University of Piraeus
Department of Digital Systems
MSc. Digital Systems Security

Master Thesis

Machine Learning in the field of Information Security

Kalachanis Athanasios (MTE1510)

Supervisor
Dadoyan Christoforos

Piraeus,2018

Acknowledgements

Here, I would like to take this chance to thank my family for supporting me throughout my whole studies by any means possible. Also, I would like to thank all the friends that I made throughout my studies, that we shared not only the same classrooms but also the same life.

Lastly, I would like to express my sincere gratitude to my supervisor for his patience and continuous support. His enthusiasm and in-depth knowledge made me eager to dive more in depth and expand my knowledges on the fascinating field of information security.

Abstract

The purpose of this thesis is to analyze existing machine learning application in the information security field and demonstrate a machine learning malware classifier. At first, we'll make a brief introduction into data science, malware, machine learning and adversarial machine learning. Moreover, we will concentrate on applications of machine learning systems in the cyber security field and how an attacker can evade such systems and impact the integrity, availability and confidentiality by exploiting the classifiers vulnerabilities. Finally, we present a custom malware classifier as a proof of concept executable files.

Related Work

There have been several types of studies carried out in the field of malware detection and machine learning mostly aiming to provide a unified solution on detection techniques or even different ways for an adversary to evade those methods.

On their research, Baldangombo, Jambaljav and Horng [32] proposed several feature extraction methods based on PE headers, API and DLL function calls using as classifier algorithms Naïve Bayes, J48 Decision Trees as well as Support Vector Machines. Their result showed that the higher overall accuracy scored the J48 algorithm with 99% detection rate.

Weilin Xu, Yanjun Qi, and David Evans, on their paper Automatically Evading Classifiers [33], presented a general approach to search for evasive variants for PDF malware classifiers. Their results suggest a general method for evaluating classifiers used in security applications and raise serious doubts about the effectiveness of classifiers based on superficial features in the presence of adversaries.

On “Automatic malware classification and new malware detection using machine learning” [34] Liu Liu, Bao-sheng Wang, Bo Yu and Qiu-xi Zhong proposed a machine learning based malware analysis system which uses the Share Nearest Neighbor clustering algorithm in order to discover new malware families. Their results showed that it was able to detect 86.7% of the new malware.

More on the Adversarial Machine Learning field, Ian Goodfellow on his “Practical Black-Box Attacks against Machine Learning” [34] introduce an attack on a Deep Neural Network which is used for classification with a success rate of 84.24%.

It is quite common for those studies to focus on solving a particular type of problem under specific conditions, e.g. malware type. However, there hasn’t been any methodology on how to implement malware detection along with machine learning techniques.

Contents

1. Introduction

The evolution computing capability of computer systems has led in the last decades to a rapid increase in the volume of data produced, stored and transported. This created the need to analyze and process large volumes and different types of data, something that was not feasible with traditional models and tools. Also, the field of data science was created that addresses this new reality in the data features and basic data categories. This increased amount of data helped also the improvements of Machine Learning algorithms, where until the last decade was not expected to work, nowadays they can even outperform humans. An example of machine learning excellence is the AlphaGo project of Google's DeepMind. The first computer program that ever defeated a professional Go player [30]. Moreover, as the technology advances and the usage of complex algorithms is getting much easier to use, more and more applications are created in order to help us with our day to day problems.

As for the Information Technology industry, a problem that has not yet been solved is the existence of malicious software. Every day more and more new complex malwares are released which aim to harm users and computer systems. The security industry has not been able to completely defend against those threats there is a need for a more "advanced" way of protecting against those-instead of the plain old solutions.

In this Thesis we present how machine learning is able to help the security industry and especially malware classification.

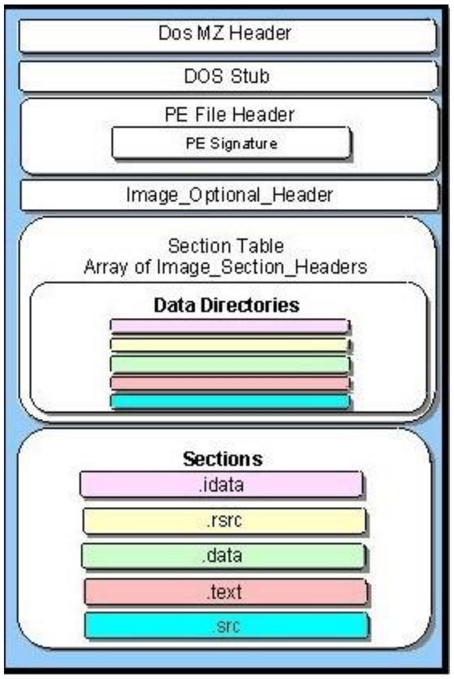
2. Portable Executable (PE)

Most malwares nowadays are in the format of a portable executable (PE) format. Below we mention the main structure and properties of a PE file.

The PE (Portable Executable) file format is used for executable files on 32 and 64- bit Windows platforms. Not only EXE but also, for example, DLL, SYS and SCR files have the PE file structure, which is a modified version of the COFF format [8]. The PE file header consists of a MZ-DOS header and MS-DOS stub, the PE signature, the COFF (PE) file header, and an optional header, the file headers are followed by section headers and sections. Typical PE file layout shown on Figure 1.

2.1 Structure

A Portable Executable (PE) basically contains two sections, which can be subdivided into several sections. One is Header and the other is Section. The diagram below shows a visualized version of the PE.



i PE file format structure [8]

2.1.1. DOS Header

DOS header starts with the first 64 bytes of every PE file. It's there because DOS can recognize it as a valid executable and can run it in the DOS stub mode.

2.1.2 DOS Stub

The DOS stub usually just prints a string, something like the message, “This program cannot be run in DOS mode.”, which is shown when the PE tries to run in DOS mode. It can be a full-blown DOS program.

When building applications on Windows, the linker sends instruction to a binary called winstub.exe to the executable file. This file is kept in the address 0x3c, which is offset to the next PE header section.

2.1.3 PE File Header

Like other executable files, a PE file has a collection of fields that defines what the rest of file looks like. The header contains info such as the location and size of code,

2.1.4 Characteristics

- Signature: It only contains the signature so that it can be easily understandable by windows   loader. The letters P.E. followed by two 0’s tells everything.
- NumberOfSections: This defines the size of the section table, which immediately follow the header.
- SizeOfOptionalHeader: This lies between top of the optional header and the start of the section table. This is the size of the optional header that is required for an executable file. This value should be zero for an object file.
- Characteristics: This is the characteristic fags that indicate an attribute of the object or image file. It has a fag called Image_File_dll, which has the value 0x2000, indicating that the image is a DLL. It has also different fags that are not required for us at this time
- Image_Optional_Header: This optional header contains most of the meaningful information about the image, such as initial stack size, program entry point location, preferred base address, operating system version, section alignment information, and so forth. We can see the information in the snapshot below.

2.1.5 The Section Table

This table immediately follows the optional header. The location of this section of the section table is determined by calculating the location of the first bytes after header. For that, we have to use the size of the optional header. The number of the array members is determined by NumberOfSections field in the file header (IMAGE_FILE_HEADER) structure. The structure is called IMAGE_SECTION_HEADER.

The number of entries in the section table is given by NoOfSectionField in the file header. Each section header has at least 40 bytes of entry. We will discuss some of the important entries below.

- Name: An 8-byte null-padded UTF8 encoding string. This is can be null.
- VirtualSize: The actual size of the section's data in bytes. This may be less than the size of the section on disk.
- SizeOfRawData: The size of section's data in the file on the disk.
- PointerToRawData: This is so useful because it is the offset from the file's beginning to the section's data.
- Characteristics: This fag describes the characteristics of the section.

2.1.6 The PE File Section

This section contains the main content of the file, including code, data, resources and other executable files. Each section has a header and body.

An application in Windows NT typically has nine different predefined sections, such as. text, bss, rdata, data, rsrc, edata, idata, pdata, and. debug. Depending on the application, some of these sections are used, but not all are used.

The Executable Code: In Windows, all code segments reside in a section called .text section or CODE. We know that windows use a page-based virtual system, which means having one large code section that is easier to manage for both the OS and application developer. This also called as entry point and thunk table, which points to IAT. We will discuss the thunk table in IAT.

- The. bss represents the uninitialized data for the application.
- The. rdata represents the read-only data on the file system, such as strings and constants.
- The. rsrc is a resource section, which contains resource information of a module. In many cases it shows icons and images that are part of the file's resources.
- The. edata section contains the export directory for an application or DLL. When present, this section contains information about the names and addresses of exported functions. We will discuss these in greater depth later.
- The. idata section contains various information about imported functions, including the import directory and import address table. We will discuss these in greater depth later.

3. Malware

Malware – short for malicious software – is a program that causes harm to a computer, network or a user. [1]

Nowadays, a new virus is released every 4.2 seconds [2] so it is crucial to be able to group them in categories depending on their behavior. Some of these categories of malware are: ransomware, worms, trojan horses, rootkits, adware, spyware. etc. In this chapter we will make a brief overview on those categories and their characteristics.

3.1 Malware Types

3.1.1 Virus

A virus is a code that recursively replicates a possibly evolved a copy of itself. In other words, it is a computer program that attaches itself to other files or processes. [3]

Most commonly, a virus will be executed only if the victim runs an infected file. Upon execution and therefore successful infection of the system, the virus replicates itself by modifying existing programs and rewriting their source code with its own malicious code.

3.1.2 Worms

Worms are network viruses, primarily replicating on networks Peter Szor. [1]

Typically, worms do not need a host file and execute themselves without the need of user interaction. The main purpose of a worm malware is to use the computer network in order to spread itself by exploiting vulnerabilities that the other computers in the network may have.

3.1.3 Backdoor

A backdoor virus is used to allow access to the attackers remotely by entering the computer system and without being detected running in the background. It is similar to the *Trojan Horse* computer virus since they both expose the system to unwanted remote access.

3.1.4 Trojan Horse

The Trojan Horse computer virus gets its name from an incident in Homer's Iliad where the Greeks sent a big wooden horse to the Trojans with an army of men hidden inside of it and as soon as the Trojans brought the horse into their city, the Greek army came out of the horse at night – surprising the Troyes and destroying the city of Troy, thus ending the war. Same as the tale, a trojan horse is a basically a backdoor installed in a legitimate looking program aiming to provide remote access to the affected machine.

3.1.5 Rootkit

A rootkit is a collection of malicious software used to elevate a low-privileged user's access on a system.

A subcategory of virus is the ware-like viruses. These are malware aimed to gain profit for the attacker.

3.1.6 Spyware

Spyware is a computer virus aimed to spy upon the victim's compromised system and gather information and personal data in order to sell them and make profit.

3.1.7 Adware

Adware is generating profit for the attacker by generating advertisements to the victim and forcing them to visit them. The profit of the attacker derives from how many times a user clicks on the advertisements.

3.1.8 Ransomware

Ransomware are one of the most dangerous viruses. Usually if a victim is infected by a ransomware it means that their data have been encrypted by the malware making them unable to retrieve and the only way to get them is by paying the attacker a ransom.

3.1.9 Polymorphic

A polymorphic virus is a self-encrypted malware designed to avoid detection by antivirus scanners. Upon infection, the polymorphic virus re-encrypts itself and duplicates by creating usable, albeit slightly modified, copies of itself.

3.1.10 Advanced Persistent Threats

Advanced persistent threats are a general term for describing a multi-step, sophisticated, stealthy attack on a network aiming to gain permanent access on the infrastructure. These types of threats are aimed mostly on organizations, or companies rather than a single user.

3.2 Malware Detection & Countermeasures

Over the years, organizations have found various ways of dealing with malware. At first endpoint protection on each computer was enough in order to stop most of the common threats but as the years passed attackers were able to evade them. Below are mentioned the different types of malware detection so far.

3.2.1 Signature-Based Detection

Signature-based detection works by searching for particular sequences of bytes within an object in order to identify exceptionally a particular version of a virus [4].

This is the most straight-forward way of malware detection. This detection method is called static analysis and relies on already known file fingerprints, static strings or file metadata which are known to be malicious. In real world scenarios, when an antivirus scans a file to check if it is malicious or not, the first step it takes is to check it against some predefined rules. For example, first it will check if it is a known file by computing its hash and cross-checking it through a list of known malicious software. If there is a match then the file is flagged as malicious. But this type of detection can only help in the detection of well-known malwares.

The problem with this type of detection is that it is based on signature databases, which need to be updated regularly in order to prevent as much as more viruses as possible. It is not considered a good practice to protect computer systems only with this technique, besides the fact that nowadays most AV products have built huge databases with signatures and are able to detect any already known virus; the amount of malware that get generated every day [2] makes this detection inadequate. Also, due to the nature of malwares in general, it is really easy for an attacker to create variants of the malware and therefore bypass the signature-based detection [5].

3.2.2 Heuristic-Based Detection

In order to overcome signature-based detection's limitations, a heuristic-based detection was developed. This type of detection is also referred as behavior-based detection. How it works is it searches for abnormal activity, keystrokes and suspicious network traffic rather than checking just for predefined signatures. It is considered to be the most reliable way of detecting malware [7], mostly because it can very well deal with unknown/new viruses. The main problem with this type of detection is that it is time-consuming, it generates more false-positives than the signature-based detection [1] and accuracy of detection is based on the implementation. A good method of implementing heuristic-based detections is by utilizing sandboxes where we can safely run the file and monitor its behavior.

3.3 Machine Learning for Malware Detection

As mentioned before, malware detection is mostly based on signatures of already known malwares and it is difficult for them to detect polymorphic viruses that are able to change their signatures or even 0-day malwares. Also, heuristics detection is not enough due to the high number of false positives [1]. So, there is a need for newer detection methods with higher accuracy and more reliable results. A solution is by combining heuristics-based analysis along with machine learning methods for better detection rates.

For example, we can combine these two modules by analyzing the file's data in two phases: pre-execution and post-execution. The first phase can be done by analyzing the file with the "traditional" ways like static analysis, signature-based checks etc. After that, and always under a controlled isolated environment (e.g. sandbox), we run the file and collect data about its behavior. Lastly, according to the data we collected from the file, we can feed them to our machine learning algorithm in order to make a prediction if it is malicious or benign combining them with the results from the previous phases.

It is important that when we rely on approaches where we combine different methods for detections, in our case signature-based, behavioral and machine learning, to properly define the threshold for each phase so that we do not create overhead to the overall analysis.

4. Machine Learning

Here we will give a brief overview of the machine learning field in order to better understand the practical part of this Thesis.

Machine learning [9] is a scientific field that lies in the intersection of fields of computer science, engineering and statistics. Its application is broad as it is a tool that can applied to a multitude of problems related to data work and interpretation. In particular, machine learning is an area of artificial intelligence that involves the development of algorithms and methods that allow computers to "learn". Its goal is to build customizable and flexible computer programs that operate on the basis of automated data sets analysis rather than the intuition of the engineers who programmed them.

Typically, machine learning programs use training examples in order to build a model that will enable them to make reliable predictions.

In this chapter we will describe the fundamentals of machine learning, the algorithms in use, what are the applications of machine learning today and how it is used in the information security field.

4. 1 Fundamentals

On a high level, we can divide machine learning techniques into two categories: supervised and unsupervised learning.

4.1.1 Supervised Learning

As supervised learning we call the techniques used to learn the relationship between independent attributes and a designated dependent attribute (the label) [9]. In general, supervised learning the type of machine learning which is done with data that are labeled.

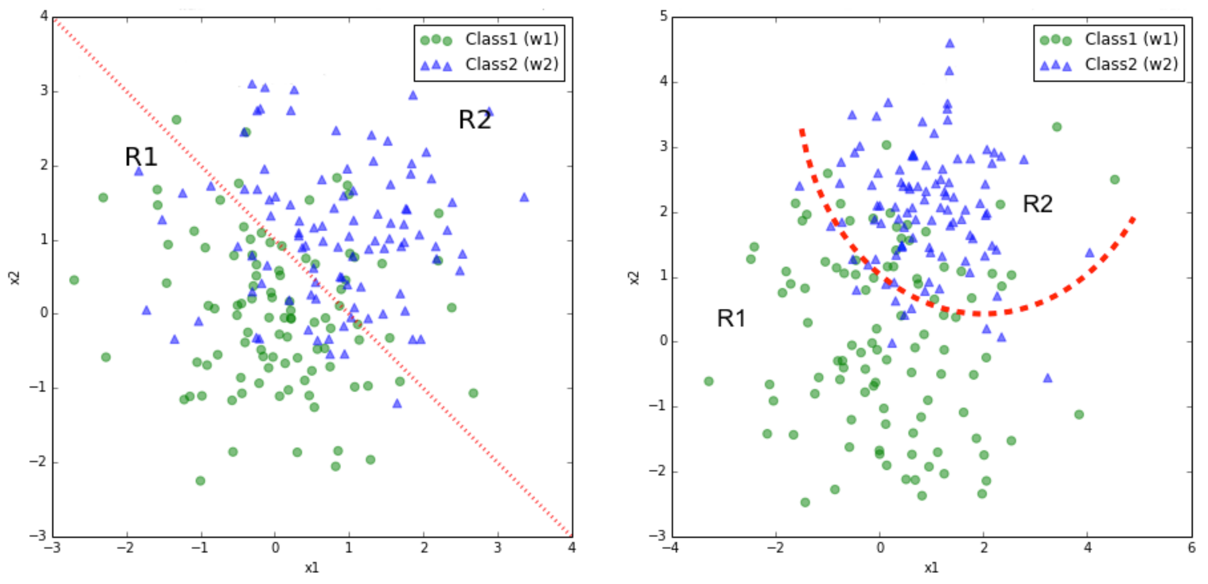
For example, if we want to make a classifier that will be able to detect malware, the data that we will need to use in order to train our model, have to be labeled as either malicious or benign.

Supervised learning solves two types of problems, classification and regression.

4.1.1.1 Classification

In classification, the goal is to predict the class to which an object belongs based on its property values. The basic point is that the categories are distinct.

A well-known example of the use of classification is the recognition of spam emails. Searching for a word is not sufficient to sort the message. But the occurrence of specific words combined with the size of the message and other factors leads to precision classification, something that is achieved with the help of engineering learning algorithms.

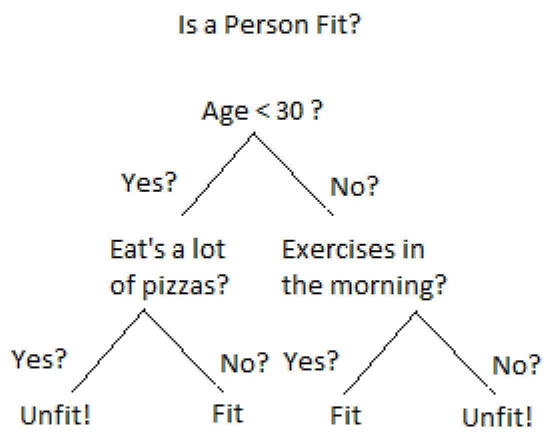


ii Classification example

Some of the algorithms that are used for classification are: Decision Trees, Logistic Regression, Random Forest, k-Nearest Neighbors, etc. [10]

Deciscion Trees

As the name suggests, decision trees are data structures in the format of a tree. More specifically, for the creation of the tree, the training dataset is used and also for making the prediction on the test data. The goal of this algorithm is to get the most accurate result with as less as decisions possible. A very naïve example of a decision tree is show in the following picture below where it demonstrates a classification problem of whether a person is fit or not.

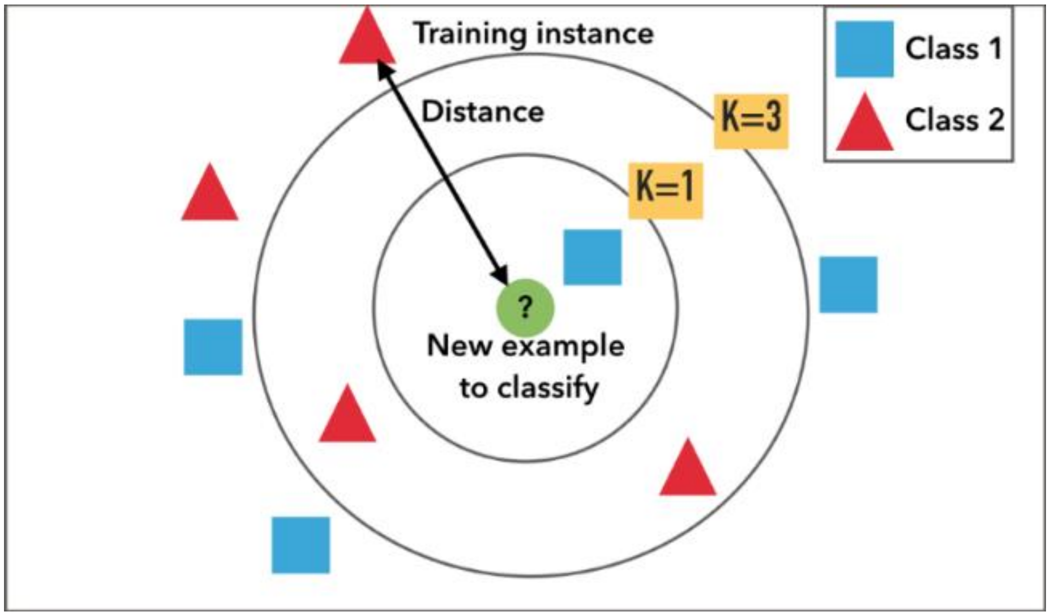


iii Decision tree

K-nearest neighbors

K-Nearest Neighbors (KNN) is considered one of the most accurate machine learning algorithms and yet quite simple. Its distinct characteristic from the other machine learning algorithms, is that it is a non-parametric algorithm, which means that it does not make assumptions about the structure of the data. This is a really important feature because in most of the real-world problems that machine learning algorithms face today, they almost always deal with unstructured data. So, KNN can be a first choice when we want to solve a classification problem without any prior knowledge about the data.

KNN is also characterized a *lazy* algorithm. What this means is that there is no training phase of the model for generalizing the data. The KNN algorithm Is based on feature similarity. For example, we have a dataset with two classes: C1, C2 and want to classify if a given sample S should be classified as either to the first or the second class.



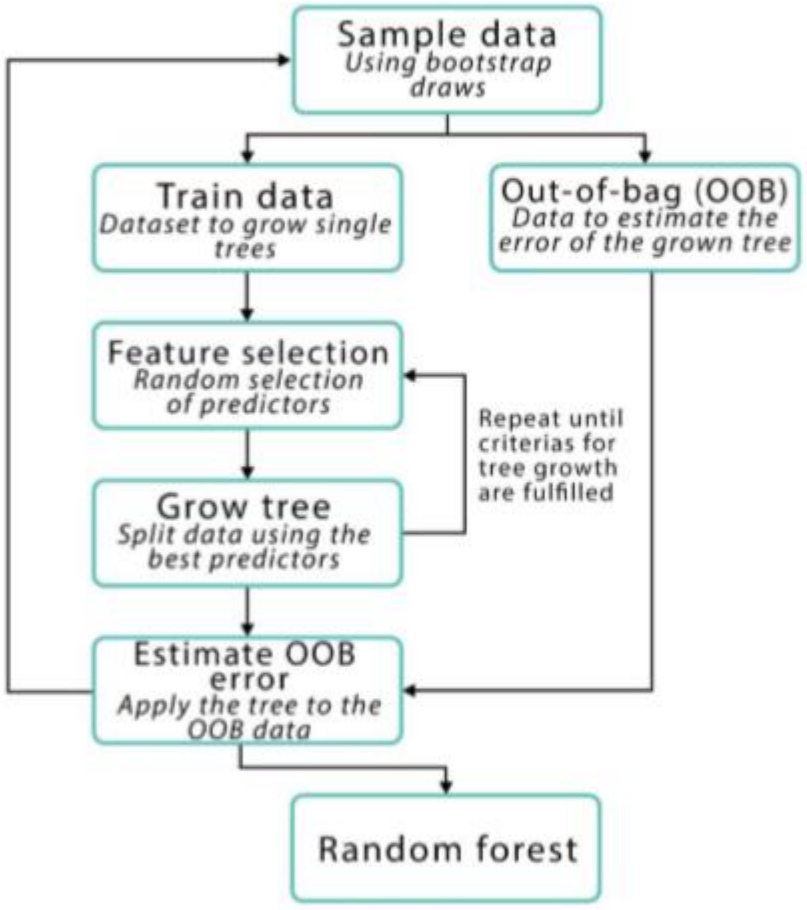
iv kNN example

So, the class that the sample S will be assigned to depends solely on the number we set k. In this case, if k=1 then the sample will be classified in the first Class C1 since there is only one object near the k circle of the class C1. Rather than, if k=3 it will be assigned to the second class C2.

Random Forest

Tin Kam Ho first introduced the general method of random decision forests at AT&T Bell Labs in 1995 [29]. The thought is, that the algorithm deducts the classification label for new documents from a set of decision trees where for each tree, a sample is selected from the training data, and a decision tree is created by choosing a random subset of all features (hence “Random”).

This is the one of the most popular machine learning algorithm because it does not require any data preparation and modelling and has very good results. A Random Forest is a set of decision trees producing better prediction accuracy.


Sample data
Using bootstrap draws
Train data
Dataset to grow single trees
Out-of-bag (OOB)
Data to estimate the error of the grown tree
Feature selection
Random selection of predictors
Repeat until criterias for tree growth are fulfilled
Grow tree
Split data using the best predictors
Estimate OOB error
Apply the tree to the OOB data
Random forest

V Random forest architecture

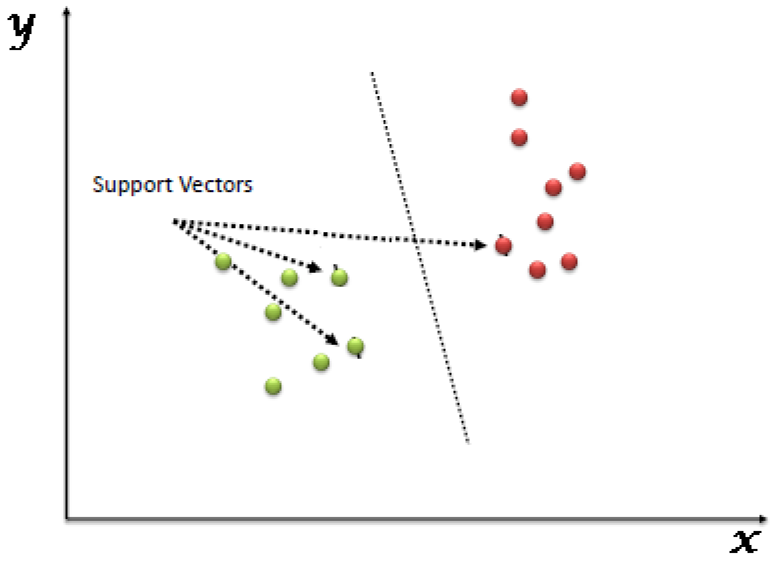
Suppose a training set is given as: [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], random forest may create three decision trees taking input of subset for example,

- [X1, X2, X3]
- [X1, X2, X4]
- [X2, X3, X4]

So finally, it predicts based on the majority of votes from each of the decision trees made.

Support Vector Machines (SVM)

This machine learning algorithm is not only used to solve classification as well as regression problems. The main idea of SVM is that it finds a hyperplane which differentiates two classes clearly.



vi Support Vector Machine (SVM)

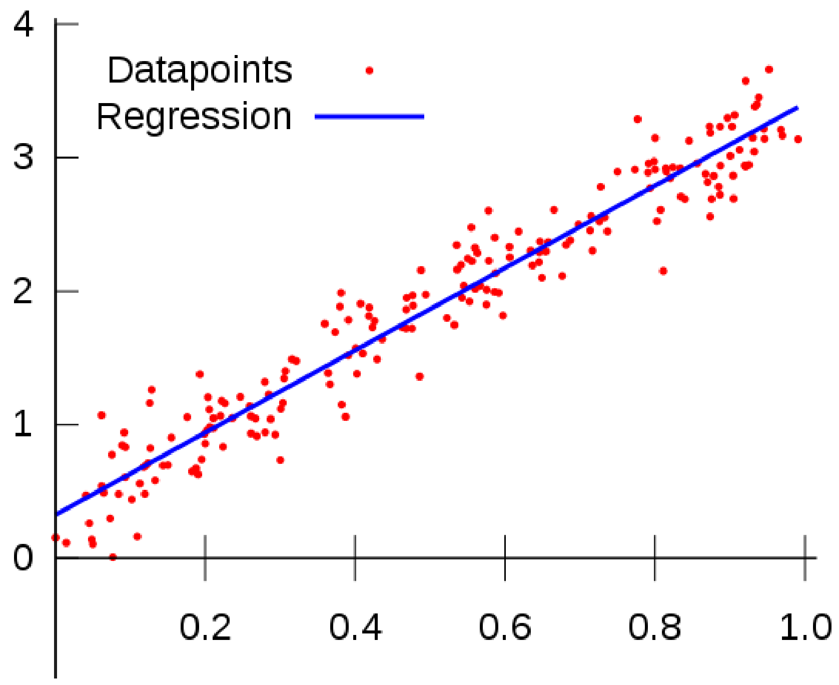
As shown in the picture above, with the hyperplane we separated the classes the best way possible. The term support vector refers to the points (data entries) lying closest to the hyperplane. The further our classes are from the hyperplane the more accurate the prediction is, so it is common for a problem to find multiple hyperplanes. The purpose of the SVM algorithm is to find a hyperplane that will result in the maximum margins possible.

Naïve Bayes

The Naïve Bayes classification machine learning algorithm is based on the Bayes Theorem [31]. The main characteristic of its method is that it evaluates every feature individually, regardless of any correlations and bases its prediction on the Bayes Theorem. Its drawback is that in most of the problems features have at least some level of correlation between each other.

4.1.1.2 Regression

The regression is the predictive value of a numeric value. A common example is finding the optimal line that runs through a set of points and represents the function.



vii Regression example

A continuous output variable is a real-value, such as an integer or floating-point value. These are often quantities, such as amounts and sizes [11].

For example, a house may be predicted to sell for a specific dollar value, perhaps in the range of $100,000 to $200,000.

- A regression problem requires the prediction of a quantity.

- A regression can have real valued or discrete input variables.
- A problem with multiple input variables is often called a multivariate regression problem.
- A regression problem where input variables are ordered by time is called a time series forecasting problem.

4.1.2 Unsupervised Learning

In unsupervised learning, the learning part is done without examples, and the algorithm is called to learn a concept from a set of data by creating the standards that exist, if any. Typically, unsupervised learning employs a clustering technique in order to group the unlabeled samples based on certain similarity (or distance) measures.

Unsupervised learning solves several kinds of problems such as grouping, probability density estimation, diminution of dimensions etc. For example, a clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior. Or, an association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y. [11]

4.1.3 Semi-Supervised Learning

As the title suggests, in semi-supervised learning some training data are labeled but most of them are not. This is quite common in a lot of real world machine learning problems as it is considered time-consuming to label huge amounts of data. Whereas, unlabeled data are much easier to collect.

You can also use supervised learning techniques to make best guess predictions for the unlabeled data, feed that data back into the supervised learning algorithm as training data and use the model to make predictions on new unseen data.

4.2 Machine Learning Examples

Machine learning is highly used nowadays due to the fact that the cloud solutions provide the ability to train huge amount of data without big costs for computing processing. Below are mentioned some examples categorized by their sector:

4.2.1 Retail

Most of the online shop (Amazon, E-bay, etc.) are using machine learning in order to provide to retailers personalized recommendations based on their purchases, or even their activity while browsing the website, thus providing a better customer service experience.

4.2.2 Social Media Services

Facebook’s DeepFace [12] algorithm uses facial dataset from the millions of Facebook’s users in order to detect nuances between human faces. Other than that, most social media nowadays they make suggestions on their users which are solely based on machine learning algorithms. For example, the *People You May Know* feature, or the *Face Recognition* for auto-tagging your friends on Facebook photos.

In the same type of machine learning application, Spotify [14], as well as Netflix use machine learning algorithm in order to figure out your likes and dislikes and suggest music or movies you might like. All these suggestions are refined as long as the user interacts more with the application and eventually they give better results. For example, if you are a newly registered user on Netflix, some of the suggestions may seem irrelevant for your preferences. However, after using the service more and more, and importantly provide feedback for every movie or TV series you watch, the algorithm is able to predict with a higher rate which shows you might be interested in.

4.2.3 Email

To further improve its phishing-detection performance, Google uses machine learning algorithms in order to detect spam emails. They also built a system that delays some Gmail messages for a little bit longer to perform more detailed phishing analysis [15].

Also, Gmail provides the option of “Smart Replies” where it suggests three responses based on the email you received [16].

4.2.4 Personal Assistants

Machine learning is important part of personal assistants since it enables them to collect and refine information based on the user’s previous involvement with them [13].

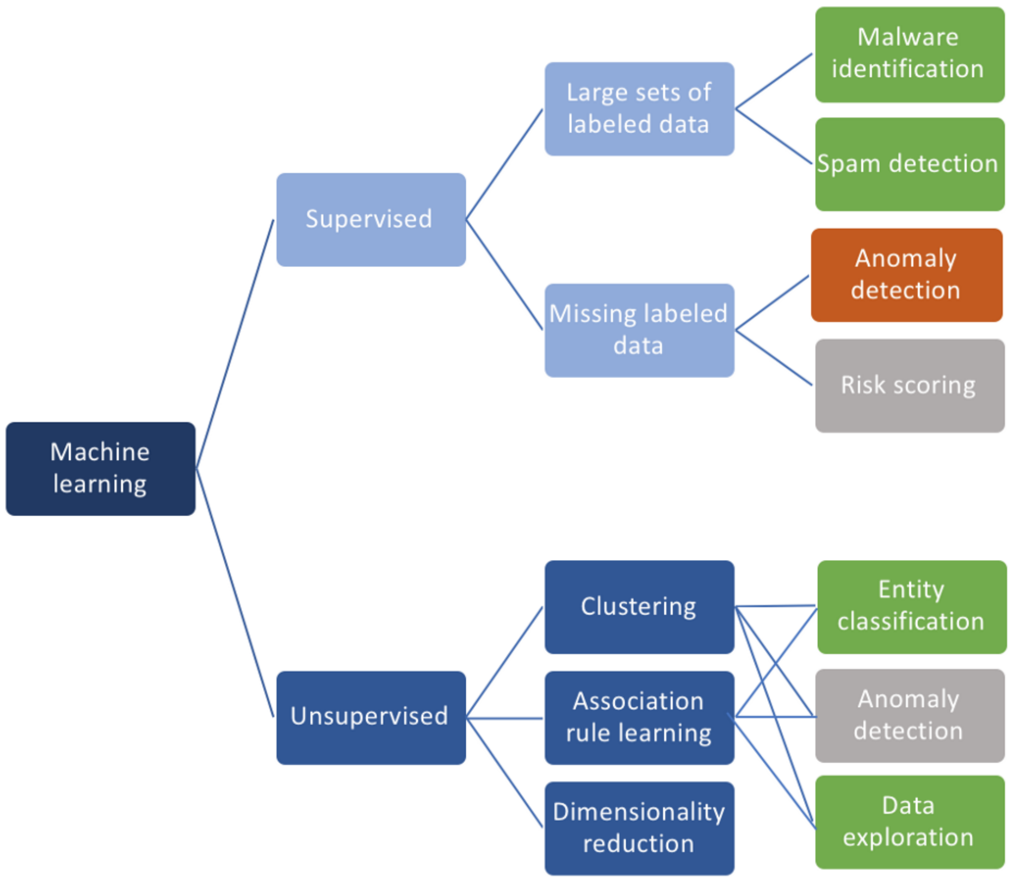
4.2.5 Information Security

Google’s reCAPTCHA uses machine learning in order to distinguish between human and bots on the web. reCAPTCHA offers more than just spam protection. Every time our CAPTCHAs are solved, that human effort helps digitize text, annotate images, and build machine learning datasets. This in turn helps preserve books, improve maps, and solve hard AI problems. [17]

Companies like DeepInstinct [18] where one of the first ones that use deep learning in order to detect anomalies in the cyber security field.

Respond Software [19] is a company which developed a method to run domain experts through a set of scenarios that they have to ‘rate’. Based on that input, they then build a statistical model which distinguishes ‘information’ from ‘noise’.

The following figure presents some of the usages of machine learning in cyber security.



viii ML in cyber-security

Even more and more cyber security firms are implementing Machine Learning algorithms in order to be able to stay ahead in the race against adversaries.

As long as more and more organizations invest in machine learning in order to improve their services, inevitably their security of those solutions is tested by the information security community. Therefore, a new field has raised which intersects between machine learning and computer security, and it’s called Adversarial Machine Learning [20].

5. Adversarial Machine Learning

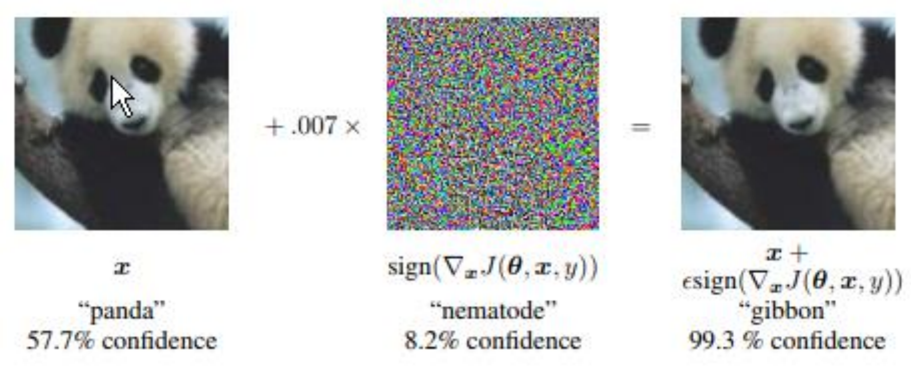
5.1 Threat Model

As in a regular penetration test where the adversary tries to infiltrate computer systems by exploiting known, or even unknown, vulnerabilities; the same approach applies for adversarial machine learning. We can even categorize the types of adversarial attacks on the classifiers as black, grey or white box. A black-box is when the adversary has no information about the classifier he is trying to exploit, grey-box when he has partial information (e.g. percentage of prediction), and white-box is when the attacker knows how the classifier is built, where the data come from etc.

At high level the attacks on machine learning classifiers can be divided into: evasion or poisoning attacks.

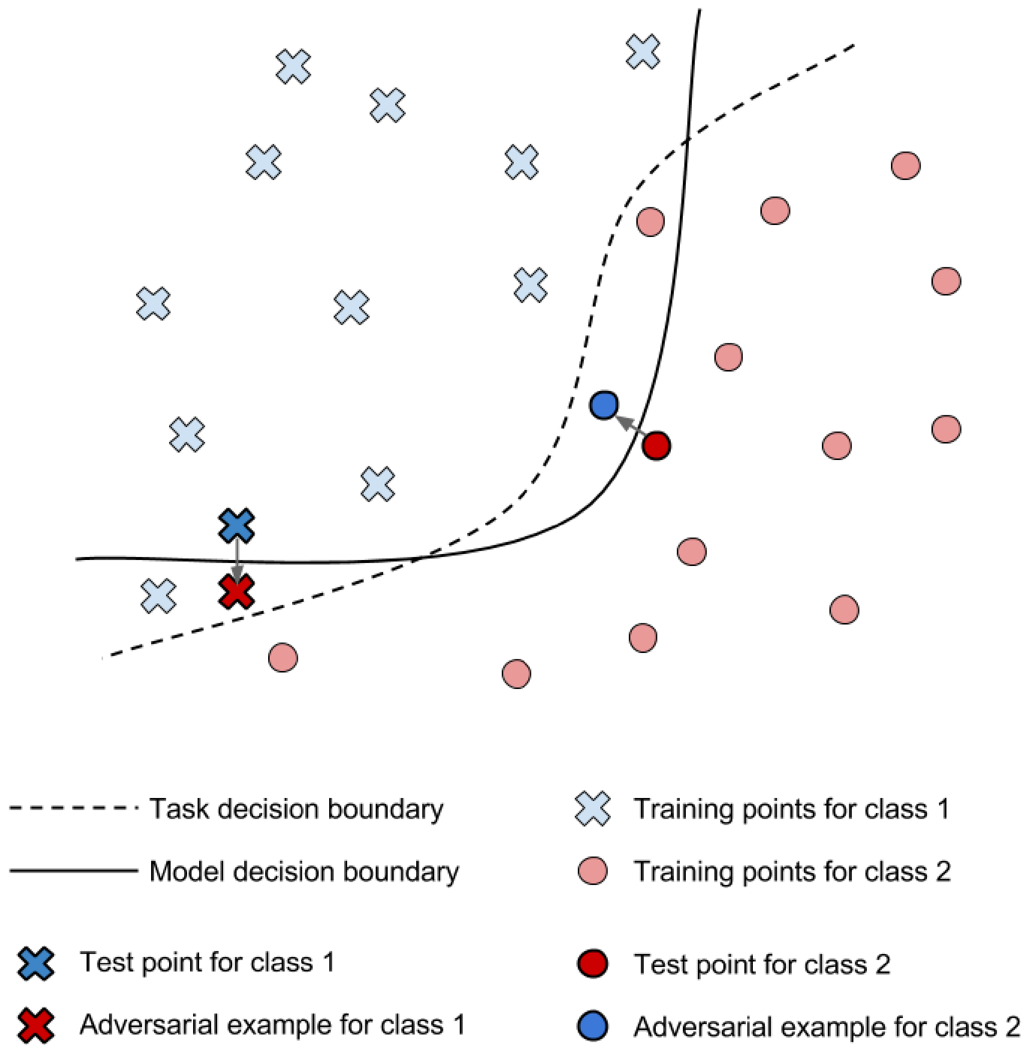
5.1.2 Evasion Attacks

Evasion attacks are the simplest kind of attacks where the adversaries try to manipulate their sample in order to be misclassified by the classifier [21] [22]. An example is the following picture [24], where a correctly classified image of a panda is being slightly modified by adding the second image, which is basically noise in order to manipulate the data of the initial image. The image on the right is the variant generated in which humans are not able to notice any difference between the original one but the classifier fails to classify it correctly.


$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

A common way to find perturbations forcing models to make wrong predictions is to compute adversarial examples [20]. The yield perturbations that are very slight and often indistinguishable to humans, yet force machine learning models to produce wrong predictions.

Note that despite being indistinguishable to the human eye, the perturbation is sufficient to change the model's prediction. Indeed, the perturbation is computed to minimize a specific norm in the input domain while increasing the model’s prediction error. This effectively pushes the input, which was originally correctly classified, across the model’s decision boundary into a wrong class. This phenomenon is illustrated in the 2D Figure below, for a problem with two output classes.


Task decision boundary
Model decision boundary
Test point for class 1
Adversarial example for class 1
Training points for class 1
Training points for class 2
Test point for class 2
Adversarial example for class 2

ix

Most attacks which are based on adversarial examples require that the adversary has some knowledge on the machine learning model parameters in order to optimize the adversarial samples better.

5.1.3 Poisoning Attacks

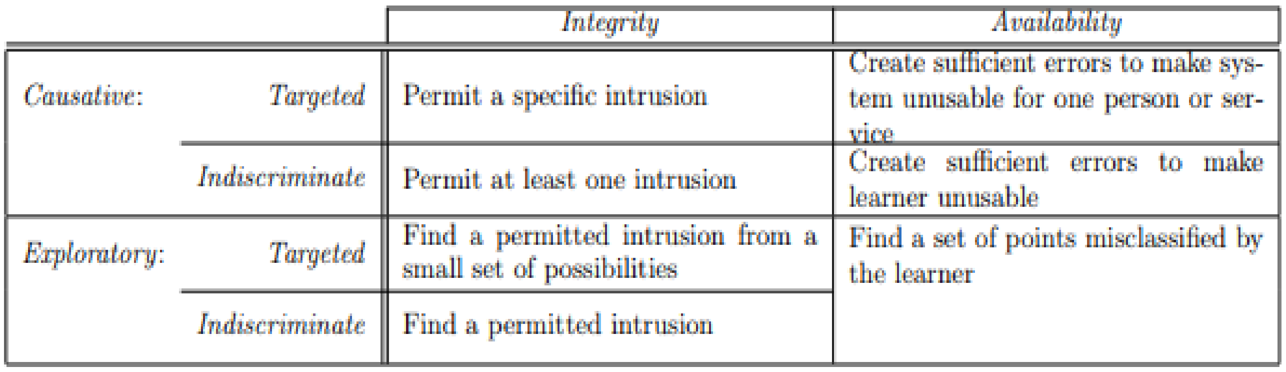
An attacker can inject arbitrary data during the learning phase of the classifier in order to influence the results of the classifier. For example, imagine an IDS which uses its incoming traffic as training data in order to train their classifier model that detects anomalous traffic, an adversary can send traffic to the network that will increase the prediction error of the algorithm. This is an attack on the integrity of the data of the classifier [23].

The attacks on the classifier extend beyond the usual attacks on Privacy, Availability and Integrity, as we are used to in the information security field. Researchers *[25]* have categorized potential attacks in 3 properties:

1. If the attack is exploratory or causative. Meaning if the adversary is able to identify the learning outcomes from the classifier or even influence the data.

2. Whether the attack is targeted or indiscriminate. For example, if the attack has a more wide-flexible goal or is targeted on a single purpose (e.g. bypass my own malware from a classifier, or bypass any malware from the classifier)

3. Lastly, if the attack focuses on any of the Integrity, Privacy, Availability. The integrity refers to the data of the model, the availability on the false-positive rate and the privacy about information from the model itself.



		Integrity	*Availability*
Causative:	*Targeted*	Permit a specific intrusion	Create sufficient errors to make system unusable for one person or service
	Indiscriminate	Permit at least one intrusion	Create sufficient errors to make learner unusable
Exploratory:	*Targeted*	Find a permitted intrusion from a small set of possibilities	Find a set of points misclassified by the learner
	Indiscriminate	Find a permitted intrusion	

x. Attack Model

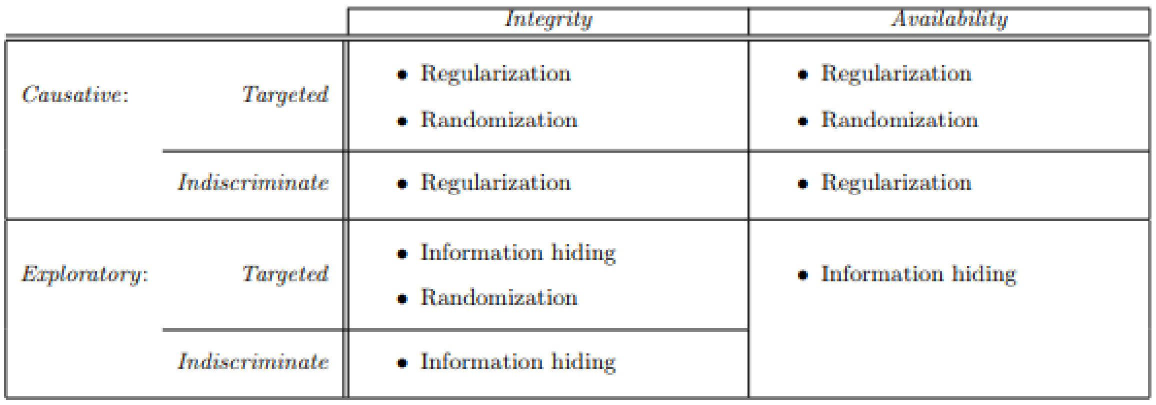
5.2 Countermeasures

Like any other computer problem that needs a security solution, the solution is not straightforward. We need to use layered defenses in order to minimize the risk of our machine learning model. For poisoning attacks, the defenses rely mostly on the integrity of our data set. Identifying the sources of our data and evaluating if they are reliable is a good security practice when we are building our model. Moreover, since most datasets come from various sources it’s unable to check every single one of them. So, it is necessary to have built-in checks while training your algorithm.

Adversarial training is a brute force solution where we generate a lot of adversarial examples and explicitly train the model with them in order for it to learn how to identify them.

Another straight forward solution is by training two classifiers - one with the base training set, and another with the base training set and the new data sample. Both classifiers are then subjected to a set of tests in order to determine their accuracy. If the second classifier is significantly worse than the first then the new training data can be discarded marked as malicious.

The following table represents some of the defenses against the attacks mentioned earlier.



		Integrity	*Availability*
Causative:	*Targeted*	• Regularization • Randomization	• Regularization • Randomization
	Indiscriminate	• Regularization	• Regularization
Exploratory:	*Targeted*	• Information hiding • Randomization	• Information hiding
	Indiscriminate	• Information hiding	

xi Defenses

These countermeasures though, rely on poisoning attacks only, whereas evasion attacks are much harder to counter since improved machine learning algorithms are also better at generating evasion attacks. It is highly important for an organization that utilizes machine learning in their production environment to consider this drawback of machine learning algorithms in general and be able to accept a security attack on the classifier without big impact on the organizational assets.

6 Implementation

For this Thesis we focused on supervised learning and more specifically on classification. We created a malware classifier for portable executable (PE) files as a proof of concept. We compared various well-known classification algorithms and chose the one that performed best. We collected our dataset from various online resources, containing both malicious and legitimate files.

We divided the phases of our machine learning implementation into four steps: data collection, feature selection, model training & evaluation. This is a high-level approach; every step included more sub-steps but for the purposes of this Thesis we mention in details only the four of them.

Before engaging in those steps, there were a couple of decisions that we made in order to proceed with our implementation. First of all, the data type we chose for our implementation is Portable Executable (PE) file format. For malware detection there are also plenty of options besides executables, we could also choose Microsoft Word documents, PDF files, or even PowerShell scripts. It is highly important for our data to be of the same type in order for our model to work properly.

6.1 Data Collection

We collected a total of 2221 legitimate files from Windows 10 core system files. Specifically, from the following paths from my personal computer:

```
# Paths for legitimate executables
C:\Windows\System32
C:\Program Files\
C:\Program Files (x86)\
```

For the malicious executables we used Virus Share’s malware collection [27]. In total we had 1998 malicious files for our learning algorithm.

We stored them in different folders, malicious and clean and used a python program to parse them, extract their features and store them in a CSV file.

```
# Import our data.csv with the list of executables
data = pandas.read_csv('C://Users//Ethan//Desktop//Malware-
Classifier//data.csv', sep='|')
x = data.drop(['Name', 'md5', 'clean'], axis=1).values
y = data['clean'].values

# Divide clean and malicious files in two lists for further usage
clean = data[0:2222].drop(['clean'], axis=1)
malicious = data[2222::].drop(['clean'], axis=1)

print "Total number of entries: %s" % len(y)
print "%s clean and %s malicious" % (len(clean),len(malicious))
# Sample Output
Total number of entries: 4218
2222 clean and 1996 malicious
```

Sample legitimate entry:

```
calc.exe|60b7c0fead45f2066e5b805a91f4f0fc|332|224|258|9|0|339456|435712|0|77164|4096|335872|16777216|4096|512|6|1|6|1|6|1|786432|1024|834864|2|33088|262144|8192|1048576|4096|0|16|4|5.47901717806|1.23095974357|7.54041397387|193792.0|15360|403456|193549.25|15164|403352|16|380|12|0|91|6.98314896545|2.88489987072|7.94986627576|4378.82417582|230|27569|72|16|1
```

Sample malicious entry:

```
VirusShare_a3f6169845e9c5fb2c3f80fd48513d2b|a3f6169845e9c5fb2c3f80fd48513d2b|332|224|258|10|0|118784|382464|0|59882|4096|122880|4194304|4096|512|5|1|0|0|5|1|520192|1024|553648|2|33088|1048576|4096|1048576|4096|0|16|5|5.66094145399|4.20006478048|7.95987755287|100249.6|9216|335360|101703.4|9354|335248|7|112|0|0|26|6.97809692101|2.45849222582|7.98853208613|12840.3846154|48|20394|72|14|0
```

The next step is to select which features we are going to use from our PE files for our model.

6.2 Feature Selection

Feature selection is a very important step for the efficiency of our classifier. In most of the real-world learning problems, the features which are selected in order to represent data are too many making it harder for the algorithm to understand which of them are relevant to the target concept [26]. And by harder we basically mean that it will make the learning process of the algorithm longer and probably not with good performance.

We extracted most of the PE parameters possible like: DllCharacteristics, SizeOfOptionalHeaders, ResourcesMaxEntropy, etc. and used a Tree-Based feature selection algorithm [28]

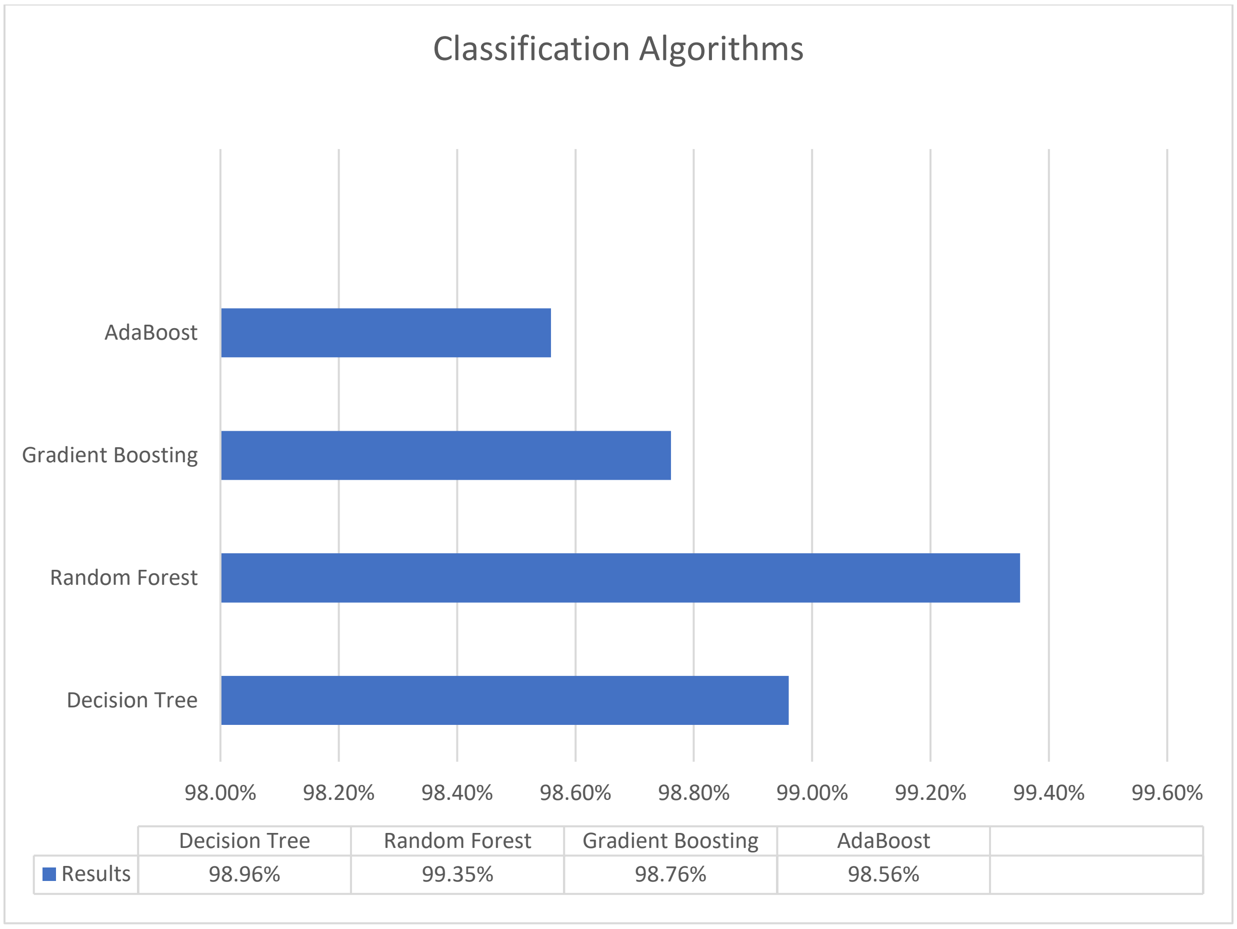
```
# So, in order to select the best features for our case we can
# use the Tree-Based feature selection algorithm
# http://scikit-learn.org/stable/modules/feature_selection.html#tree-based-feature-selection

clf = ExtraTreesClassifier()
```

```
f_selector = clf.fit(x, y)
model = SelectFromModel(f_selector, prefit=True)
X_new = model.transform(x)
X_new.shape

nb_features = X_new.shape[1]
indices = np.argsort(f_selector.feature_importances_)[::-1][:nb_features]
features = [data.columns[2+indices[f]] for f in range(nb_features)]

print features

['DllCharacteristics', 'Subsystem', 'SizeOfOptionalHeader', 'VersionInformationSize', '
ResourcesMaxEntropy', 'MajorSubsystemVersion', 'ResourcesMinEntropy', 'SectionAlignment
', 'MajorLinkerVersion', 'ResourcesMeanEntropy', 'SectionsMaxEntropy', 'Machine', 'Size
OfHeaders', 'FileAlignment', 'LoadConfigurationSize']
```

6.3 Training

Before proceeding with training our model, we have to select which classification algorithm we will use. In order to do that we need to compare some of them and choose the one that performs the best with our current dataset.

After running the test on four of the most common classification algorithms we got the following results:



	Decision Tree	Random Forest	Gradient Boosting	AdaBoost	
🟦 Results	98.96%	99.35%	98.76%	98.56%	

So, we will use the RandomForest as our classifier as it performed the best with our training data.

```
# For the classification part we selected the RandomForest algorithm.
# But first of all, we need to get our training and test data.

x_train, x_test, y_train, y_test = cross_validation.train_test_split(X_new, y ,test_siz
e=0.2)

clf = RandomForestClassifier(n_estimators=50) # we can play with n for better results
clf.fit(x_train, y_train)
score = clf.score(x_test, y_test)
print("Random Forest classification results : %f %%" % (score*100))

# Output
Random Forest classification results: 99.35100 %
```

6.4 Results

Some samples we tried with our classifier and their results:

Filename	VirusTotal	Classifier
Putty.exe	1/60	Clean
Skype-portable.exe	4/67	Clean
Msf_reverse_shell.exe	50/63	Malicious
Msf_reverse_shell(enc).exe	48/63	Malicious
Msf_reverse_shell.exe (peCloak)	35/64	Malicious
Msf_reverse_shell.exe (ssl encypted)	49/65	Malicious
0day crypter	N/A	Malicious (n>65)
Putty_64bit.exe	0/63	Malicious (FP)
Skype_64bit.exe	1/67	Malicious (FP)

In total our classifier was able to detect malicious PE files with a successful rate of 99,28%.

Evaluation	Score
False positive	1.237624 %
False negative	0.227273 %

6.5 Attacking the Classifier

For the purposes of this Thesis we attempt to demonstrate an evasion attack.

6.5.1 Getting started

Since we know how the model works, the classifier it uses, the features it extracts, etc. it is considered a white-box attack.

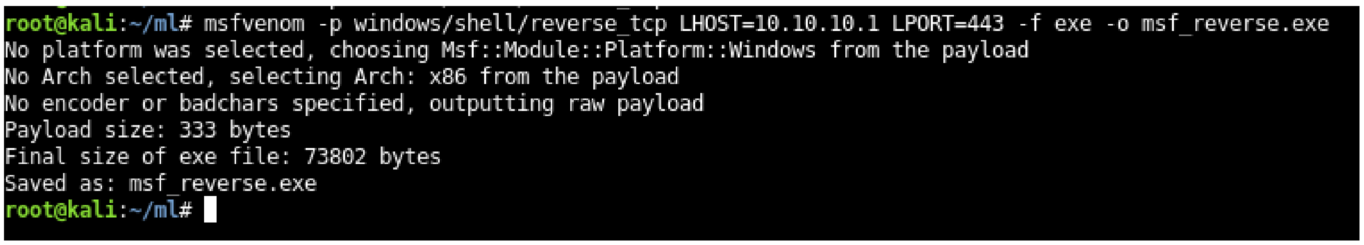
A successful attack will be if we manage to find a variant of our malicious file which will be classified as legitimate by the classifier. In order to succeed we need to modify our existing malicious file by making small changes into its data, and most importantly to the features that the classifier uses for classifications, so that it gets misclassified as clean classifier. Another successful attack would be if we can manage to create a new clean file which is falsely classified as malicious by the classifier.

The steps we followed are the following:

1. Generate malicious payload
2. Analyze it with our classifier
3. If it is classified as malicious perform one cycle of evasive transformation and analyze it again
4. Continue the same process until the file is classified as legitimate or the file is corrupted.

6.5.2 Malicious Payload Generation

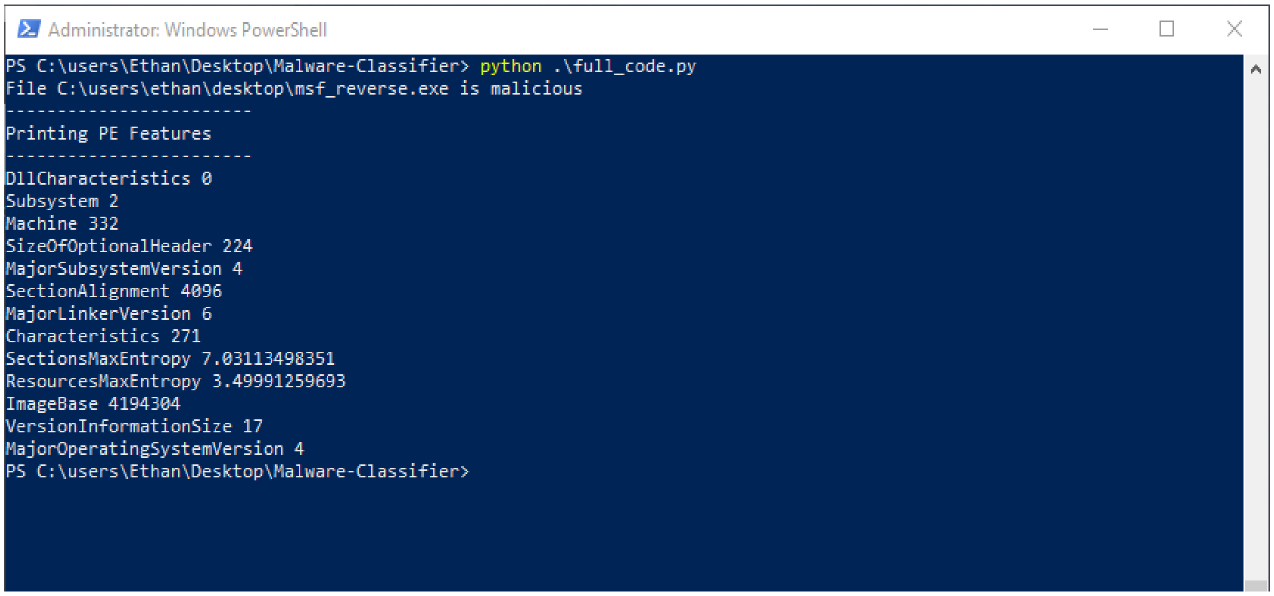
We used Metasploit to generate a malicious windows reverse shell.



```
root@kali:~/ml# msfvenom -p windows/shell/reverse_tcp LHOST=10.10.10.1 LPORT=443 -f exe -o msf_reverse.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of exe file: 73802 bytes
Saved as: msf_reverse.exe
root@kali:~/ml# 
```

6.5.3 Classification

We test it against our classifier and we see that it detects that it is a malicious file.



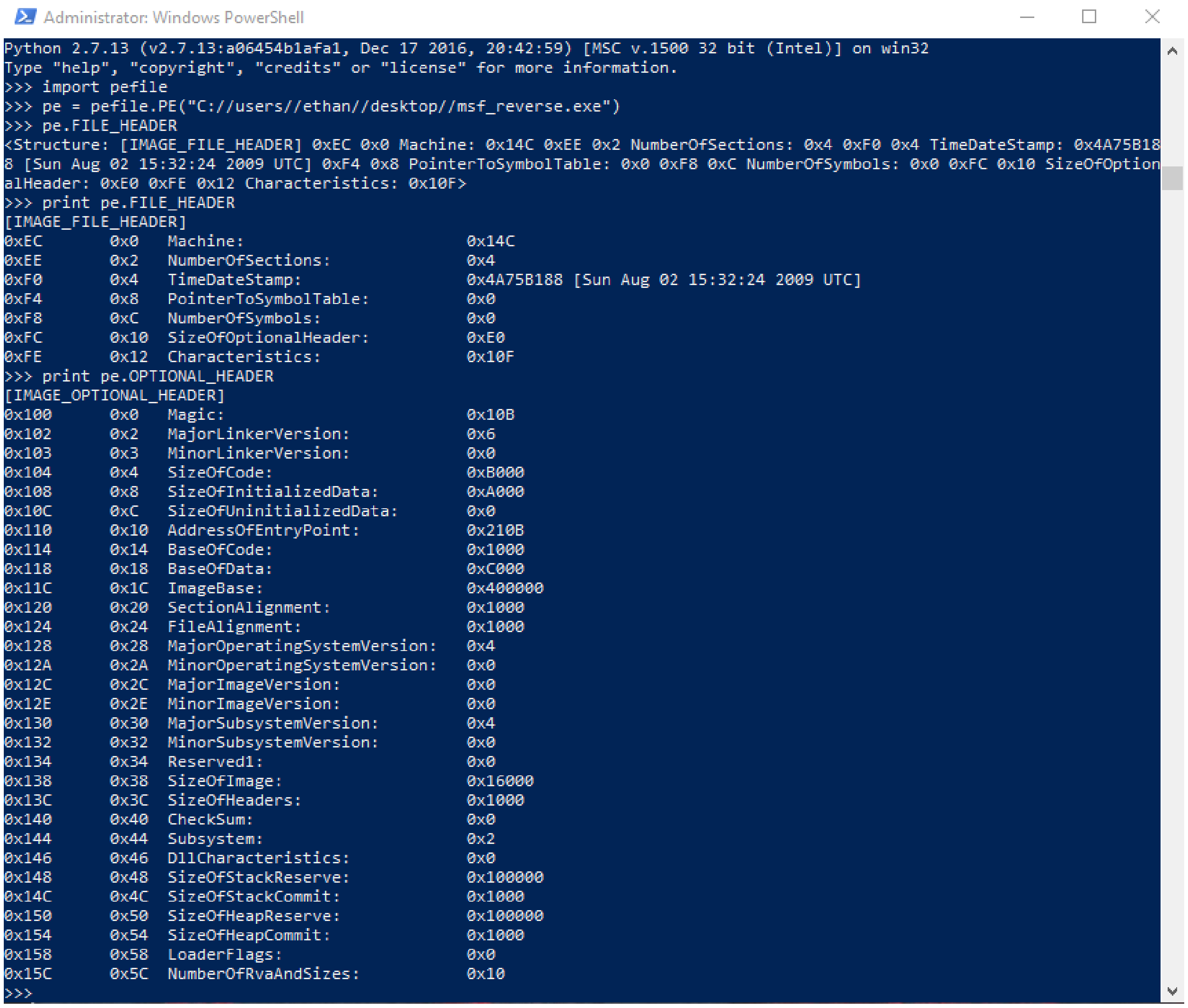
```
PS C:\users\Ethan\Desktop\Malware-Classifier> python .\full_code.py
File C:\users\ethan\desktop\msf_reverse.exe is malicious
------------------------
Printing PE Features
------------------------
DllCharacteristics 0
Subsystem 2
Machine 332
SizeOfOptionalHeader 224
MajorSubsystemVersion 4
SectionAlignment 4096
MajorLinkerVersion 6
Characteristics 271
SectionsMaxEntropy 7.03113498351
ResourcesMaxEntropy 3.49991259693
ImageBase 4194304
VersionInformationSize 17
MajorOperatingSystemVersion 4
PS C:\users\Ethan\Desktop\Malware-Classifier>
```

From the execution of the classifier we see that the important features it selected for the classification are: SizeOfOptionalHeader, DllCharacteristics, Subsystem, Machine, SectionAlignment, MajorLinkerVersion, Characteristics, SectionsMaxEntropy, ResourcesMaxEntropy, ImageBase, VersionInformationSize, MajorOperatingSystemVersion.

Features	Values
DllCharacteristics	0
Subsystem	2
Machine	332
SizeOfOptionalHeader	224
MajorSubsystemVersion	4
SectionAlignment	4096
MajorLinkerVersion	6
Characteristics	271
SectionsMaxEntropy	7.03113498351
ResourcesMaxEntropy	3.49991259693
ImageBase	4194304
VersionInformationSize	17
MajorOperatingSystemVersion	4

6.5.4 Features

Then we extract the files pe features in order to manipulate the ones who are relevant and the classifier uses.



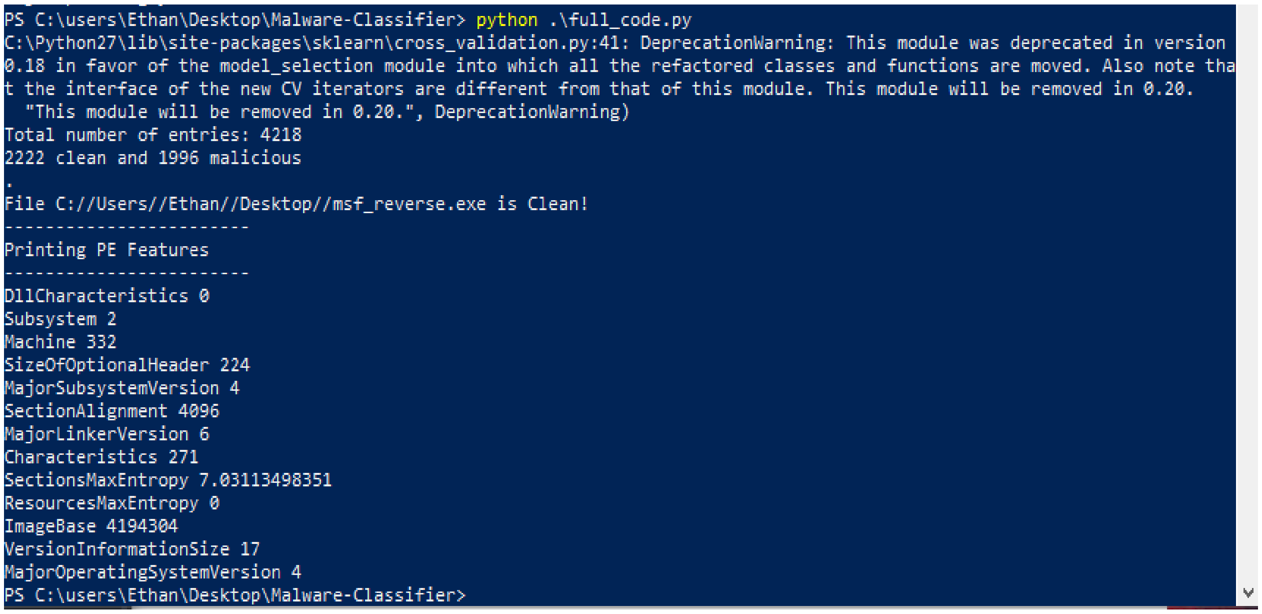
Administrator: Windows PowerShell

```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pefile
>>> pe = pefile.PE("C://users//ethan//desktop//msf_reverse.exe")
>>> pe.FILE_HEADER
<Structure: [IMAGE_FILE_HEADER] 0xEC 0x0 Machine: 0x14C 0xEE 0x2 NumberOfSections: 0x4 0xF0 0x4 TimeDateStamp: 0x4A75B18
8 [Sun Aug 02 15:32:24 2009 UTC] 0xF4 0x8 PointerToSymbolTable: 0x0 0xF8 0xC NumberOfSymbols: 0x0 0xFC 0x10 SizeOfOption
alHeader: 0xE0 0xFE 0x12 Characteristics: 0x10F>
>>> print pe.FILE_HEADER
[IMAGE_FILE_HEADER]
0xEC       0x0   Machine:                       0x14C
0xEE       0x2   NumberOfSections:              0x4
0xF0       0x4   TimeDateStamp:                 0x4A75B188 [Sun Aug 02 15:32:24 2009 UTC]
0xF4       0x8   PointerToSymbolTable:          0x0
0xF8       0xC   NumberOfSymbols:               0x0
0xFC       0x10  SizeOfOptionalHeader:          0xE0
0xFE       0x12  Characteristics:               0x10F
>>> print pe.OPTIONAL_HEADER
[IMAGE_OPTIONAL_HEADER]
0x100      0x0   Magic:                         0x10B
0x102      0x2   MajorLinkerVersion:            0x6
0x103      0x3   MinorLinkerVersion:            0x0
0x104      0x4   SizeOfCode:                    0xB000
0x108      0x8   SizeOfInitializedData:         0xA000
0x10C      0xC   SizeOfUninitializedData:       0x0
0x110      0x10  AddressOfEntryPoint:           0x210B
0x114      0x14  BaseOfCode:                    0x1000
0x118      0x18  BaseOfData:                    0xC000
0x11C      0x1C  ImageBase:                     0x400000
0x120      0x20  SectionAlignment:              0x1000
0x124      0x24  FileAlignment:                 0x1000
0x128      0x28  MajorOperatingSystemVersion:   0x4
0x12A      0x2A  MinorOperatingSystemVersion:   0x0
0x12C      0x2C  MajorImageVersion:             0x0
0x12E      0x2E  MinorImageVersion:             0x0
0x130      0x30  MajorSubsystemVersion:         0x4
0x132      0x32  MinorSubsystemVersion:         0x0
0x134      0x34  Reserved1:                     0x0
0x138      0x38  SizeOfImage:                   0x16000
0x13C      0x3C  SizeOfHeaders:                 0x1000
0x140      0x40  CheckSum:                      0x0
0x144      0x44  Subsystem:                     0x2
0x146      0x46  DllCharacteristics:            0x0
0x148      0x48  SizeOfStackReserve:            0x100000
0x14C      0x4C  SizeOfStackCommit:             0x1000
0x150      0x50  SizeOfHeapReserve:             0x100000
0x154      0x54  SizeOfHeapCommit:              0x1000
0x158      0x58  LoaderFlags:                   0x0
0x15C      0x5C  NumberOfRvaAndSizes:           0x10
>>>
```

The advantage that the adversary has in white-box attacks is that he can fine-tune the evasion cycle in order to have better results against the specific classifier. This means that we will adjust our encoder in order to modify the features that are important for the classification.

6.5.5 Evasive Transformation

For the generation of adversarial examples, we used an open-source encoder for PE files and we modified it in order to better perform against the classifier.



```
PS C:\users\Ethan\Desktop\Malware-Classifier> python .\full_code.py
C:\Python27\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version
0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note tha
t the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
Total number of entries: 4218
2222 clean and 1996 malicious
.
File C://Users//Ethan//Desktop//msf_reverse.exe is Clean!
------------------------
Printing PE Features
------------------------
DllCharacteristics 0
Subsystem 2
Machine 332
SizeOfOptionalHeader 224
MajorSubsystemVersion 4
SectionAlignment 4096
MajorLinkerVersion 6
Characteristics 271
SectionsMaxEntropy 7.03113498351
ResourcesMaxEntropy 0
ImageBase 4194304
VersionInformationSize 17
MajorOperatingSystemVersion 4
PS C:\users\Ethan\Desktop\Malware-Classifier>
```

Features *(after encoding)*	**Values**
DllCharacteristics	0
Subsystem	2
Machine	332
SizeOfOptionalHeader	224
MajorSubsystemVersion	4
SectionAlignment	4096
MajorLinkerVersion	6
Characteristics	271
SectionsMaxEntropy	7.03113498351
ResourcesMaxEntropy	**0**
ImageBase	4194304
VersionInformationSize	17
MajorOperatingSystemVersion	4

As we can observe the encoder identified that only by changing the value **ResourcesMaxEntropy** is sufficient in order for our payload to be misclassified as clean.

This is expected since Entropy is one of the features with the highest “weight” in our classifier. Usually files with high entropy values are more likely to be malicious that’s why our classifier bases its prediction a lot from the values of the features SectionsMaxEntropy and ResourcesMaxEntropy.

Conclusions

The results from our malware classifier implementation show that there is promising future on machine learning for the security industry but the traditional methods of detection are still valid. It is also worth mentioning that in the most cases, the antiviruses detected the malicious payloads during their “Heuristics” analysis of the file. Although, the sample data that we used were not a lot, they gave us a very good overview of what are the important features for malware detection. Also, for future work it may be interesting to see how the classifier would perform it is trained with a very large amount of data, and how well does it detect adversarial input.

References

[1] The Art of Computer Virus Research and Defense. Szor P. February 2005

[2] https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017

[3] A Comparative Study of Virus Detection Techniques, Sulaiman Al Amro, Ali Alkhalifah, November 2015

[4] Morales, J.A., 2008. A behavior-based approach to virus detection, Florida International University. Sulaiman Al Amro, Ali Alkhalifah

[5] Signature Based Malware Detection is Dead, James Scott, Sr. Fellow. ICIT, February 2017

[6] Conry-Murray, A., 2002. Behavior-blocking stops unknown malicious code. Network Magazine.

[7] Morales, J. A., Clarke, P. J. and Deng, Y., 2010. Identification of file infecting viruses through detection of self-reference replication. Journal in computer virology, 6(2), pp. 161-180

[8] Microsoft, "Microsoft Portable Executable and Common Object File Format https://msdn.microsoft.com/enus/windows/hardware/gg463119.aspx

[9] R. Kohavi and F. Provost, "Glossary of terms," Machine Learning, vol. 30, no. 2–3, pp. 271–274, 1998

[10] Supervised Machine Learning: A Review of Classification Techniques, S. B. Kotsiantis, July 2007

[11] https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

[12] http://www.adweek.com/digital/deepface/

[13] https://www.amazon.jobs/en/teams/alexa-machine-learning

[14] https://medium.com/s/story/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe

[15] https://techcrunch.com/2017/05/31/google-says-its-machine-learning-tech-now-blocks-99-9-of-gmail-spam-and-phishing-messages/

[16] https://blog.google/products/gmail/save-time-with-smart-reply-in-gmail/

[17] https://www.google.com/recaptcha/intro/v3beta.html

[18] https://www.deepinstinct.com/

[19] https://respond-software.com/pages/product

[20] Intriguing properties of neural networks, C. Szegedy et al., arxiv 2014

[21] D. Lowd and C. Meek, “Adversarial learning,” in ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, 2005, pp. 641–647.

[22] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In KDD ’04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining, pages 99–108. ACM Press, 2004.

[23] Biggio, B., Nelson, B., & Laskov, P. (2012). Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389.

[24] Explaining and Harnessing Adversarial Examples, I. Goodfellow et al., ICLR 2015

[25] https://people.eecs.berkeley.edu/~tygar/papers/SML2/Adversarial_AISEC.pdf

[26] A Practical Approach to Feature Selection, K Kira, L A. Rendell, 1992

[27] https://archive.org/download/virusshare_malware_collection_000

[28] http://scikit-learn.org/stable/modules/feature_selection.html#tree-based-feature-selection

[29] Random decision forests, T K Ho, 1995 Bell Labs.

[30] https://deepmind.com/research/alphago/

[31] https://en.wikipedia.org/wiki/Bayes%27_theorem

[32] A Static Malware Detection System Using Data Mining Methods, Usukhbayar Baldangombo, Nyamjav Jambaljav, Shi-Jinn Horng, 2013

[33] Weilin Xu, Yanjun Qi, and David Evans. Automatically Evading Classifiers, A Case Study on PDF Malware Classifiers. Network and Distributed Systems Symposium 2016, 21-24 February 2016, San Diego,

[34] Practical Black-Box Attacks against Machine Learning, N. Papernot, 2016

Appendix

Source code (Python)

```python
'''''
@author Thanos Kalachanis - MTE1510
'''
import os
import pandas
import extractor
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
from sklearn import cross_validation
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
from sklearn.feature_selection import SelectFromModel

# Import our data.csv with the list of executables
data = pandas.read_csv('C://Users//Ethan//Desktop//Malware-Classifier//data.csv', sep='|')
x = data.drop(['Name', 'md5', 'clean'], axis=1).values
y = data['clean'].values

# Divide clean and malicious files in two lists for further usage
clean = data[0:2222].drop(['clean'], axis=1)
malicious = data[2222::].drop(['clean'], axis=1)

print "Total number of entries: %s" % len(y)
print "%s clean and %s malicious" % (len(clean),len(malicious))

# So in order to select the best features for our case we can
# use the Tree-Based feature selection algorithm
# http://scikit-learn.org/stable/modules/feature_selection.html#tree-based-feature-selection
clf = ExtraTreesClassifier()
f_selector = clf.fit(x, y)
model = SelectFromModel(f_selector, prefit=True)
X_new = model.transform(x)
X_new.shape

nb_features = X_new.shape[1]
indices = np.argsort(f_selector.feature_importances_)[::-1][:nb_features]
features = [data.columns[2+indices[f]] for f in range(nb_features)]

print features

# For the classification part we selected the RandomForest algorithm.
# But first of all we need to set our training and test data.
x_train, x_test, y_train, y_test = cross_validation.train_test_split(X_new, y ,test_size=0.2)

clf = RandomForestClassifier(n_estimators=75) # we can play with n for better results
clf.fit(x_train, y_train)
score = clf.score(x_test, y_test)
print("Random Forest classification results : %f %%" % (score*100))

# So let's check the algorithm against one of our metasploit generated payload!
file = "C://Users//Ethan//Desktop//msf_reverse.exe"
data = extractor.extract_infos(file)
pe_features = map(lambda x:data[x], features)
```

```
res = clf.predict([pe_features])[0]
if res:
    print "File %s is Malicious!" % file
else:
    print "File %s is Clean!" % file

print "------------------------"
print "Printing PE Features"
print "------------------------"
for f,p in zip(features,pe_features):
    print f,p

''''' Sample output

File C://Users//Ethan//Desktop//msf_reverse.exe is Malicious!
------------------------
Printing PE Features
------------------------
DllCharacteristics 0
Subsystem 2
SizeOfOptionalHeader 224
VersionInformationSize 17
ResourcesMaxEntropy 0
MajorSubsystemVersion 4
ResourcesMinEntropy 0
SectionAlignment 4096
MajorLinkerVersion 6
ResourcesMeanEntropy 0
SectionsMaxEntropy 7.01671734144
Machine 332
SizeOfHeaders 4096
FileAlignment 4096
LoadConfigurationSize 0
'''
```