that the encryption process encountered a directory with a high volume of target files resulting in an extended period of heavy CPU utilization. The results in table 5.2 actually indicate that the logistic regression model made at least one correct "under attack" prediction during all attack instances resulting in all attacks being caught. Additionally, the average time before each attack was caught was 61.99 seconds most likely due to target files not being encountered by the encryption processes until that point. These results indicate that measuring the performance of the predictive model based on the rate at which "under attack" instances are correctly predicted may not provide the actual account of how well they perform. The results show that all algorithms, except for k-means clustering, successfully caught all attack instances with average times between 48.10 and 70.97 seconds. The extra tree model could even be viewed as the superior model when

weighed in this manner as it had a comparable MCC score to the logistic regression model and an average time of attack recognition that was 8.96 seconds faster.

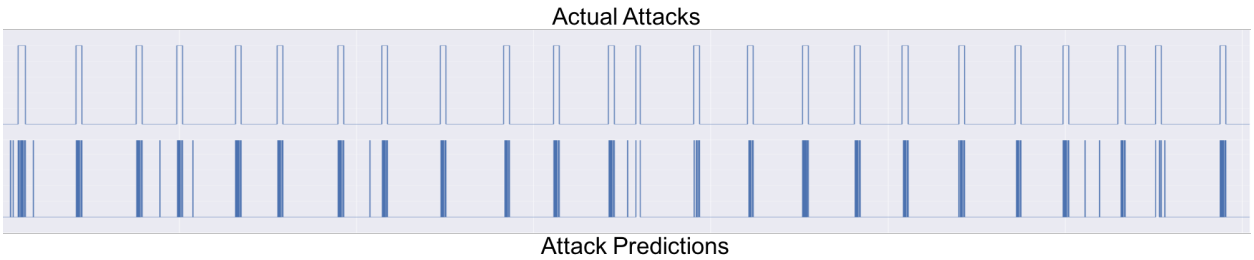

Figure 5.1. HP ENVY m4-1015dx Logistic Regression CBC Trained and Tested Time Series

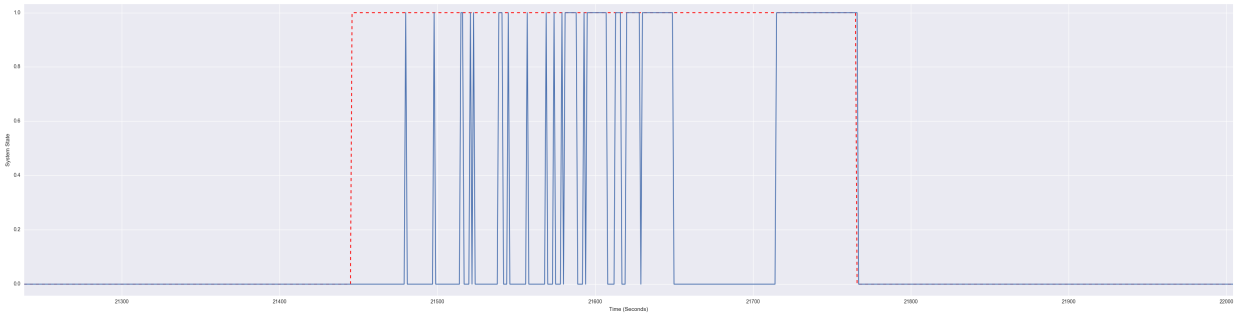

Figure 5.2. HP ENVY m4-1015dx Logistic Regression Single Attack Instance Time Series

5.1.3. Robustness Analysis

The robustness analysis shows results which are again very similar to the cross validation and accuracy analyses. Logistic regression performed better than all the other machine learning algorithms with an MCC score of 0.4952. One interesting result to note is that several models, including logistic regression, had a higher MCC score for the robustness analysis than the accuracy analysis. Additionally, those models which did not have higher scores were in general not significantly lower. This would suggest that the prediction models work for encryption methods they have not been trained for almost as well, if not equally as well, as encryption methods they have been trained for. Furthermore, the average time

Algorithm	MCC	Sen	Spec	Prec	Fall	Acc	F1	RAR	MTAR
Log Reg	0.4952	0.2864	0.9981	0.9246	0.0019	0.9443	0.4344	1.0	63.76
One-V-One	0.4885	0.2818	0.9979	0.9136	0.0021	0.9438	0.4290	1.0	70.3618
One-V-Rest	0.4885	0.2818	0.9979	0.9136	0.0021	0.9438	0.4290	1.0	70.3618
SVC	0.4885	0.2818	0.9979	0.9136	0.0021	0.9438	0.4290	1.0	70.3618
Extra Tree	0.4864	0.2834	0.9976	0.9054	0.0024	0.9436	0.4285	1.0	47.51
MLP	0.4747	0.2705	0.9976	0.9048	0.0024	0.9426	0.4136	1.0	70.83
Rand Forest	0.4707	0.2614	0.9981	0.9214	0.0019	0.9425	0.4036	1.0	63.97
KNN	0.4526	0.2736	0.9857	0.8575	0.0143	0.9324	0.4035	1.0	67.29
Lin Reg	0.4433	0.2263	0.9988	0.9463	0.0012	0.9403	0.3617	1.0	70.98
N Bayes	0.4125	0.2664	0.9583	0.8102	0.0417	0.9060	0.3733	1.0	68.17
Decision Tree	0.4013	0.2609	0.9884	0.7269	0.0116	0.9334	0.3709	1.0	52.63
K-Means	0.1363	0.3807	0.6780	0.6102	0.3220	0.6556	0.2700	0.9944	53.57

Table 5.3. HP ENVY m4-1015dx Robustness Analysis Results

to attack recognition was only slower in the logistic regression model by 1.77 seconds. Once again the argument could be made that the extra tree model had the superior performance to the logistic regression model as they had comparable MCC scores and the extra tree model had an average attack recognition 16.25 seconds faster than the logistic regression model. In fact, the extra tree model actually had an average attack recognition time 5.52 seconds faster than it did during the accuracy analysis. This indicates the extra tree model performs very well with encryption methods it has not been directly trained to detect. However, as the determination for the most effective machine learning algorithm which will be used in the training time and simulated load analyses was decided to be the MCC scores of the accuracy and robustness analyses the logistic regression model must be selected as it had the highest MCC score in both tests. Figure 5.3 illustrates the logistic regression model's performance when trained with one encryption method and tested with another.

5.1.4. Training Time Analysis

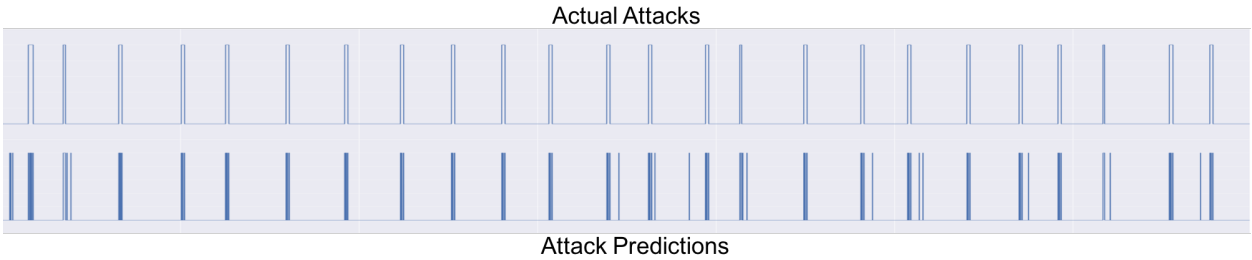

Figure 5.3. HP Laptop Logistic Regression CFB Trained XOR Tested Time Series

Training Time (Hours)	**MCC**	**Sen**	**Spec**	**Prec**	**Fall**	**Acc**	**F1**	**RAR**	**MTAR**
2	0.5364	0.3250	0.9984	0.9419	0.0016	0.9478	0.4833	1.0	39.44
4	0.5785	0.3734	0.9984	0.9485	0.0016	0.9514	0.5359	1.0	14.23
6	0.5398	0.3291	0.9983	0.9417	0.0017	0.9480	0.4877	1.0	35.48
8	0.4911	0.2789	0.9982	0.9258	0.0018	0.9441	0.4287	1.0	69.52
10	0.5247	0.3131	0.9983	0.9373	0.0017	0.9468	0.4694	1.0	45.67
12	0.5122	0.2997	0.9983	0.9344	0.0017	0.9458	0.4538	1.0	57.73
14	0.4906	0.2782	0.9982	0.9261	0.0018	0.9441	0.4279	1.0	69.65
16	0.4993	0.2862	0.9983	0.9314	0.0017	0.9448	0.4378	1.0	66.75
18	0.4851	0.2764	0.9979	0.9135	0.0021	0.9436	0.4244	1.0	70.28
20	0.4854	0.2761	0.9979	0.9157	0.0021	0.9437	0.4242	1.0	70.35
22	0.4849	0.2760	0.9979	0.9141	0.0021	0.9436	0.4240	1.0	70.34
24	0.4898	0.2799	0.9980	0.9185	0.0020	0.9440	0.4291	1.0	70.07

Table 5.4. HP ENVY m4-1015dx Training Time Analysis Results

The models tested in the accuracy and robustness analyses were trained with twenty four hours of data for each encryption method. Table 5.4 shows the performance of twelve models trained with increasing training times from two hours to twenty four hours in intervals of two hours. Interestingly, the models tend to decrease in performance as the training times increase. In order to determine if this behavior extended to even smaller training times models were made and analyzed for training times from one minute to two hours in increments of one minute. The results can be seen in figure 5.4 were the MCC scores were generally comparable after one hour of training. Figure 5.5 shows the MCC scores for the models trained from two hours to twenty four hours with the red line indicating the average of the MCC scores from one hour to two hours of training. Interestingly, the MCC scores tend to level out on the red line as they increase. Further testing showed the particular two hour set of training data added for the testing at four hours performed much better than any other single two hour training set which explains the increased performance at four hours. This leads to the conclusion that four hours of training is not necessarily the optimal training time, but only appears as such due to the high performance of the single set of two hour training data. When utilizing the high performing two hour training data set first, the MCC is actually brought down from the additional training data. Considering that the new method presented in this experiment would require a user to allow time for training prior to use the conclusion reached is that one to two hours of training per encryption method is sufficient to create a predictive model comparable to the one used to achieve the results reported in the accuracy and robustness analyses. For this reason the training time for each encryption method during the simulated load training data collection will be two hours at each simulated load level.

5.1.5. Simulated Load Analysis

Table 5.5 shows the results for the implementation of the ensemble prediction model with no simulated load applied to the system. The MCC scores are slightly lower than the MCC scores recorded during the accuracy and robustness analyses. The ensemble method uses the

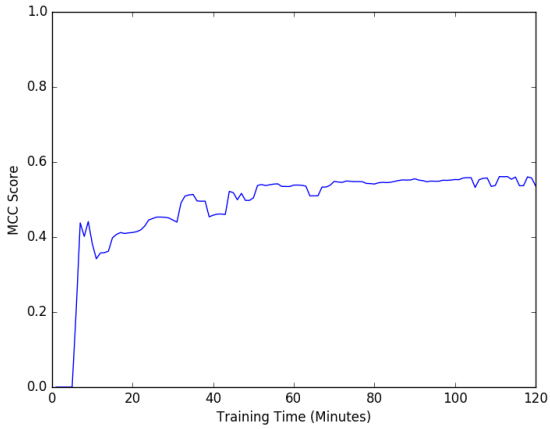

Figure 5.4. HP ENVY m4-1015dx 2 Hour Training Time Analysis

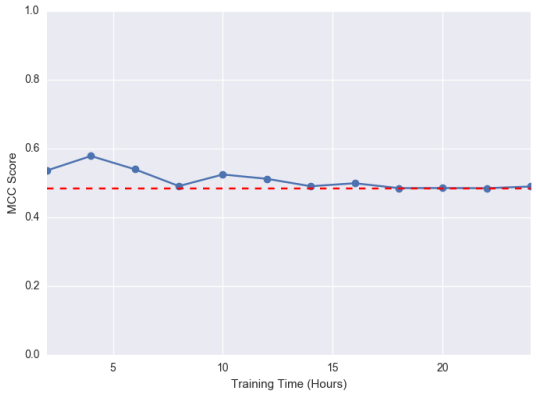

Figure 5.5. HP ENVY m4-1015dx 24 Hour Training Time Analysis

Encryption	MCC	Sen	Spec	Prec	Fall	Acc	F1	RAR	MTAR
ECB	0.4724	0.2861	0.9961	0.8393	0.0039	0.9490	0.4267	1.0	31.334
CBC	0.3057	0.2696	0.9749	0.4328	0.0251	0.9280	0.3323	1.0	5.3334
CFB	0.3313	0.2740	0.9693	0.5409	0.0307	0.8883	0.3638	1.0	120.84
OFB	0.3921	0.1932	0.9976	0.8657	0.0024	0.9385	0.3159	1.0	50.167
XOR	0.5695	0.3540	0.9990	0.9620	0.0010	0.9569	0.5176	1.0	52.0
ALL	0.5076	0.2895	0.9987	0.9505	0.0013	0.9436	0.4439	1.0	57.467

Table 5.5. HP ENVY m4-1015dx Simulated Load Analysis Results 0% Load

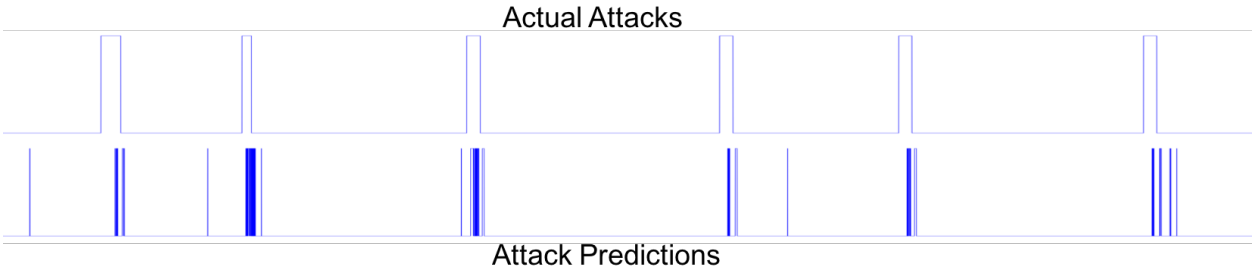

Figure 5.6. HP ENVY m4-1015dx Simulated Load Analysis XOR Time Series Plot (0% Load)

Encryption	MCC	Sen	Spec	Prec	Fall	Acc	F1	RAR	MTAR
ECB	0.2449	0.0649	1.0	1.0	0.0	0.9242	0.1220	0.6667	92.0
CBC	0.3168	0.1084	0.9999	0.9921	0.0001	0.9347	0.1955	0.6667	111.5
CFB	0.4219	0.2101	0.9985	0.9521	0.0015	0.9026	0.3442	1.0	142.83
OFB	0.3707	0.1802	0.9971	0.8413	0.0030	0.9322	0.2968	1.0	91.0
XOR	0.2344	0.3016	0.9424	0.2763	0.0575	0.8989	0.2885	1.0	36.334
ALL	0.4755	0.2596	0.9985	0.9403	0.0015	0.9358	0.4069	1.0	55.167

Table 5.6. HP ENVY m4-1015dx Simulated Load Analysis Results 25% Load

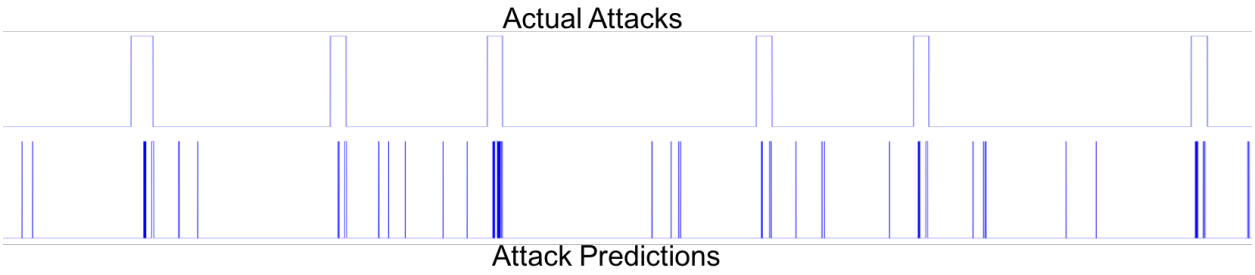

Figure 5.7. HP ENVY m4-1015dx Simulated Load Analysis OFB Time Series Plot (25% Load)

Encryption	MCC	Sen	Spec	Prec	Fall	Acc	F1	RAR	MTAR
ECB	0.3293	0.1476	0.9971	0.8203	0.0029	0.9274	0.2502	1.0	58.834
CBC	0.4789	0.2870	0.9963	0.8865	0.0037	0.9411	0.4313	1.0	63.667
CFB	0.5928	0.4170	0.9960	0.9528	0.0040	0.9020	0.5801	1.0	123.0
OFB	0.3576	0.2227	0.9904	0.6624	0.0096	0.9304	0.3334	1.0	77.334
XOR	0.3619	0.1458	0.9996	0.9634	0.0004	0.9383	0.2532	1.0	67.834
ALL	0.5148	0.3171	0.9968	0.9118	0.0032	0.9320	0.4705	1.0	78.667

Table 5.7. HP ENVY m4-1015dx Simulated Load Analysis Results 50% Load

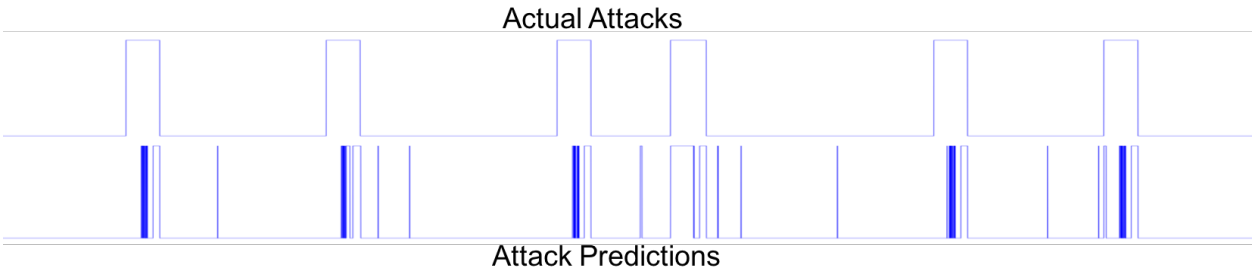

Figure 5.8. HP ENVY m4-1015dx Simulated Load Analysis CFB Time Series Plot (50% Load)

current system CPU load to determine which models to use for prediction confidence scores. Further analysis of the test files shows that the average CPU load placed on the system as a direct result of the simulated ransomware process was 18.52%. Additionally, the average load placed on the system CPU due to background processes was found to be 9.85%. The process of selecting the models, given the accuracy of the previous calculations, likely resulted in most predictions for both "normal operation" and "under attack" being performed by the model trained with no simulated CPU load. However, there were occurrences were the background processes reached as high as 23.1% likely resulting in a model being selected for prediction which had been trained with a simulated load resulting in incorrect predictions. The specificity of the tests were very high along with fallout values that were very low. When the high precision values are also taken into account it shows that the ensemble prediction model rarely made positive predictions. However, when positive predictions were made they were mostly correct. Figure 5.6 illustrates how the majority of positive predictions occurred during attack periods with several isolated false positive predictions also occuring.

Table 5.6 and table 5.7 show the results for the simulated load analysis at 25% and 50% respectively. They generally display the same trends found in the analysis performed at 0%. The sensitivity and fallout are low, the specificity and precision are high, and the MCC scores are slightly lower than the scores recorded during the accuracy and robustness analyses. It would appear that most predictions were carried out by the model most closely associated with the actual state of the system, but the models themselves have performance limitations in this system. The limits of the ensemble prediction model's success, when used on this system, is most likely attributed to the very low number of reported sensor values, 9, along with all but one of the sensors directly measuring CPU performance. Figures 5.7 and 5.8 illustrate how they predictions are made with some resemblance to the actual attacks, but that the ensemble prediction models used on this system have notable false positive and false negative prediction rates.

All configurations tested at 0%, 25%, and 50% simulated loads caught every attack instance except for two, the ECB and CBC tests performed at 25%. The ECB and CBC

ensemble prediction models only caught two thirds of the attacks in both instances. These two instances also have exceptionally low sensitivity and fallout scores leading to the conclusion that they were extremely conservative in making positive predictions. However, most configurations caught all attack instances with average attack recognition times of as little as 5.34 seconds and no more than 142.83 seconds and mostly displayed high precision with low fallout. Despite this system's limitations the ensemble prediction model did perform in a manor which could be seen as successful. The system's limitations, including the lack of diversity among the reported sensors and the small number of reported sensors, is addressed when testing the second system, a MacBook Air. The MacBook Air contains 67 reported sensor values which are almost evenly distributed between the CPU, memory, hard drive, battery, motherboard, and various other elements of the system. It is hypothesized that the second system will be able to implement both simple predictive models and the simulated load ensemble prediction model with a much higher level of success than the first system.

5.2. MacBook Air 13-Inch Mid 2013

5.2.1. Stratified 10-Fold Cross Validation

The results of the cross validation analysis for the MacBook Air 13-inch mid 2013, shown in table 5.8, reinforce the hypothesis that the larger and more diverse number of sensors result in better predictive models. Every machine learning algorithm, except for k-means clustering, performed better than the highest performing machine learning algorithm tested on the HP ENVY m4-1015dx. The more complex tree based algorithms, random forest and extra tree, performed the best with MCC scores of 0.9987 and 0.9985 respectively. The multilayer perceptron once again ranked high in the cross validation results at an MCC score of 0.9984, which is very near the top two performers. Unlike the first system tested, the one-vs-one algorithm top performing configuration was trained as a multiclass classifier and scored an MCC of 0.9983 which is again very near the algorithms above it. However, the one-vs-rest optimal configuration was once again trained as a binary classifier and matched

Algorithm	**Highest MCC**	**Configuration**
Rand Forest	0.9987	Binary Classification, Min-Max Scaling, No Dimensionality Reduction, No Moving Average
Extra Tree	0.9985	Binary Classification, No Scaling, Feature Selection Upper 90%, Weighted Moving Average Window Size 2
MLP	0.9984	Binary Classification, Standardized Scaling, Feature Selection Upper 90%, Weighted Moving Average Window Size 2
One-V-One	0.9983	Multiclass Classification, Min-Max Scaling, No Dimensionality Reduction, No Moving Average
Log Reg	0.9979	Binary Classification, Standardized Scaling, Feature Selection Upper 50%, No Moving Average
One-V-Rest	0.9979	Binary Classification, Standardized Scaling, No Dimensionality Reduction, No Moving Average
SVC	0.9979	Binary Classification, Standardized Scaling, No Dimensionality Reduction, No Moving Average
KNN	0.9975	Binary Classification, Standardized Scaling, Feature Selection Upper 70%, No Moving Average
Decision Tree	0.9970	Multiclass Classification, No Scaling, Feature Selection Upper 70%, No Moving Average
Lin Reg	0.9965	Binary Classification, No Scaling, No Dimensionality Reduction, No Moving Average
N Bayes	0.9917	Multiclass Classification, No Scaling, Feature Selection Upper 70%, Weighted Moving Average Window Size 2
K-Means	0.8327	Binary Classification, Normalization Scaling, No Dimensionality Reduction, No Moving Average

Table 5.8. MacBook Air Stratified 10-Fold Cross Validation Results

the results of the simple SVC implementation. The logistic regression optimal model was only ranked fifth unlike the previous system were it was ranked first. K-means clustering once again scored the lowest MCC at 0.8327 which was the only algorithm to score lower than 0.99. The optimal configurations of three different algorithms were trained as multiclass classifiers where only one such optimal configuration was determined in the previous system. Additionally, eight algorithm optimal configurations included no moving average where there was no such optimal configurations determined in the previous system. This is likely due to the models generating less isolated erroneous results which are accounted for by the moving average. Implementing a moving average actually decreases the performance as the predictive models become less responsive. The results of the cross validation lead to the conclusion that there is a high likelihood that the predictive models in this system will perform at a higher level than the predictive models in the previous system.

5.2.2. Accuracy Analysis

Algorithm	**MCC**	**Sen**	**Spec**	**Prec**	**Fall**	**Acc**	**F1**	**RAR**	**MTAR**
Extra Tree	0.9981	0.9983	0.9998	0.9985	0.0002	0.9996	0.9984	1.0	0.4794
KNN	0.9980	0.9978	0.9998	0.9986	0.0002	0.9995	0.9982	1.0	0.5812
Log Reg	0.9980	0.9983	0.9997	0.9983	0.0003	0.9995	0.9983	1.0	0.4924
Rand Forest	0.9980	0.9982	0.9997	0.9984	0.0003	0.9995	0.9983	1.0	0.4539
MLP	0.9970	0.9988	0.9993	0.9961	0.0007	0.9992	0.9975	1.0	0.3891
Lin Reg	0.9964	0.9967	0.9995	0.9971	0.0004	0.9992	0.9970	1.0	0.5600
One-V-Rest	0.9916	0.9988	0.9980	0.9871	0.0021	0.9980	0.9924	1.0	0.4044
SVC	0.9914	0.9988	0.9978	0.9868	0.0022	0.9979	0.9922	1.0	0.4050
N Bayes	0.9790	0.9939	0.9933	0.9737	0.0067	0.9933	0.9802	1.0	1.511
Decision Tree	0.9687	0.9929	0.9911	0.9564	0.0089	0.9913	0.9713	1.0	0.4113
One-V-One	0.9528	0.9604	0.9921	0.9599	0.0079	0.9881	0.9588	1.0	0.5700
K-Means	0.7743	0.5838	0.6380	0.5810	0.3620	0.6302	0.5448	0.9900	54.31

Table 5.9. MacBook Air Accuracy Analysis Results

The accuracy analysis, shown in table 5.9, shows that the extra tree model performed marginally better than the KNN, logistic regression, and random forest models. The extra tree model scored an MCC of 0.9981 while the KNN, logistic regression, and random forest models all scored 0.9980. When comparing the scores of the accuracy analysis in this system to the scores in the previous system it can clearly be concluded that the models perform at a much higher level. The extra tree model had an accuracy of 99.96%, a precision of 99.85%, a sensitivity of 99.83%, and a fallout of 0.02% which indicate it is very nearly a perfect prediction model. However, it must be noted that the scores in the analyses performed prior to the simulated load analysis are performed with no additional user activity and minimal background processes running with a predictive model trained under the same conditions. This allows the predictive models to very easily differentiate between periods of "no attack" and "under attack". However, the extremely high performance does indicate the potential for high performance if implemented correctly with additional user activity. One of the most promising results is that the extra tree model not only caught every attack instance, but had an average attack recognition of 0.4794 seconds. The previous system took around one minute to recognize an attack with almost entirely CPU related sensors. It would appear that the sensor array in this system allows the predictive models to determine an encryption sensor signature which is far more robust and responsive. In fact, the only algorithm model which had an average attack recognition of over a second was the k-means clustering algorithm. The predictive model must find a sensor signature for periods of "under attack" in which there are periods of time where the simulated ransomware process is walking file directories looking for target files and periods of time where the simulated ransomware process is actually encrypting. Therefore, the sensor values associated with such components as the CPU, memory, hard drive, motherboard, and power supply may persist in such a way during the entire simulated ransomware process that "under attack" predictions are made in both scenarios as evidenced by the sensitivity of the extra tree model being 0.9983. Additionally, although there are minimal background processes running there are certain OSX background processes, such as Spotlight indexing, which operate in a way that should

cause a false positive result if the prediction models were using simple sensor signatures like a rise in CPU sensor values or increased hard drive activity. However, the fallout of 0.0002 indicates that very rarely a false positive prediction was made which shows that essential OSX background processes generally do not cause errors. Figure 5.9 shows the prediction time series for the highest single encryption method tested, CFB. In this time series the MCC score was 0.9994 with an accuracy of 0.9998, a precision of 0.9997, a sensitivity of 0.9999, a specificity of 0.9999, and a fallout of 0.0001. Only one instance of a false positive prediction can be easily noticed prior to the fourth attack instance. Otherwise, the two time series plots are very nearly identical. Figure 5.10 shows the prediction time series for the lowest single encryption method tested, OFB. In this time series the MCC score was 0.9971 with an accuracy of 0.9993, a precision of 0.9983, a sensitivity of 0.9966, a specificity of 0.9997, and a fallout of 0.0002. In this time series only a few false positive predictions are easily noticed along with some attack instances which appear to have some false negatives.

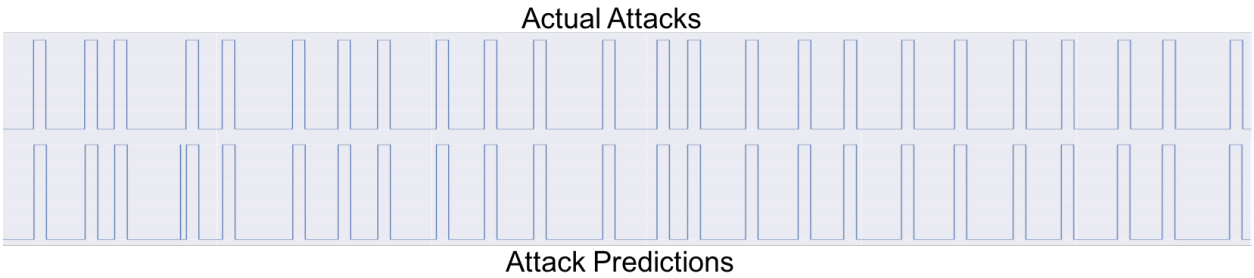

Figure 5.9. MacBook Air Extra Tree CFB Trained and Tested Time Series

5.2.3. Robustness Analysis

The extra tree model and the logistic regression model both scored the highest MCC with 0.9980 as shown in table 5.10. However, the extra tree model had an average attack recognition of 0.4667 seconds to the logistic regression model's 0.4733 seconds. The random forest model had an MCC score of 0.9979, nearly identical to the top two models, and only had an average attack recognition time of 0.4530 seconds. However, the random forest model

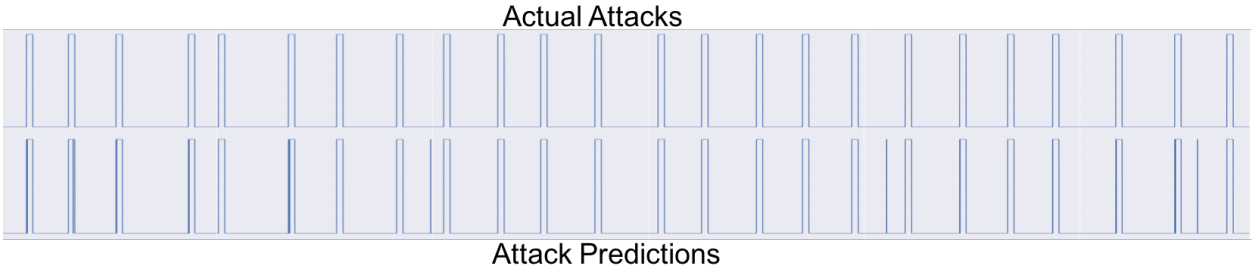

Figure 5.10. MacBook Air Extra Tree OFB Trained and Tested Time Series

Algorithm	**MCC**	**Sen**	**Spec**	**Prec**	**Fall**	**Acc**	**F1**	**RAR**	**MTAR**
Extra Tree	0.9980	0.9982	0.9998	0.9983	0.0002	0.9996	0.9983	1.0	0.4667
Log Reg	0.9980	0.9983	0.9997	0.9982	0.0003	0.9995	0.9983	1.0	0.4733
Rand Forest	0.9979	0.9981	0.9997	0.9982	0.0003	0.9995	0.9982	1.0	0.4530
KNN	0.9978	0.9978	0.9998	0.9985	0.0002	0.9995	0.9981	1.0	0.5739
Lin Reg	0.9960	0.9959	0.9996	0.9971	0.0004	0.9991	0.9965	1.0	0.5613
One-V-Rest	0.9958	0.9987	0.9990	0.9938	0.0010	0.9990	0.9960	1.0	0.4070
SVC	0.9958	0.9987	0.9990	0.9938	0.0010	0.9990	0.9960	1.0	0.4078
N Bayes	0.9839	0.9936	0.9958	0.9801	0.0042	0.9956	0.9856	1.0	1.520
Decision Tree	0.9559	0.9942	0.9786	0.9428	0.0214	0.9810	0.9593	1.0	0.4010
MLP	0.9378	0.9818	0.9732	0.9317	0.0268	0.9743	0.9405	1.0	0.4207
One-V-One	0.9040	0.9504	0.9706	0.9008	0.0294	0.9681	0.9135	1.0	0.6223
K-Means	0.7681	0.5649	0.6405	0.5943	0.3595	0.6303	0.5333	1.0	55.73

Table 5.10. MacBook Air Robustness Analysis Results

had a slightly higher fallout score which in the context of this experiment is more detrimental than the advantage of catching attack instances 0.0137 seconds faster. The extra tree model only had an MCC score decrease of 0.0001 under the robustness analysis. Additionally, the average time to attack recognition dropped 0.0127 seconds under the robustness analysis from 0.4794 under the accuracy analysis. Therefore, the extra tree model works for encryption methods that it has not been trained for almost as well, if not equally as well, as encryption methods it has been trained for. Considering the extra tree model performed the best under the accuracy analysis and the robustness analysis it was made the selection for the training time and simulated load analyses. Figure 5.11 shows the prediction time series for the highest performing extra tree model which was trained with a single encryption method and tested with a different single encryption method, ECB and CFB respectively. In this time series the MCC score was 0.9991 with an accuracy of 0.9996, a precision of 0.9994, a sensitivity of 0.9991, a specificity of 0.9998, and a fallout of 0.0002. Only a few false positive instances can be easily seen as well as a small number of attack instances with noticeable false negatives. It is also important to note that the extra tree model performed the best overall for the first system when using rate of attack recognition and average time to attack recognition as the primary metrics of performance.

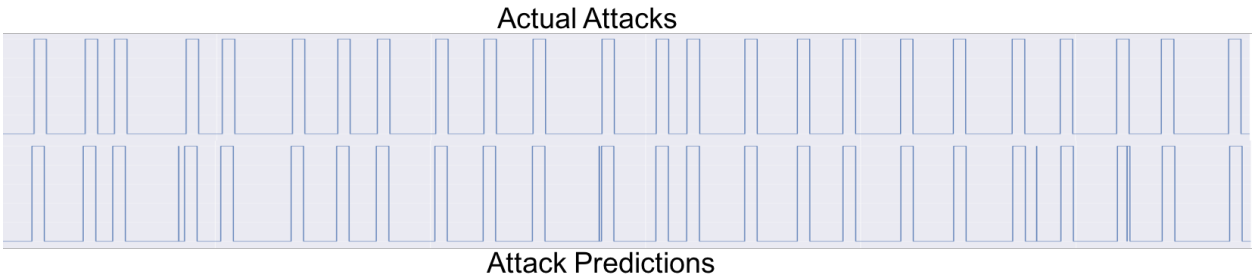

Figure 5.11. MacBook Air Extra Tree ECB Trained CFB Tested Time Series

5.2.4. Training Time Analysis

Training Time (Hours)	**MCC**	**Sen**	**Spec**	**Prec**	**Fall**	**Acc**	**F1**	**RAR**	**MTAR**
2	0.9972	0.9964	0.9998	0.9989	0.0002	0.9993	0.9976	1.0	0.6667
4	0.9982	0.9982	0.9998	0.9987	0.0002	0.9996	0.9985	1.0	0.5083
6	0.9984	0.9984	0.9998	0.9988	0.0002	0.9996	0.9986	1.0	0.4833
8	0.9984	0.9984	0.9998	0.9988	0.0002	0.9996	0.9986	1.0	0.4583
10	0.9984	0.9985	0.9998	0.9987	0.0002	0.9996	0.9986	1.0	0.5083
12	0.9984	0.9987	0.9998	0.9986	0.0002	0.9996	0.9986	1.0	0.4500
14	0.9985	0.9987	0.9998	0.9987	0.0002	0.9996	0.9987	1.0	0.4083
16	0.9984	0.9985	0.9998	0.9988	0.0002	0.9996	0.9987	1.0	0.4833
18	0.9983	0.9987	0.9997	0.9985	0.0003	0.9996	0.9986	1.0	0.4333
20	0.9985	0.9988	0.9998	0.9986	0.0002	0.9996	0.9987	1.0	0.3917
22	0.9984	0.9987	0.9998	0.9986	0.0002	0.9996	0.9987	1.0	0.4417
24	0.9984	0.9985	0.9998	0.9988	0.0002	0.9996	0.9987	1.0	0.4833

Table 5.11. MacBook Air Training Time Analysis Results

The training time analysis for this system fluctuated very little from two hours of training to twenty four hours of training. In fact, it can be seen in table 5.11 seven of the twelve time intervals had the same MCC score of 0.9984. Figure 5.12 shows the training time results along with a red line indicating the average MCC score for all models trained between one and two hours in intervals of one minute. There is a noticeable difference between the location of the red line and the score of the model trained at twenty four hours. Thus, two hours would once again be the minimum time needed to generate a model which performed at a comparable level to the model trained at twenty four hours used in the accuracy and robustness analyses. In the same way as the first system the training time for each encryption method during the simulated load training data collection will be two hours at each simulated load level.

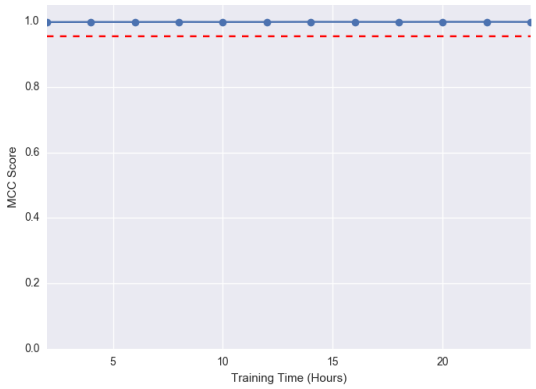

Figure 5.12. MacBook Air 24 Hour Training Time Analysis

5.2.5. Simulated Load Analysis

Early in the simulated load testing for the second system it was found that many isolated erroneous predictions were occuring with the ensemble predictive model. In order to smooth out isolated errors the window size for the weighted moving average calculation was increased until a reasonable reduction in isolated errors was reported. The original prediction model implemented a weighted moving average with a window size of 2. It was found that the isolated occurrences of both false positives and false negatives were greatly reduced with

Encryption	MCC	Sen	Spec	Prec	Fall	Acc	F1	RAR	MTAR
ECB	0.9410	0.9776	0.9833	0.9265	0.0167	0.9823	0.9514	1.0	8.6667
CBC	0.9561	0.9483	0.9956	0.9797	0.0043	0.9421	0.8061	1.0	3.1667
CFB	0.7238	0.5743	0.9996	0.9971	0.0004	0.9240	0.7288	1.0	6.6667
OFB	0.8430	0.8172	0.9854	0.9239	0.0145	0.9555	0.8673	1.0	1.3334
XOR	0.6985	0.5337	0.9999	0.9992	0.0001	0.9226	0.6957	1.0	5.8334
ALL	0.6098	0.4207	0.9996	0.9960	0.0003	0.8966	0.5916	1.0	9.0

Table 5.12. MacBook Air Simulated Load Analysis Results 0% Load

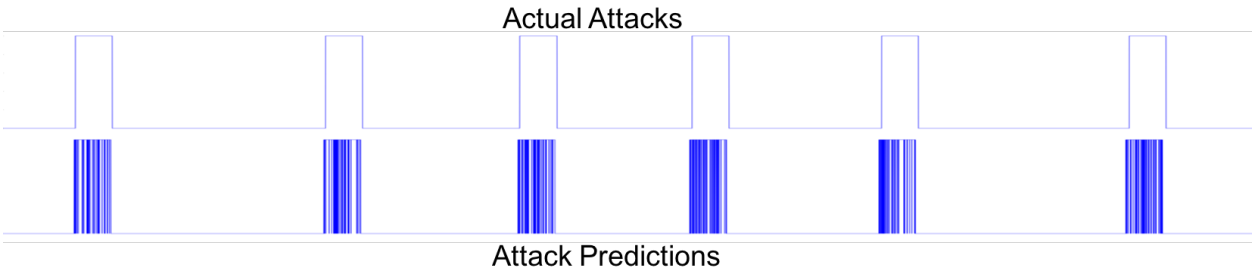

Figure 5.13. MacBook Air Simulated Load Analysis ECB Time Series Plot (0% Load)

Encryption	MCC	Sen	Spec	Prec	Fall	Acc	F1	RAR	MTAR
ECB	0.9368	0.9568	0.9867	0.9395	0.0132	0.9814	0.9480	1.0	13.0
CBC	0.9267	0.9856	0.9751	0.8972	0.0248	0.9770	0.9393	1.0	6.0
CFB	0.9134	0.9343	0.9748	0.9545	0.0252	0.9601	0.9443	1.0	10.834
OFB	0.9303	0.9810	0.9776	0.9073	0.0224	0.9782	0.9427	1.0	8.0
XOR	0.9237	0.8808	0.9989	0.9944	0.0011	0.9784	0.9342	1.0	10.167
ALL	0.9132	0.8844	0.9949	0.9794	0.0051	0.9711	0.9295	1.0	11.867

Table 5.13. MacBook Air Simulated Load Analysis Results 25% Load

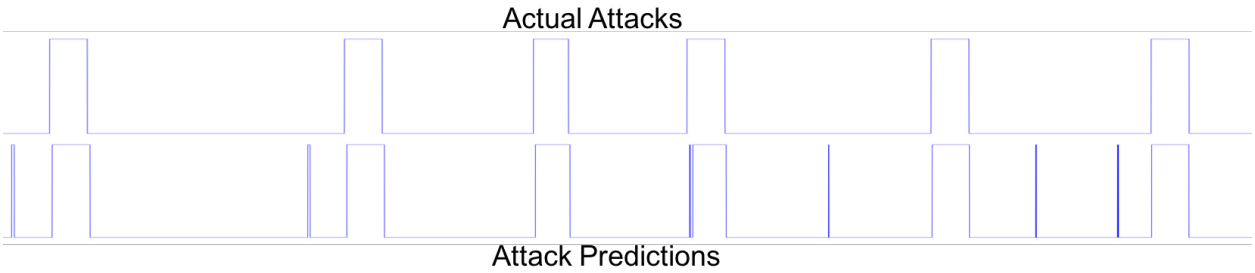

Figure 5.14. MacBook Air Simulated Load Analysis ECB Time Series Plot (25% Load)

Encryption	**MCC**	**Sen**	**Spec**	**Prec**	**Fall**	**Acc**	**F1**	**RAR**	**MTAR**
ECB	0.9693	0.9776	0.9939	0.9720	0.0060	0.9910	0.9748	1.0	8.8334
CBC	0.9590	0.9875	0.9838	0.9509	0.0162	0.9847	0.9847	1.0	7.0
CFB	0.9164	0.9925	0.9546	0.8897	0.0454	0.9648	0.9383	1.0	7.75
OFB	0.9408	0.9874	0.9739	0.9249	0.0261	0.9772	0.9551	1.0	7.1667
XOR	0.9745	0.9855	0.9927	0.9749	0.0073	0.9911	0.9802	1.0	7.5
ALL	0.9518	0.9855	0.9820	0.9405	0.0180	0.9828	0.9625	1.0	7.5

Table 5.14. MacBook Air Simulated Load Analysis Results 50% Load

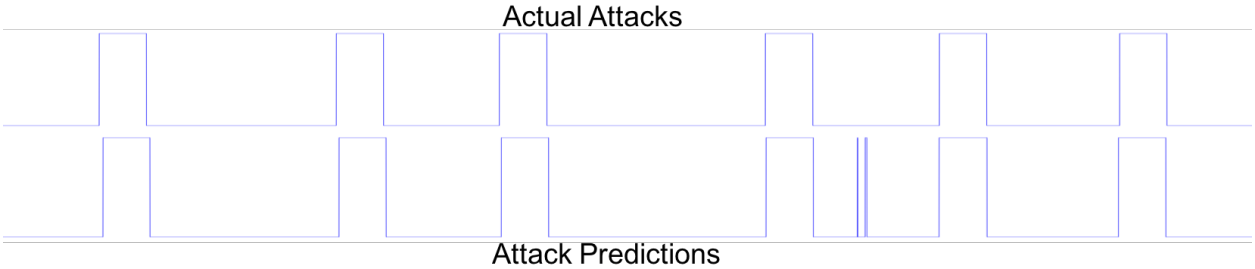

Figure 5.15. MacBook Air Simulated Load Analysis XOR Time Series Plot (50% Load)

a weighted moving average of window size 30. This resulted in a much more conservative prediction model which resulted in a great decrease in false positive and false negative predictions. However, the more conservative model came at the cost of a potential decrease in sensitivity and specificity and a high probability of the average attack detection time increasing.

Table 5.12 shows the results for the implementation of the ensemble prediction model with no simulated load applied to the system. The MCC scores are noticeably lower than the MCC scores recorded in both the accuracy and robustness analyses. This reduced score is due to the ensemble method having to determine whether the current CPU load of the system is due to background processes and additional user load or the actual simulated ransomware process where previously it was known that no additional load was being applied to the system. Analysis of the test files with a 0% simulated load show that background processes resulted in an average CPU load of 5.2021% with a peak load of 46.3%. It can easily be seen that the ensemble prediction model, when tested with no simulated load, generally results in a very high specificity and a very low fallout. Additionally, the sensitivity is generally at or higher than 50%, but because it is much lower than in previous tests the MCC score is also relatively much lower. This leads to the conclusion that the ensemble prediction model makes positive predictions very conservatively when tested with no additional system CPU load. However, when positive predictions are made they tend to be correct as evidenced by the very high precision scores. Each test configuration at a simulated load of 0% caught every attack with an average attack recognition time as low as 1.3334 seconds and no higher than 9.0 seconds. This is a major improvement upon the ensemble prediction model implemented in the first system. Figure 5.15 illustrates the way the ensemble predictive model has a high level of precision and a low level of fallout at 0% simulated load. Analysis of the test results for a simulated load level of 0% can be interpreted to mean the ensemble prediction model is a competent detector of the simulated ransomware process at and around that particular additional user load level.

Table 5.13 shows the results for the analysis with a simulated load level of 25%. The MCC scores are noticeably higher than they were at a simulated load of 0%. When comparing the results from the two different load levels it can be seen that the sensitivity is much higher at 25%. Additionally, the fallout is also relatively higher at 25%, but is still very low as most test configurations were between 0% and 3%. The much higher specificity scores at the cost of a slightly higher fallout score resulted in the increased MCC scores. Figure 5.13 illustrates the way the ensemble predictive model has a high level of sensitivity at 25% simulated load as the periods of attack are correctly predicted at a high rate. However, the instances of false positive predictions can easily be seen which are the result of increased sensitivity at the cost of slightly increased fallout. Analysis of the test results for a simulated load level of 25% can be interpreted to mean the ensemble prediction model is a competent detector of the simulated ransomware process at and around that particular additional user load level.

Table 5.14 shows the results for the simulated load analysis at 50%. The MCC score at 50% are even higher than the MCC scores at 25%. The sensitivity scores are about the same as the scores at 25%, but the fallout scores are considerably lower at 50%. This means that the ensemble predictive model performed at a very high level when tested with a 50% simulated load. Figure 5.15 illustrates just how well the ensemble prediction model performed when trained and tested to detect the simulated ransomware process using XOR encryption. There is only one noticeable period in which false positive predictions were made. Analysis of the test results for a simulated load level of 50% can be interpreted to mean the ensemble prediction model is a competent detector of the simulated ransomware process at and around that particular additional user load level.

The average CPU load placed on the system which is a direct result of the simulated ransomware process averaged 34%. The model closest to the current CPU system load and the model closest to the current CPU system load after subtracting 34% must act together to determine if the current sensor signature more closely matches a state of "normal operation" or a state of "under attack". The results indicate that, as predicted, the majority of the time the same predictive model is selected to provide both the "normal operation" and "under

attack" confidence scores as the unaccounted for background processes in this test generally only placed a load of 5% on the CPU. The observation that background processes play a very small role in determining the prediction models selected shows that the most important variable in determining which models most accurately make predictions of "normal operation" and "under attack" for a given system state are the processes created through user interaction. This analysis showed that an ensemble of models trained at different simulated load levels which represent user generated processes through interaction is effective at predicting the simulated ransomware process when the additional load on the system is within about 5% to 10% of the load level one of the models was trained to interpret. Training models at more frequent intervals from 0% to 100% simulated load is most likely to result in the ensemble prediction model performing as intended with a user present who is capable of placing a varying degree of additional load on the system.

The presence of a large and diverse array of sensors likely caused the high level of predictive performance seen in the MacBook Air system. Machine learning algorithms, when provided with good training data, are able to find and exploit patterns in the data which we as humans are not able to easily understand. In this system the feature importance values were ranked in an effort to determine which sensors were most valued in accurately predicting the presence of the simulated ransomware process. The highest ranking sensor value by a large margin, as determined by the extra tree predictive models, was the WLAN card power sensor. In all likelihood even a seasoned engineer or computer scientist would not suspect the WLAN card power to serve a purpose when modeling ransomware attacks much less be the most important. The six sensors appearing the highest in the ranking were all power sensors. The top six sensors in order of importance were WLAN card power (0.1920), system supply power (0.1488), DDR3 memory line power (0.1004), CPU total package power core 1 (0.0998), CPU high side power (0.0994), and CPU total package power core 2 (0.0805). The WLAN card temperature ranked tenth at a score of 0.0356 which was the highest ranked temperature sensor. Additionally, the WLAN card current ranked fifteenth at 0.0019 which made three of the four WLAN card sensors rank in the top fifteen. However, the WLAN card

voltage sensor tied for the least important with an importance score of 0. It would appear that the large array of sensors allows the predictive model to exploit some collective power signature in the system which is the most accurate method for predicting the presence of the simulated ransomware process in both its file walking state and actual encryption state.

Chapter 6

CONCLUSION

System side channel data has commonly been used to attack systems in which an attacker has knowledge of how a process is physically carried out on a machine. Instead of attacking a system with side channel data it has been shown in this experiment that it is possible to defend a system by training machine learning algorithms to detect intricate patterns in the physical behavior of a system which correlate to a malicious process. Additionally, once the machine learning algorithms have been trained and a predictive model has been generated, predictions about the state of a system are calculated quickly and with a low computational overhead.

Perhaps the most important aspect of this experiment is the speed in which an attack may be detected. In experimental testing, the highest performing system had an average time to attack recognition which was as little as 1.3 seconds and never exceeded 12 seconds in even the lowest performing models. However, it has been found that even the best predictive models generate some false positive predictions. For this reason it is believed that the most effective method for implementing this technique would occur when more detailed analysis follows the indication of a positive prediction. When the quick acting predictive model indicates a positive prediction a suspected process may then be checked against a list of known good processes which act in a similar manner as ransomware, but with non-malicious intentions. If the targeted process is not on the list more detailed data signature analysis may be performed to determine if the process is in fact malicious. In this way the computational overhead of having a background process constantly performing the in-depth data signature analysis is avoided and only performed when the physical state of the system indicates a ransomware like process is currently running.

The performance of the method presented in this experiment may also be augmented by creating an ensemble predictive model which includes a higher number of individual models which have been trained at various simulated loads. It has been shown that the ransomware process generally consumes a predictable amount of the system's CPU resources. Additionally, it has been shown that the typical combination of background processes utilize CPU resources at a low enough level that models trained at a certain simulated CPU load are capable of making accurate predictions for a small range of total system CPU loads. It is highly likely that implementing an ensemble predictive model with individual models trained from a simulated load of 0% to a simulated load of 100% at increments of 5% to 10% will result in an ensemble model capable of accounting for the highly dynamic nature of additional user load due to interaction.

While this experiment focused on detecting ransomware attacks with side channel data it may be possible to apply this method in a more broad manner to detect different processes which have predictable behavior. The predictive models only require training data which demonstrates the behavior of the system in a "normal" state and also demonstrates the behavior of the system when the target process is active. This idea opens up the possibility for advanced sensor based monitoring of a system which could include sensors that have been added to the system for the sole purpose of augmenting its ability to physically model a computational process.

BIBLIOGRAPHY

[1] Abrams L. "Cryptolocker Ransomware Information Guide and Faq". http://www.bleepingcomputer.com/virus-removal/cryptolocker-ransomware-information, 2013. Note: This is an electronic document. Date of publication: [October 14, 2013];.

[2] Alharbi A, Thornton M. "Demographic Group Classification of Smart Device Users". *IEEE Int. Conf. on Machine Learning and Applications* (Dec 2015), 481–486.

[3] Altman N. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression". *The American Statistician 46*, 3 (Aug 1992), 175–185.

[4] BBCNews. "Cyber-attack: Europol Says it was Unprecedented in Scale". http://www.bbc.com/news/world-europe-39907965, 2017. Note: This is an electronic document. Date of publication: [May 13, 2017];.

[5] Bellare M. "Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements". Tech. rep., Springer Berlin Heidelberg, 2000.

[6] Bonderud D. "CryPy Ransomware Slithers Onto PCs With Unique, Python-Based Encryption". https://securityintelligence.com/news/crypy-ransomware-slithers-onto-pcs-unique-python-based-encryption/, 2016. Note: This is an electronic document. Date of publication: [October 17, 2016];.

[7] Brandon R. "Petya Ransomware Authors Demand $250,000 in First Public Statement Since the Sttack". https://www.theverge.com/2017/7/5/15922216/petya-notpetya-ransomware-authors-bitcoin-demand-decrypt, 2017. Note: This is an electronic document. Date of publication: [July 5, 2017];.

[8] Bresink M. "Hardware Monitor: Reference Manual". https://www.bresink.com/osx/216202/Docs-en/index.html, 2017. Note: This is an electronic document. Date of publication: [2017];.

[9] Cohen J, Cohen P, West, S, Aiken L. *"Applied Multiple RegressionCcorrelation Analysis for the Behavioral Sciences"*, 2 ed. Lawrence Erlbaum Associates, 2003.

[10] Correa R. "How Fast Does Ransomware Encrypt Files? Faster Than You Think". https://blog.barkly.com/how-fast-does-ransomware-encrypt-files, 2016. Note: This is an electronic document. Date of publication: [April 2016];.

[11] Cortes C, Vapnik V. "Support-Vector Networks". *Machine Learning 20*, 3 (1995), 273–297.

[12] D, C. "Documentation of Scikit-Learn 0.18". Tech. rep., 2017.

[13] Daemen J, Rijmen V. "AES Proposal: Rijndael". Tech. rep., National Institute of Standards and Technology, Feb 21, 2013.

[14] Demme J, Maycock M, Schmitz J, Tang A, Waksman A, Sethumadhavan S, Stolfo S. "On the Feasibility of Online Malware Detection with Performance Counters". *40th Annual Int. Symposium on Comp. Arch* (June 2013).

[15] Devcic J. "Weighted Moving Averages: The Basics". http://www.investopedia.com/articles/technical/060401.asp. Note: This is an electronic document. Date of publication: [2009];.

[16] Draper N, Smith H. *"Applied Regression Analysis"*, 3 ed. John Wiley, 1998.

[17] Freedman D. *"Statistical Models: Theory and Practice"*, 2 ed. Cambridge University Press, 2009.

[18] Geron A. *"Hands-On Machine Learning with Scikit-Learn & TensorFlow"*, 1 ed. O'Reilly, 2017.

[19] Geurts P, Ernst D, Wehenkel L. "Extremely Randomized Trees". *Machine Learning 63*, 1 (Apr 2006), 3–42.

[20] Golden T. "WMI 1.4.9". http://timgolden.me.uk/python/wmi/index.html, 2003. Note: This is an electronic document. Date of publication: [2003];.

[21] Ho T. "Random Decision Forests". In *Proceedings of the 3rd International Conference on Document Analysis and Recognition* (The address of the publisher, Aug 1995), pp. 278–282.

[22] Hunter J. "Matplotlib". https://matplotlib.org/, 2003. Note: This is an electronic document. Date of publication: [2003];.

[23] I, F. "A Survey of Dimension Reduction Techniques". Tech. rep., June 2002.

[24] Kanungo T, Mount D, Netanyahu N, Piatko C, Silverman R, Wu A. "An Efficient K-Means Clustering Algorithm: Analysis and Implementation". *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Jul 2002), 881–892.

[25] Loden J, Daeschler D, Rodola G. "Psutil". https://pythonhosted.org/psutil/, 2009. Note: This is an electronic document. Date of publication: [2009];.

[26] Mehmood S. "Enterprise Survival Guide for Ransomware Attacks". Tech. rep., The SANS Institute, April 30, 2016.

[27] Mohri M, Rostamizadeh A, Talwalkar A. *"Foundations of Machine Learning (Adaptive Computation and Machine Learning series) "*, 1 ed. The MIT Press, 2012.

[28] NIST. "Announcing the Advanced Eencryption Standard (AES)". Tech. rep., Nov 26, 2001.

[29] O'Gorman G, McDonald G. "Ransomware: A Growing Menace". Tech. rep., Symantec Corporation, 2012.

[30] Oliphant T. "NumPy". http://www.numpy.org/, 2006. Note: This is an electronic document. Date of publication: [2006];.

[31] Packard H. "HP ENVY m4-1015dx Notebook PC Product Specifications". https://support.hp.com/gb-en/document/c03511047, 2012. Note: This is an electronic document. Date of publication: [2012];.

[32] Pedregosa F, Varoquaux G, G. A. M. V. "Scikit-Learn User Guide". Tech. rep., 2010.

[33] Raschka S. *"Python Machine Learning"*, 1 ed. Packt Publishing, 2015.

[34] Rish I. "An Empirical Study of the Naive Bayes Classifier". Tech. rep., IBM Research Division, Nov 02, 2001.

[35] Rokach L, Maimon O. *"Data Mining With Decision Trees: Theory and Applications"*, 2 ed. World Scientific Pub Co Inc, 2008.

[36] Rosenblatt F. "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". *Psychological Review 65*, 6 (1958), 386–408.

[37] Scaife N, Carter H, Traynor P, Butler K. "Cryptolock (and Drop It): Stopping Ransomware Attacks on User Data". *IEEE Int. Conf. on Dist. Computing Systems* (June 2016).

[38] Schmidhuber J. "Deep Learning in Neural Networks: An Overview". *Neural Networks 61* (Jan 2015), 85–117.

[39] Tang A, Sethumadhavan S, Stolfo S. "Unsupervised Anomaly-based Malware Detection using Hardware Features". *Int. Symposium on Research in Attacks, Intrusions, and Defenses* (Sept 2014).

[40] W, M. "Pandas: Powerful Python Data Analysis Toolkit". Tech. rep., July 07, 2017.

[41] Wyke J, Aijan A. "The Current State of Ransomware". Tech. rep., SophosLabs, December 2015.