

Vulnerabilities Assessment of Machine Learning Based Malware Classification Models

Godwin Raju, Yasir Malik, Pavol Zavarsky
Department of Information Systems Security Management
Concordia University of Edmonton, Edmonton, Alberta, Canada
graju@concordia.ab.ca, {yasir.malik, pavol.zavarsky}@concordia.ab.ca

Abstract— The primary focus of the machine learning model is to train a system to achieve self-reliance. However, the learning phase itself is not secured because of the absence of the inbuilt security functions which results in vulnerability in the model. When a malicious adversary manipulates the input data, it exploits specific vulnerabilities of machine learning phases or algorithms which compromises the entire system security. This paper illustrates how machine learning model reacts to an adversarial machine learning environment. In this research, a machine learning model is introduced to an adversarial machine learning environment to conduct a vulnerability assessment of the malware classification model. The vulnerability assessment is done using Black-Box attack environment.

Keywords— *machine learning; adversarial machine learning; Black-Box attack; poisoning attack; confusion matrix; feature sets; decision boundary; knowledge base; training test; test set.*

I. INTRODUCTION

Machine learning (ML) is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to learn with data and act, without being explicitly programmed. A machine's learning algorithm enables it to identify patterns in observed data, build models that explain the world, and predict things without any pre-programmed rules and models [33].

Numerous applications of ML techniques are adversarial in nature, insofar the objective is to distinguish instances which are “bad” from those which are “good”. Adversarial machine learning (AML) is a research field that lies at the intersection of ML and computer security. AML's primary focus is to find vulnerabilities which are inherent to an ML model, which creates opportunities for attackers, for example, embedding malicious codes during the preliminary stages to produce disruption in the results [3] [5]. Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake. For example, attackers could target autonomous vehicles by using stickers or paint to create an adversarial stop sign that the vehicle would interpret as a ‘yield’ or other sign. Hence, higher the quality of the machine learning approach, higher will be the security level in an organization.

In this research, AML approach known as the “Black-Box attack” is adapted to introduce a poison attack on the supervised ML-based malware classification model. A Black-Box attack is a scenario in which an adversary has full access

to the environment including the feature sets, data sets, algorithms, and location of the decision boundary.

The rest of the paper is organized in the following structure. Section II discusses the preliminary information, section III includes the related work, section IV illustrates the methodology used, section V include the results and their discussion, and section VI concludes the paper.

II. PRELIMINARY

Machine learning models are defined and categorized according to the approach used in training the machine. The following are the most popular approaches used to train machine models.

- a. **A supervised machine learning approach:** In this type of approach, the training environment is strictly controlled, and the machine model is trained using well-defined labels and feature sets. This approach is used in scenarios where the dataset to be trained is of a feasible quantity [18].
- b. **An unsupervised machine learning approach:** In this type of approach, the machine learning model functions under some boundaries or rulesets. However, this approach differs from supervised learning in terms of the dataset provided which are not labeled or well defined and the machine model is responsible for labeling and creating clusters from the unstructured or raw data. This approach is preferred in the situation where creating labels is not feasible (e.g. Network traffic information, self-driving cars) [19].
- c. **A reinforced machine learning approach:** In this approach, a system is provided with dataset and environment similar to the supervised learning approach, but here an incentive model is integrated to the design, which would reward the machine if the machine takes an efficient decision. Thus, the model tries to strive for a better and efficient result in the consecutive run. In this model, the path is decided and improved by the machine with every iteration [20] [21] [22].

All the three approaches of ML depends upon the type (static or dynamic) of dataset trained and is divided into the following two models:

- i. **Static learning model:** It is a machine model in which the system is trained using a batch or static dataset. For

example, if there is a need to train a model on malware detection, then a signature dataset is taken, and the machine model is trained on that batch. This type of ML model has been used to develop the environment in this paper [8].

- ii. *Dynamic/Online learning model*: In this model, the learning phase is a continuous process, the model could start with a static batch, but is incremented routinely. These kinds of models evolve with time and provide more flexibility for the attackers to deploy an attack as the learning model presents higher states of opportunities [8].

Mitigation strategies can be divided into two distinct categories, reactive approaches, and proactive approaches, *reactive approaches* aim to counter post attack scenarios, and includes (i) timely detection of novel attacks, (ii) frequent classifier retraining, and (iii) verification of the consistency of classifier detection against training data and labels. Whereas, proactive approaches aim to prevent future attacks. The categories developed for proactive defense approaches are security by design and security by obscurity. AML assesses the ML model from all paradigms, therefore, policies and protocols are to be designed which should dictate the development location and privileges given to personnel working on the development of the ML model [32]. The following sub-sections describe the attack models available in AML and their countermeasures.

1) Attack models

In AML, the attacks can be classified according to their properties or nature, even though the attacks are classified in different classes they are not mutually exclusive, rather most of them can and are used together [5]. The following are the main categories of the attack models in AML.

- **Influence category**

This category is about understanding the nature of the attack and how it interacts with the ML model, and to which extent it influences the output of the model. This category is further divided into two categories:

- i. *Causative attack*: Altering or manipulation of training process through influence over the training dataset of the ML model.
- ii. *Exploratory attack*: Does not alter the training process but passively observes the process to obtain information which can be used [4].

- **Specificity category**

This category deals with the location or the point of interest of the attack and they have been categorized into:

- i. *Targeted attack*: Location of the attack is specified and focus is on a point or small set of points. There is less flexibility to attack opportunities compared to the indiscriminate attack type.
- ii. *Indiscriminate attacks*: Attack has the much broader spectrum to work. Thus, providing the

attacker with flexibility and range. The attack can be made on a general class of points, such as “any false negative”.

- **Security violation category**

This category is formed by keeping the security triad in mind (confidentiality, integrity, and availability) when dealing with the machine model itself and how AML can harm the security triad.

- i. *Integrity*: An integrity attack results in intrusion points being classified as normal points.
- ii. *Availability*: If a broader class of attack is conducted on integrity and this would result in a high number of false positive and false negatives which would render the system useless [10].

2) AML countermeasures

Following are some of the countermeasures available for the previously mentioned AML attack models along with their limitations:

- *Robustness*: To solve the learning problem which is introduced by the function ‘f’ the model is introduced with a new function λ , this penalizes the model for making any generalization. Thus, provides smoothening of complexity but this smoothening can also be exploited by the attacker.
- *Detecting attacks*: Detecting attacks can be difficult even if the attacker is not trying to hide his attempts, but it is possible to detect the causative attack by running comparison analysis on a special test case, which would alert the user if the model has been compromised by experiencing discrepancies in the result from the actual model and the special test case model.
- *Disinformation*: Any known vulnerable spots or weak logistic points can be left unchecked and filled with information which can later be disregarded. This is a type of honeypot situation regarding machine learning where the machine model is providing the attacker with futile and inaccurate information.
- *Randomization*: The most crucial point in a machine model security is to hide the decision boundary from the attacker if the attacker is able to identify where the boundary resides. In this situation, the attacker may be able to circumvent it. By adding a random function to the decision boundary, the attacker would not be able to attain the accurate location of the decision boundary [4].

The cost of these countermeasures restricts the machine model’s flexibility, expressivity, and constraints. And due to the lack of ability to adapt, the machine model become susceptible to vulnerabilities and attacks.

3) Transferability

Transferability is a property of AML, if one machine model is affected by any attack (causative or influence), the attack can be transferred to a similar machine models working on the same algorithm or has the same training technique [1]. This can be done even if the machine models are using different techniques to train the models if both the machine model has same tasks to perform. The attack focuses on the logical preceptive taken by a machine model as different machine models using different algorithms might do the same task in a unique way, but the result and the logic behind the process would overlap, and transferability uses this overlap to introduce the attack from one model to another [1] [7] [31].

4. The FAIL model

Black-Box attack requires certain requisites for deployment. Happenstance criteria of Black-Box attack restrict or reduces its practicality in a real-world scenario, i.e. the attacker should be an insider working in the team developing the machine model or should obtain a substantial amount of resources to create and simulate an ML model to find the exploits and vulnerabilities.

The FAIL model is a framework for AML which focuses on some prime objectives which would enable the attacker to successfully deploy the attack/intrusive elements into the environment with minimum threshold requirement for resources. By using FAIL model as a framework, an attacker can overcome the limitations of Black-Box attack and ensure deployment of a realistic adversarial model. The four dimensions of FAIL model starts with Feature knowledge (the subset of features known to the adversary), Algorithm knowledge is the second dimension which dictates adversary should possess the knowledge of the algorithm for crafting the poison samples for the model. Label training and clustering falls into the third dimension of Instance knowledge and finally, the last dimension is the leverage, adversary's ability to modify the subset of features which were obtained from the first dimension. The FAIL model also talks about attacker's constraints, which bifurcates attacker's strategy into three categories *i.e. Success, Defense, and Budget*. *Success* constraints encode the attacker's goals and consideration that directly affect the effectiveness of the attack, such as the assessment of the target instance classification. *Defense* constraints allures to the attack characteristics meant to circumvent contemporary defenses used by the system. *Budget* constraints address the limitations in an attacker's resources, such as the ability to create instances, the maximum number of poison attack deployable, processing power residing with the attacker, the maximum number of queries to the victim model [9] [26].

1. Implementation of the FAIL model's dimensions

- **Questions for the "F" dimension:** *what features are kept as a secret? Is access to the exact feature value possible?* For example, in some cases, the feature subsets are not publicly available, or might not be directly defined from instances not available to the attacker (E.g. low-frequency word features).
- **Questions for the "A" dimension:** *Is the algorithm class known? Is the training algorithm secret? Are the classifier parameters secret?* These questions define the spectrum for adversarial knowledge about the learning algorithm. (Black-Box, where the knowledge is public, Gray-Box, where the knowledge is partial and White-Box, where the adversary has no information about the algorithm used).
- **Questions for the "I" dimension:** *Is the entire training set known? Is the training set partially known? Are the instances known to the attacker sufficient to train a robust classifier?* **This** dimension controls the intersection between the instances available to the attacker and which are used/implemented by the victim. The application might use instances from a public domain (E.g. a vulnerability exploit predictor) and the attacker could leverage them to the full extent to derive their attack strategy.
- **Questions for the "L" dimension:** *Which features are modifiable by the attacker?* This dimension is about the capability of the attacker to craft attack samples. In case of some applications, the attacker would not be able to modify the certain type of features, either because the attacker does not have the control over the generating process or when a modification is done, the integrity of the instance is compromised. (E.g. watermark on the model which prevents the attacker from modifying certain features).

5. Zero-day Evasion Attack Analysis (Arms Race)

Deep neural networks (DNNs) outputs excellent performances in ML tasks such as image recognition, pattern recognition, speech recognition, and intrusion detection. As the adversarial examples are a huge threat to the DNNs, therefore, studies are conducted to examine these adversarial examples and defense strategies which could be implemented to circumvent adversarial threats. Zero-day adversarial examples are created with the new test data and are unknown to the classifiers. Hence, they represent a more significant threat to DNNs. In [27], Zero-day adversarial examples are studied and the correlation with the defense strategies helps in deploying countermeasures for the model against Zero-day adversarial examples. The results discovered shows that, if the attacker has real-time knowledge of the model, the success rate of adversarial attack is almost 100%. Similarly, if the

model is aware of the attack in real time, it can reduce the attack, or the impact associated significantly [25] [27].

III. RELATED WORK

Machine Learning is integrated into most of the technologies and Network intrusion detection system (NIDS) is one of the security-based technology, which can greatly benefit from ML [23]. By integrating ML with NIDS, an intrusion detection system can improve detection system while requires a minimum amount of maintenance with the ability of the system to learn from the datasets and the parameters provided to them. But a machine can be trained using a malicious dataset which would degrade the quality of work produced by the model, and there is no innate functionality present in ML phase which would be able to detect or comprehend the motive behind an input. In a situation where an attacker has access to the ML model or can attack the ML model in order to ‘*mis-train*’ the NIDS. In this case, the NIDS would grant access to malicious entities and might deny access to the legitimate user. Moreover, doing so would not trigger any alerts or flags as the system would consider these events as normal events because it was trained to do so [2]. This concept is known as adversarial machine learning (AML). Using AML, an attacker can degrade the performance of a learning model, and bypass filtering rules or may find vulnerabilities with the information available [4].

J. Goodfellow et. al. performed experiments to find a hypothesis function “*f*” which is used to map events for the output produced using logistic regression. In this, the system is designed to take the input values in a binary (i.e. 1 or 0). In a supervised learning approach where logistic regression is used as the regressor, here regression approach will map all the points for function ‘*f*’ by utilizing the well-defined labels and features provided by the supervised learning environment. Consider an Intrusion Detection System as an example where the function value ‘*f*’ represents the points in the systems which would be mapped as an event and are classified as normal or intrusive. (i.e. $f = \text{either normal or intrusive}$). Because of this generalization, the model loses flexibility and cannot provide an elaborative result. Therefore, the system is not able to handle complex situations or parameters, and efforts for amending this lack of flexibility usually leads to overfitting of machine model resulting in a decrement in quality classification made by the machine learning model [5] [24].

M. Barreno et. al. adopted the causative attack model to manipulate a naïve learning algorithm. The model simply yields an optimal policy for the adversary and a bound of effort is required to achieve the adversary’s objective. Resulting bound is extended by using outlier’s detection technique. Outliers detection technique is used to find anomalous data and is a widely used paradigm in fault detection, virus detection, and other intrusion detection. In this technique, the model selects a small region that contains some fixed percentage of the observed data, which is known as support of the data’s distribution. The outlier detector classifies points inside the support as normal and those outside as anomalous. This

technique is often used in a situation where anomalous data is low in quantity or has minimal growth rate. The support region is expanded by retraining the outlier with new points. M. Barreno et. al. found that using hyper dimension outlier’s trajectory for expansion could inject malicious points where the outlier would move next. Therefore, a dynamic machine learning model presents more opportunity for the attackers, as it is possible to skew the learning process using adversarial inputs [4].

L. Huang et. al. on adversarial machine learning provides an accurate description of the adversary’s control over the features by discussing domain limitations that are set upon the adversary due to the application domain itself. These limits can include how an adversary interacts with the application and what kind of data is realistically modifiable by the adversary. Contrasting feature spaces is the second limitation imposed on the adversary by the space of features used by the learning algorithm. In many learning algorithms, data is represented in a feature space in which each feature captures a relevant aspect of a data point for the learning task at hand. Another application-specific aspect of the threat discussed in the paper is contrasting data distribution which not only impacts the performance of the learning algorithm but also its vulnerabilities. The data’s distribution may contain properties which can conflict with the learner’s assumption [6] [10].

IV. METHODOLOGY

This section presents the methodology of introducing the malware classification model to an AML environment using the Black-Box attack. This would enable complete access over the functionality and response of the model when injected with an adversarial input. The methodology used is divided into phases as follows:

1. Requirements Phase

This phase defines all the resources utilized to create an AML model which would perform the Black-Box attack on the malware classification model. The dataset consists of the legitimate and malicious files obtained from VirusShare repository. The programming language here is Python, to develop the ML model, as it provides libraries and tools to assist in the creation of ML model and sufficient tools to assess vulnerability for the same. Table-1 illustrates all the resources used and includes a brief description of the resources.

VirusShare and VirusTotal are an invitation-only community, before getting access to their vast repository of dataset and report on malicious files, a request to join their community should be sent to the administrator for both communities respectively. Because the datasets contain highly volatile and malicious file. Access to this dataset is only granted for research purpose and to run security vulnerability assessments. The approval process takes from 4-7 days in average in case of VirusShare and 6-8 days for VirusTotal.

TABLE I RESOURCES USED DURING THE METHODOLOGY

Resource	Description
Dataset	Corpus of files collected, both “malicious” and “legitimate” for training the machine model. The dataset used is downloaded from the website “VirusShare”.
VMware	The virtual environment is used for security concern (as we are dealing with malware infected files).
Python	The scripts are written using python as the base language.
Features	Feature selection from the files was required to train the machine.
Jupyter notebook	“Jupyter” is used as the platform for executing the python scripts.
Algorithms	The algorithms used are: Decision Tree [17], Random Forest [13] [14], Ada boost, Gradient Boosting, SVM, Linear Regression [15].
VirusShare	Source of the datasets of malicious and legitimate files.

2. Design Phase

Figure-1 depicts the environment designed for the vulnerability assessment of ML-based malware classification models.

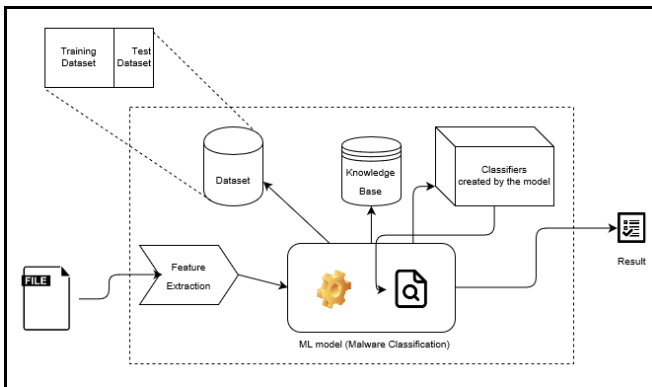


Figure 1 Environment designed for vulnerability assessment

The modules used in figure-1 are listed and described as follows.

- *Dataset*: The corpus collected from the website VirusTotal and VirusShare, contains malicious files and legitimate files.
- *Training Set*: The fragment of the dataset which is used to train the machine model for classification of malware using algorithms mentioned in table 1.
- *Test Set*: Remaining fragment of the dataset which is used to test the knowledge base developed by the classification model.
- *Feature Extraction*: Feature extractor will extract the features from the file, which would enable the model to compare it with the classifiers built by the machine model using the algorithms. The features extracted are mentioned in Table II, and a detail of all the features are mentioned in Table III [30].
- *Classifiers*: The Classifiers are the stored classifications compiled by the model using the best-suited algorithm.
- *File*: The file introduced to the environment which is examined by the malware classification model to determine whether the file is malicious or not.
- *Knowledgebase*: The correlations created or discovered by the ML algorithms using the features from the dataset [16].
- *Result*: The malware classification model, classifying the examined file as a malicious or legitimate file.

3. Implementation Phase

Figure-2 represents all the possible attack areas (i.e. possible points of exploits which can be targeted) present in the environment designed. The modules “dataset”, “knowledgebase”, and “classifiers” are vulnerable to attacks. In this research, the poison attack is conducted on the module “classifiers”.

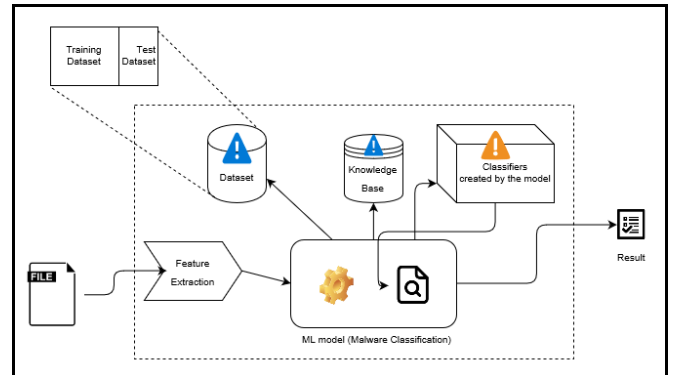


Figure 2 Possible areas in the environment designed vulnerable to attacks

AML's Black-Box attack approach is adopted in this paper. A malware classification model is created which provides complete control over the environment, from the feature engineering for the dataset to the algorithms used for classification. Following are all of the parameters required to develop the machine model:

- *Feature engineering/selection*
The features (unique parameters for an object) are required for a machine learning model to create associations between the objects or elements provided to it. The dataset consists of files belonging to windows platform (operating system), therefore the features selected here are the "Operational Headers" and "System Information" on a file which includes but not limited to "Size of File", "MajorLinkerVersion", "BaseofCode", "AddressOfEntryPoint" [11] [12].
- *Cross-Validation*
Dataset is split into a training set and a test set. The ratio of the split plays a crucial part in creating an effective and accurate machine model, as it affects the ability of machine model to create classes (classification). Here, the dataset is split into the ratio of eighty to twenty (80% training set and 20% test set).
- *Feature Extraction*
It is necessary for an ML model to extract the features which it requires to analyze an object, as "Operation Headers" and "System Information" are being used as features. The model should be able to extract that information from a file. Feature extraction enables the ML model to extract the features from the file.
- *Algorithms*
Algorithms are imported from the "scikit" and "SciPy" Python libraries. The algorithms are used to train the machine model using the features selected. All algorithms are used to create classes from the training sets.
- *Selecting Algorithm*
The results from all the algorithms are stored in the variable "results" (different from the result in the figure-1). The algorithm with the highest result is stored as a variable named "winner" and classifiers are created by the algorithm selected.
- *Result*
Provides the information about the file injected into the system, whether it is legitimate or not.
- *Poison attacks*
In poison attack, the knowledge base is injected with adversarial inputs to create perturbations [26].

- *Decision boundary*
It is a plot or a graph created by the machine model using its algorithm on the dataset and within the boundary would reside all the objects which are deemed viable for the machine model [17].

4. Testing Phase

All the tests and simulations are conducted in a virtual environment using "VMware" as the files dealt with are malicious. During the initial simulation to test the stability of the environment, the dataset is stored in a Comma-separated value (CSV) file and the virtual environment would crash or freeze. The issue was resolved by dedicating more memory to the virtual environment. Figure-8 illustrates that the datasets are loaded successfully with the features intact.

The environment deploys amalgamation of algorithms which are used in this classification model. Parameters for each algorithm are entered in accordance with the requirement of the algorithm (For example, depth for the decision tree, estimators for random forest and so on).

Figure-9 (in Appendix-B) illustrates the results for the different algorithms, and the scores for the algorithms are stored in a variable known as "algo" which will be used to determine the algorithm with the highest result and is used for classification. Table-VI (in Appendix-A) illustrates the total number of legitimate and malicious files present in the dataset. Figure-10 shows the features used for classification by the algorithm which is a part of preprocessing of the dataset. The algorithm with the highest score is used to create the classifiers and is stored in the knowledgebase. Table-VII shows the false positive rate and false negative rate obtained by the current classifiers. This is known as the confusion matrix of an ML model. And this confusion matrix is utilized as the benchmark.

The FAIL model is used as a framework for this adversarial attack on the malware classification model. The environment provides the attacker with sufficient control and information in this scenario to modify the subsets. Implementation of the FAIL model dimensions:

- Features
- Algorithms
- Instances
- Leverages

The first dimension is the "Feature". *What features could be kept a secret? Is access to the exact feature value possible?* These are the questions presented to implement the first dimension. Features kept secret are the pre-processed features extracted from the model for clustering and classification of a dataset. There are 15 features which are kept secret by the classification model and yes, access to exact feature values is available. These values can be modified by the attacker.

The second dimension is the Algorithm. *Is the algorithm class known? Is the training algorithm secret? Are the classifier parameters secret?* The environment is a Black-Box attack. Therefore, the algorithm class is known to the attacker. The training algorithm is not a secret in this environment. Therefore, fulfilling the pre-requisite for the FAIL model is that the attackers need to the algorithm. The answer to the third question in this dimension is, “no, the classifiers generated by the algorithms are not secret”. A collection of algorithms is used in this environment, some of the examples are decision tree [17], random forest [13] [14], ada boost, gradient boosting, SVM, and linear regression [15].

The *instance* is the third dimension of FAIL model and questions related to the third dimension are, “*Is the entire training set known? Is the training set partially known? Are the instances known to the attacker sufficient to train a robust classifier? Yes, the entire training set is known and there is sufficient information about the instances available to create robust classifiers*”.

The last dimension of FAIL model is about leverage, *which features are modifiable by the adversary? The adversary has access and capable of modifying all the features which are selected by the classification model as shown in figure-8*. Leverage dimension is all about introducing adversarial examples into the classification features which are intentionally corrupted/polluted to introduce noise which would decrement the quality of classification done by the model. Figure-3 and table-VII illustrate the confusion matrix before the poison was injected into the classifiers.

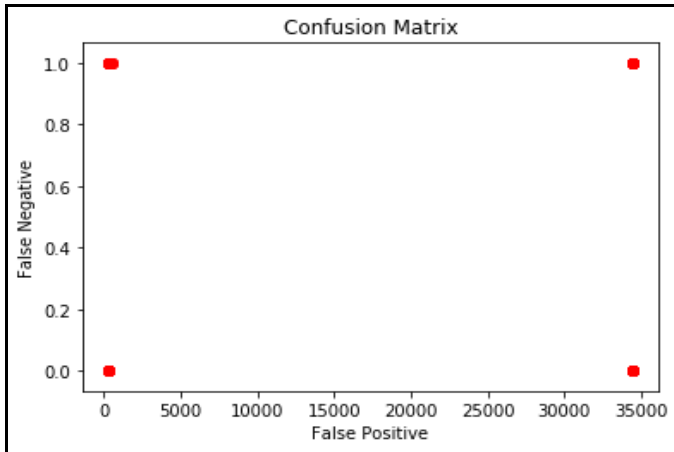


Figure 3 Confusion matrix (pre-poisoning)

V. RESULTS AND DISCUSSION

This section includes the results obtained from injecting adversarial inputs (i.e. poison attacks) to different feature sets. The following figure-4 to figure-8 demonstrate that the injection of adversarial examples into the classifiers of a malware classification model introduces perturbations in the results. The perturbations cause an increment of false negative

and false positive in the system. The value of false positive and false negative depends upon the attack vector (i.e. classifiers), for example, *DLCharacteristics*, *ResourceMaxEntropy*, *ImageBase*, *SectionsNB*.

Figure-4 illustrates the result of poisoning the feature “DLcharacteristics” which increases the rate of false positives to 13.78% and false negatives to 11.57%.

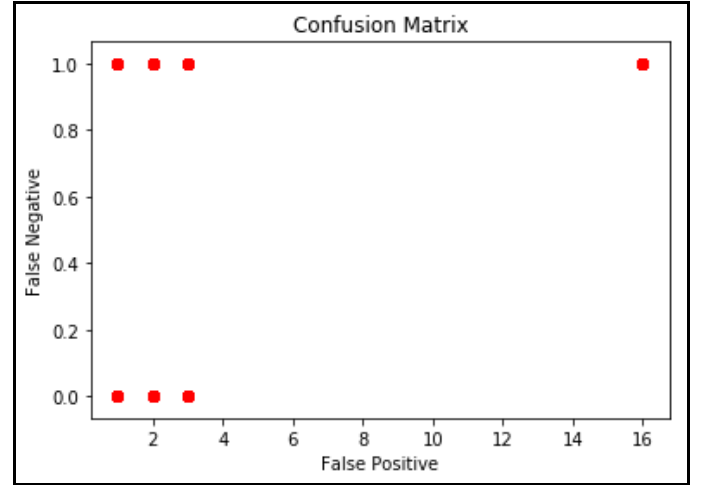


Figure 4 Poisoning of DLcharacteristics

Figure-5 illustrates the result of poisoning the feature “ExportNb” and which increases the rate of false positives to 8.58% and false negatives to 21.53%.

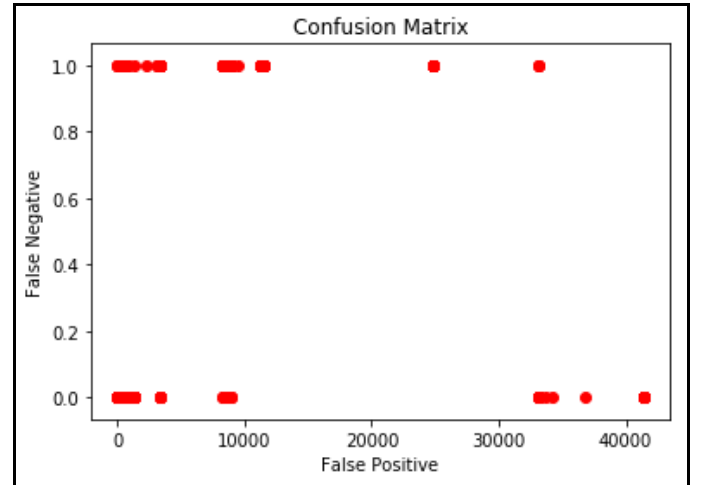


Figure 5 Poisoning feature “ExportNb”

Figure-6 illustrates the result of poisoning the feature “ResourceMaxEntropy” which increases the rate of false positives to 37.57% and false negatives to 31.53%.

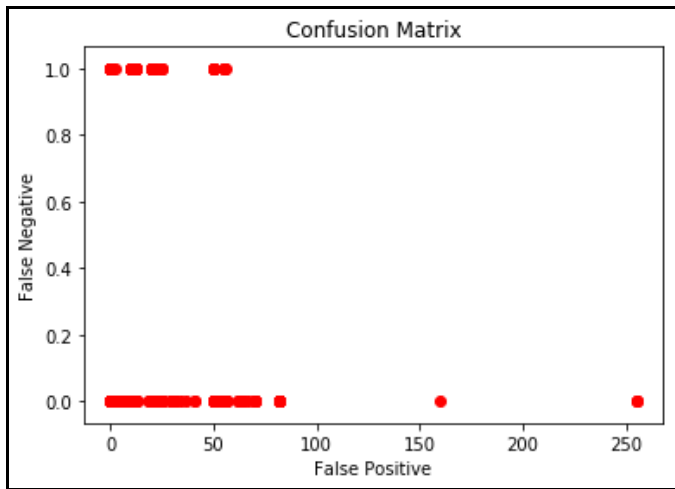


Figure 6 Poisoning feature “ResourceMaxEntropy”

Figure-7 illustrates the result of poisoning the feature “ResourceMaxEntropy” which increases the rate of false positives to 75.57% and false negatives to 53.53%.

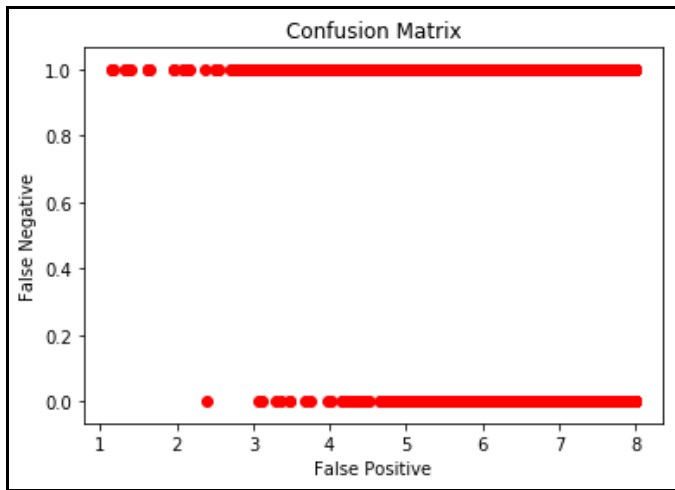


Figure 7: Poisoning feature “ImageBase”

Figure-13 and figure-15 (Appendix-C), depicts the impact of injecting adversarial examples as the classification model misclassifies the file “skype.exe”, and “BCN12ui4923” as malicious and legitimate respectively. This result proves the inherent security vulnerability which exists in ML model. So, if an attacker is able to attack the model and is able to detect or create these vulnerabilities, a malicious code can be developed and deployed to any machine model designed to perform a similar task (by the property of transferability, as discussed in related work).

In the testing phase, the simulations provided an insight into the inputs (adversarial examples) injected into the classifiers. If these inputs are not moderated and injected on a large scale (indiscriminative attack), then it would introduce high-levels

of noise into the model which would result in the classification model not being able to create any clusters or classes. Therefore, the system would be completely incompetent. To prevent such a predicament, the adversarial examples should be moderated.

VI. CONCLUSION AND FUTURE WORK

Injecting adversarial examples into classifiers creates distortion in the classification model by focusing the knowledge base itself. The vulnerability of ML systems against evasion and poisoning attacks which are incorporated using AML leads to an arms race, where the defenses of systems seem adequate, but are quickly circumvented by new attacks and if the impact of the attack is quantifiable or detectable, then the defenses are updated in accordance to the attack. This places the attacker with constraints weather to craft an attack which could have a high impact on the system but remains undetected (i.e. false negative for a malicious software), or the attacker can purposely reveal the attack areas to map all the defense boundary implemented to prevent it. Application of game theory can be done to identify how much an attacker can reveal and similarly how much a system can accept as lose while generating an efficient and accurate classification model.

The security model approaches for ML (reactive security model, and proactive security model). The *reactive security model* would ensure countermeasure after the system detects intrusion or impact is profoundly felt by the model, whereas proactive model would devise methods to simulate the attack on the model and comprehend the impact and effects of the adversarial attack and develop a mitigation strategy. (This paper can be viewed as an effort of a proactive approach to security models in machine learning). The work can be further expanded in the following areas:

- Developing a framework to analyze and adopt which approach is best suited for the machine model (i.e. reactive approach or proactive approach).
- Implementing the mitigation strategies as discussed in the “Results” section and evaluate the strength of the model.
- Develop a continuous adversarial attack and impact analysis model as mentioned in the “Results and discussion” section.
- Implementation of game theory in both security model approaches (*proactive and reactive*).

ACKNOWLEDGMENT

I would like to thank God and Concordia University of Edmonton for the support and guidance, which made this research a reality. I would also like to thank my parents for their unwavering support and encouragement to pursue excellence. Finally, I would like to thank “VirusShare Community” for providing access to their vast dataset of

malicious files which played a crucial role in the completion of this paper.

REFERENCES

- [1] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in Machine Learning: from phenomena to Black-Box Attacks using Adversarial samples", 2016.
- [2] M. Barreno, B. Nelson, A. D. Joseph, "The Security of Machine Learning", *Machine Learning*, 81(2), 2010.
- [3] A. Kurakin, J. Goodfellow and S. Bengio, "Adversarial Machine Learning at Scale", arXiv preprint arXiv:1611.01236, 2016.
- [4] M. Barreno, B. Nelson, R. I. Sears, A. D. Joseph, and J. D. Tygar, "Can Machine Learning be Secure?", *Proceedings of the 2006 ACM Symposium on Information, computer, and communications security. ACM*, 2006.
- [5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing Adversarial Examples", arXiv preprint arXiv:1412.6572, 2014.
- [6] N. Papernot, N. Carlini, I. Goodfellow, R. Feinman, F. Faghri, A. Matyasko, K. Hambardzumyan, Y. Juang, A. Kurakin, R. Sheatsley, A. Garg, Yen-Chen Lin, P. Hendricks and P. McDaniel, "Clever Hans V2.0.0: An Adversarial Machine Learning Library", arXiv preprint arXiv:1610.00768, 2016.
- [7] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. Berkay Celik, A. Swami, "The Limitations of Deep Learning in Adversarial settings", *Security and Privacy (EuroS&P)*, 2016 IEEE European Symposium on. IEEE, 2016.
- [8] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, E. Vazquez, "Anomaly-based Network intrusion detection Techniques, systems and challenges", *computers & security* 28.1-2, 2009.
- [9] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, T. Dumitras, "When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks", arXiv preprint arXiv:1803.06975, 2018.
- [10] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, Bo Li, "Automated Poisoning Attacks and Defenses in Malware Detection Systems: An Adversarial Machine Learning Approach", *computers & security* 73, 2018.
- [11] S. Scott, S. Matwin, "Feature Engineering For Text Classification", *ICML. Vol. 99.*, 1999.
- [12] CR Turner, AL Wolf, A Fuggetta, L Lavazza "A Conceptual Basis For Feature Engineering", *Journal of Systems and Software* 49.1, 1999.
- [13] MM Adankon, M Cheriet "Support Vector Machine", *Encyclopedia of biometrics. Springer, Boston, MA*, 2009.
- [14] S. Suthaharan, "Machine Learning Models and Algorithms for Big Data Classification", *Integr. Ser. Inf. Syst* 36, 2016.
- [15] A Liaw, M Wiener, "Classification And Regression By Randomforest", *R news* 2.3, 2002.
- [16] KP Murphy, "Naive Bayes classifiers", *University of British Columbia* 18, 2006.
- [17] C Lee, DA Landgrebe, "Feature Extraction Based on Decision Boundaries", *IEEE Transactions on Pattern Analysis & Machine Intelligence* 4, 1993.
- [18] SB Kotsiantis, I Zaharakis, "Supervised Machine Learning: A Review Of Classification Techniques", *Emerging artificial intelligence applications in computer engineering* 160, 2007.
- [19] R Gentleman, VJ, "Unsupervised Machine Learning", *Bioconductor Case Studies. Springer, New York, NY*, 2008.
- [20] RS Sutton, AG Barto, "Reinforcement Learning: An Introduction", *MIT Press*, 1998.
- [21] CJCH Watkins, P Dayan, "Q-learning", *Machine learning* 8.3-4, 1992.
- [22] H Van Hasselt, A Guez, D Silver, "The Predictron: End-To-End Learning And Planning", arXiv preprint arXiv:1612.08810, 2016.
- [23] M Sun, T Chen - US Patent App. 12/411,916, 2010 - Google Patents.
- [24] L Huang, AD Joseph, B Nelson, "Adversarial Machine Learning", *Proceedings of the 4th ACM workshop on Security and artificial intelligence. ACM*, 2011.
- [25] B Biggio, I Corona, D Maiorca, B Nelson et al, "Evasion Attacks Against Machine Learning At Test Time" *6th European Machine Learning and Data Mining Conference*, 2013.
- [26] B Biggio, B Nelson, P Laskov, "Poisoning Attacks Against Support Vector Machines" *Int'l Conference on Machine Learning*, 2012.
- [27] H. Kwon, H. Yoon, D. Choi, "Zero-Day Evasion Attack Analysis on Race between Attack and Defense" *ASIACCS '18 Proceedings of 2018 on Asia Conference on Computer and Communications Security*, 2018.
- [28] VirusShare.com, "VirusShare.com - Because Sharing is Caring," 2017 [Online]. Available: <https://virusshare.com/about.4n6>. [Accessed 25 11 2017].
- [29] Virus Total, [Online]. Available: <https://www.virustotal.com/en/about/>.
- [30] I. Guyon., A. Elisseeff, "An Introduction to Feature Extraction" *The Journal of Machine Learning Research*, 2003.
- [31] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, T. Dumitras, "When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks", 2018.
- [32] B. Biggio, F. Roli, "Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning", *University of Cagliari*, 2018.
- [33] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, 1959.

APPENDIX A.

TABLE II. LIST OF ALL AVAILABLE FEATURES BEFORE PRE-PROCESSING

Features List			
Name	md5	SizeOfOptionalHeader	Characteristics
MajorLinkerVersion	MinorLinkerVersion	SizeOfInitializedData	SizeOfUninitializedData
AddressOfEntryPoint	BaseOfCode	ImageBase	SectionAlignment
FileAlignment	MajorOperatingSystemVersion	MajorSubsystemVersion	MinorSubsystemVersion
SizeOfImage	MajorImageVersion	SizeOfHeaders	Checksum
Subsystem	DllCharacteristics	SizeOfStackCommit	SizeOfHeapReserve
SizeOfHeapCommit	LoaderFlags	SectionsNb	SectionsMeanEntropy
SectionsMinEntropy	SectionsMaxEntropy	SectionsMinRawsize	SectionMaxRawsize
SectionsMeanVirtualsize	SectionsMinVirtualsize	ImportsNbDLL	ImportsNb
ImportsNbOrdinal	ExportNb	ResourcesMeanEntropy	ResourcesMinEntropy
ResourcesMaxEntropy	ResourcesMeanSize	ResourcesMaxSize	LoadConfigurationSize
VersionInformationSize	legitimate	MinorOperatingSystemVersion	MinorImageVersion
Machine	BaseOfData	NumberOfRvaAndSizes	SectionsMeanRawsize
SizeOfCode	SizeOfStackReserve	SectionMaxVirtualsize	ResourcesNb
ResourcesMinSize			

- Table-II contains a list of all the features obtained from a file after the feature extraction process. There are a total of 57 listed features starting from feature “Name” to feature “ResourcesMinSize”.
- Features are Operational Headers in Windows Operating system.
- Features in table II are going to be used for pre-processing and select features for the machine model.
- Table IV depicts features selected from the pre-processing phase.
- Table III shows the feature values for file “Skype.exe” using feature extraction. A total number of features extracted from the file “Skype.exe” is 55. All the features underlined are the selected features of the ML model used to create correlation and classifiers for an effective predictive model.

TABLE III. FEATURE EXTRACTION OF SKYPE.EXE

S.no	Feature	Value	S.no	Feature	Value
1	Name	Skype.exe	31	<u>MinorLinkerVersion</u>	<u>25</u>
2	md5	5cbc122dd419711cae36d8ef88a5bc09	32	<u>SizeOfCode</u>	<u>17645568</u>
3	<u>Machine</u>	<u>332</u>	33	SizeOfInitializedData	16098816
4	SizeOfOptionalHeader	224	34	SizeOfUninitializedData	0
5	<u>Characteristics</u>	<u>33679</u>	35	AddressOfEntryPoint	1805886
6	BaseOfCode	4096	36	MinorSubsystemVersion	0
7	BaseOfData	17649664	37	SizeOfImage	33853440
8	<u>ImageBase</u>	<u>4194304</u>	38	<u>SizeOfOptionalHeaders</u>	<u>1024</u>
9	SectionAlignment	4096	39	Checksum	19578734
10	FileAlignment	512	40	Subsystem	2
11	<u>MajorOperatingSystemVersion</u>	<u>5</u>	41	<u>DllCharacteristics</u>	<u>0</u>
12	MinorOperatingSystemVersion	0	42	<u>SizeOfStackReserve</u>	<u>1048576</u>
13	MajorImageVersion	0	43	SizeOfStackCommit	16384
14	MinorImageVersion	0	44	SizeOfHeapReserve	1048576
15	MajorSubsystemVersion	5	45	SizeOfHeapCommit	4096
16	LoaderFlags	0	46	<u>SectionsMaxEntropy</u>	<u>7.999983</u>
17	NumberOfRvaAndSizes	16	47	SectionsMeanRawsize	3910246
18	<u>SectionsNb</u>	<u>5</u>	48	SectionsMinRawsize	0
19	SectionsMeanEntropy	3.873043	49	SectionMaxRawsize	16517120
20	SectionsMinEntropy	0	50	SectionsMeanVirtualsize	6769588
21	SectionsMinVirtualsize	35460	51	<u>ExportNb</u>	<u>0</u>
22	SectionMaxVirtualsize	17645568	52	ResourcesNb	843
23	ImportsNbDLL	1	53	ResourcesMeanEntropy	7.010084
24	ImportsNb	2	54	ResourcesMinEntropy	1.569342
25	ImportsNbOrdinal	0	55	<u>ResourcesMaxEntropy</u>	<u>7.99254</u>
26	ResourcesMeanSize	3548.04			
27	<u>ResourcesMinSize</u>	<u>16</u>			
28	ResourcesMaxSize	270376			
29	LoadConfigurationSize	0			
30	<u>VersionInformationSize</u>	<u>17</u>			

- Table IV illustrates feature selected for classification. There are a total of 15 features selected.

TABLE IV. FEATURE SELECTION

DLLCharacteristics	Machine	ImageBase	SizeOfOptionalHeader	SizeOfStackReserve
SectionMaxEntropy	Characteristics	ExportNb	ResourceMaxEntropy	VersionInformationSize
MajorOperatingSystemVersion	SectionsNb	ResourceMinSize	MinorLinkerVersion	MajorSubsystemVersion

TABLE V. FEATURE SCORE BY EXTRA TREE CLASSIFIERS

S.no	Feature Name	Feature Score
1	DllCharacteristics	0.154046
2	Characteristics	0.045312
3	Machine	0.129020
4	SectionsMaxEntropy	0.052787
5	MajorSubsystemVersion	0.021180
6	ResourcesMinSize	0.025336
7	ResourcesMaxEntropy	0.036192
8	ImageBase	0.099113
9	VersionInformationSize	0.034737
10	SizeOfOptionalHeader	0.091030
11	SizeOfStackReserve	0.053154
12	Subsystem	0.046852
13	MajorOperatingSystemVersion	0.033150
14	SectionNb	0.030250

TABLE VI. : CONTENTS OF DATASET DIVIDED ACCORDING TO THEIR NATURE

Nature of File	Number of Files
Legitimate	41323
Malicious	96724

- Table-VI depicts the total number of files in the dataset collected from ViruShare, the dataset is grouped into a legitimate and malicious file.

TABLE VII. : CONFUSION MATRIX BEFORE ATTACK

Parameter	Score (Percentage)
False Positive rate	0.035811
False Negative rate	0.015376

- Table-VII shows the confusion matrix score for machine model, this score can be used as the benchmark to show the impact of adversarial input on the model.

TABLE VIII. : CHANGES IN CONFUSION MATRIX AFTER RESOURCEMAXENTROPY FEATURE IS POISONED

Parameter	Score (Percentage)
False Positive rate	0.099251
False Negative rate	0.147618

- Table-VIII depicts the increase in false positive and false negative, once the feature “ResourceMaxEntropy” was poisoned using adversarial examples.

APPENDIX B.

Testing Phase outputs

In [3]: dataset.head()

Out[3]:

	Name	md5	Machine	SizeOfOptionalHeader	Cha
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	333
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	333
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258

5 rows x 57 columns

Figure 8 Testing deployment of the dataset

- Figure-8 illustrates the successful deployment of the dataset into the environment, with all the features intact. Md5 values make sure that the dataset was not altered during the loading process.

```
In [12]: model = { "DecisionTree":tree.DecisionTreeClassifier(max_depth=10),
                  "RandomForest":ek.RandomForestClassifier(n_estimators=50),
                  "Adaboost":ek.AdaBoostClassifier(n_estimators=50),
                  "GradientBoosting":ek.GradientBoostingClassifier(n_estimators=50),
                  "GNB":GaussianNB(),
                  "LinearRegression":LinearRegression()
                }
```

Figure 9 Collection of Algorithms

- Figure-9 illustrates how the machine learning model incorporates an algorithm model which consists of multiple algorithms, from these algorithms the highest scoring (in terms of classification) is selected. This process helps to find the best-suited algorithm according to the dataset introduced.

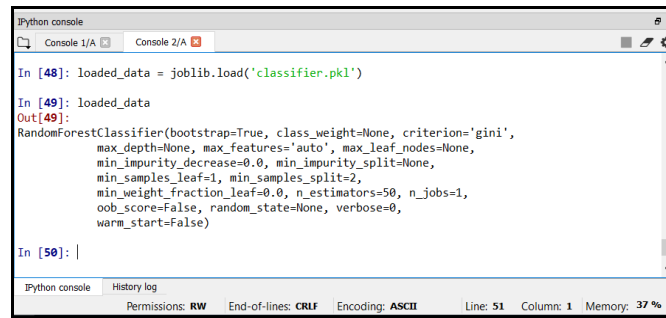
```
Console 1/A Console 2/A
In [22]: index = numpy.argsort(extratrees.feature_importances_)[-1][:nbfeatures]

In [23]: for f in range(nbfeatures):
...:     print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]],
...:     extratrees.feature_importances_[index[f]]))
...:     features.append(dataset.columns[2+f])
1. feature DllCharacteristics (0.154046)
2. feature Machine (0.129020)
3. feature ImageBase (0.099113)
4. feature SizeOfOptionalHeader (0.091030)
5. feature SizeOfStackReserve (0.053154)
6. feature SectionsMaxEntropy (0.052787)
7. feature Subsystem (0.046852)
8. feature Characteristics (0.045312)
9. feature ResourcesMaxEntropy (0.036192)
10. feature VersionInformationSize (0.034737)
11. feature MajorOperatingSystemVersion (0.033150)
12. feature SectionsNb (0.030250)
13. feature ResourcesMinSize (0.025336)
14. feature MinorLinkerVersion (0.022170)
15. feature MajorSubsystemVersion (0.021180)

In [24]:
```

Figure 10 Feature score by ExtraTreeClassifier

- Figure-10 illustrates the features created by the classifier, the score is given to the classifier by the model and using the concept of wrapping the model will find an association between these features to create a good prediction model.



```
Python console
Console 1/A Console 2/A

In [48]: loaded_data = joblib.load('classifier.pkl')

In [49]: loaded_data
Out[49]:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1,
                        oob_score=False, random_state=None, verbose=0,
                        warm_start=False)

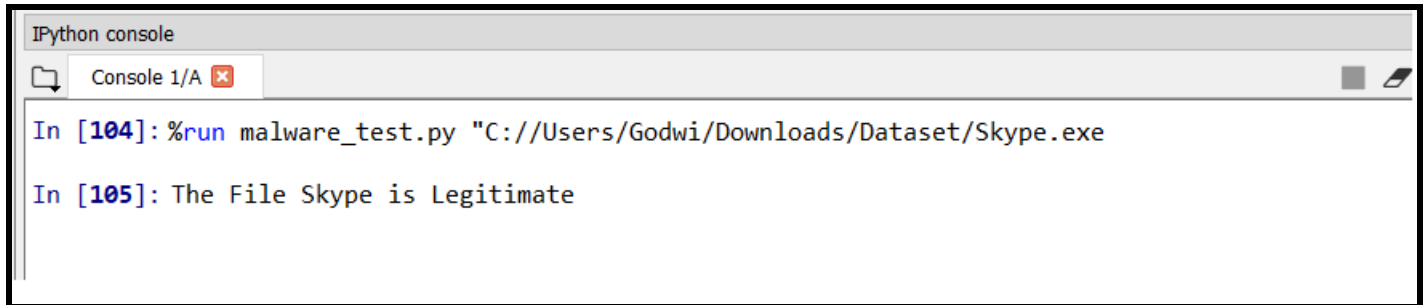
In [50]: |
```

Figure 11 *The file Classifier.pkl is created by loading result from Algorithm*

- Figure-11 illustrates the machine model using classifiers from the pickle file called “classifier.pkl”, this classifier file is used to make the prediction model by implementing the RandomForest Algorithm

APPENDIX C.

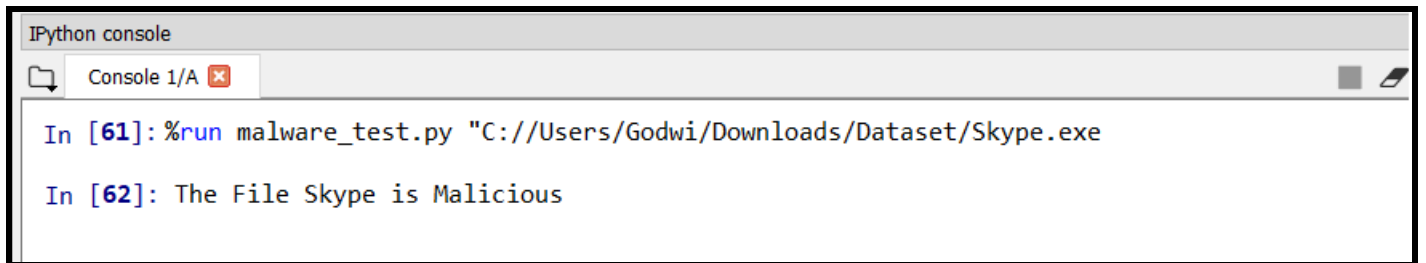
Malware Classification Model's Outputs



```
IPython console
Console 1/A x
In [104]: %run malware_test.py "C://Users/Godwi/Downloads/Dataset/Skype.exe"
In [105]: The File Skype is Legitimate
```

Figure 12: Before-Dataset Poison

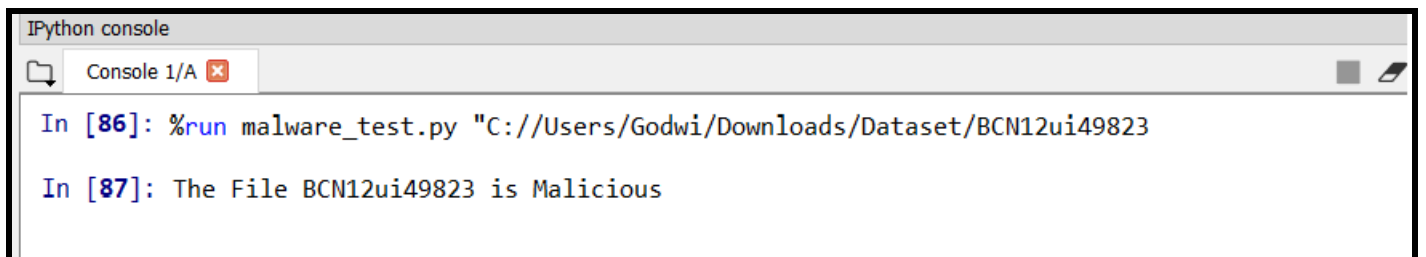
- **Figure-12** illustrates the result of malware classification model when provided with “skype.exe” file, this is a pre-attack result (i.e. the classifiers are not poisoned yet). Malware classification model was successfully able to classify skype as a legitimate file.



```
IPython console
Console 1/A x
In [61]: %run malware_test.py "C://Users/Godwi/Downloads/Dataset/Skype.exe"
In [62]: The File Skype is Malicious
```

Figure 13: After-Dataset Poison

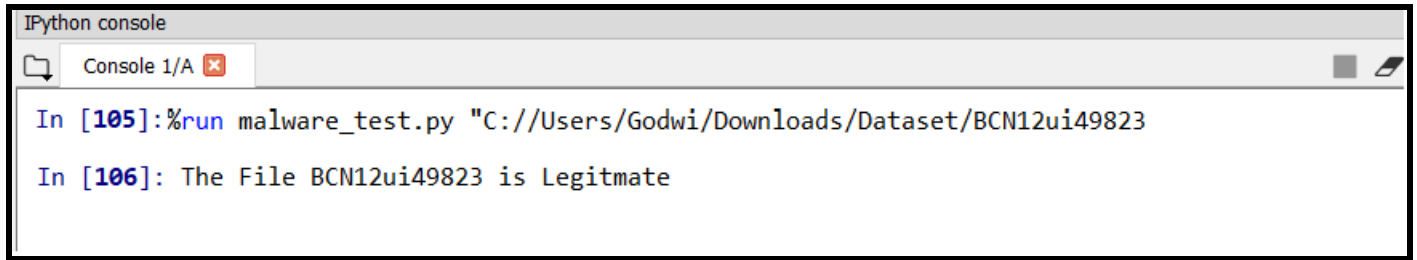
- **Figure-13** illustrates the result of malware classification model provided with the “skype.exe” file, this is a post-attack (i.e. the classifiers are poisoned). Malware classification was not successful in classifying the file and the file “skype.exe” is misclassified as malicious (which is a false negative).



```
IPython console
Console 1/A x
In [86]: %run malware_test.py "C://Users/Godwi/Downloads/Dataset/BCN12ui49823"
In [87]: The File BCN12ui49823 is Malicious
```

Figure 14: Before Dataset Poison

- **Figure-14** illustrates the result of malware classification model when provided with “BCN12ui4923” file, this is a pre-attack result (i.e. the classifiers are not poisoned yet). Malware classification model was successfully able to classify “BCN12ui4923” as a malicious file.

An IPython console window with a title bar 'IPython console' and a tab 'Console 1/A'. The console shows two lines of text: 'In [105]: %run malware_test.py "C://Users/Godwi/Downloads/Dataset/BCN12ui49823"' and 'In [106]: The File BCN12ui49823 is Legitmate'.

```
IPython console
Console 1/A x
In [105]: %run malware_test.py "C://Users/Godwi/Downloads/Dataset/BCN12ui49823"
In [106]: The File BCN12ui49823 is Legitmate
```

Figure 15: *After Dataset Poison*

- **Figure-15** illustrates the result of malware classification model when provided with “BCN12ui4923” file, this is a post-attack result (i.e. the classifiers are poisoned). Malware classification model was not successfully able to classify the file “BCN12ui4923” and misclassifies it as a legitimate file (which is a false positive).