Spring 4-9-2020

Feature Extraction and Analysis of Binaries for Classification

Micah Flack

Feature Extraction and Analysis of Binaries for Classification

DAKOTA STATE UNIVERSITY

MICAH FLACK, ADVISOR: DR. BRAMWELL BRIZENDINE

DSU SRI Poster

April 9th, 2020

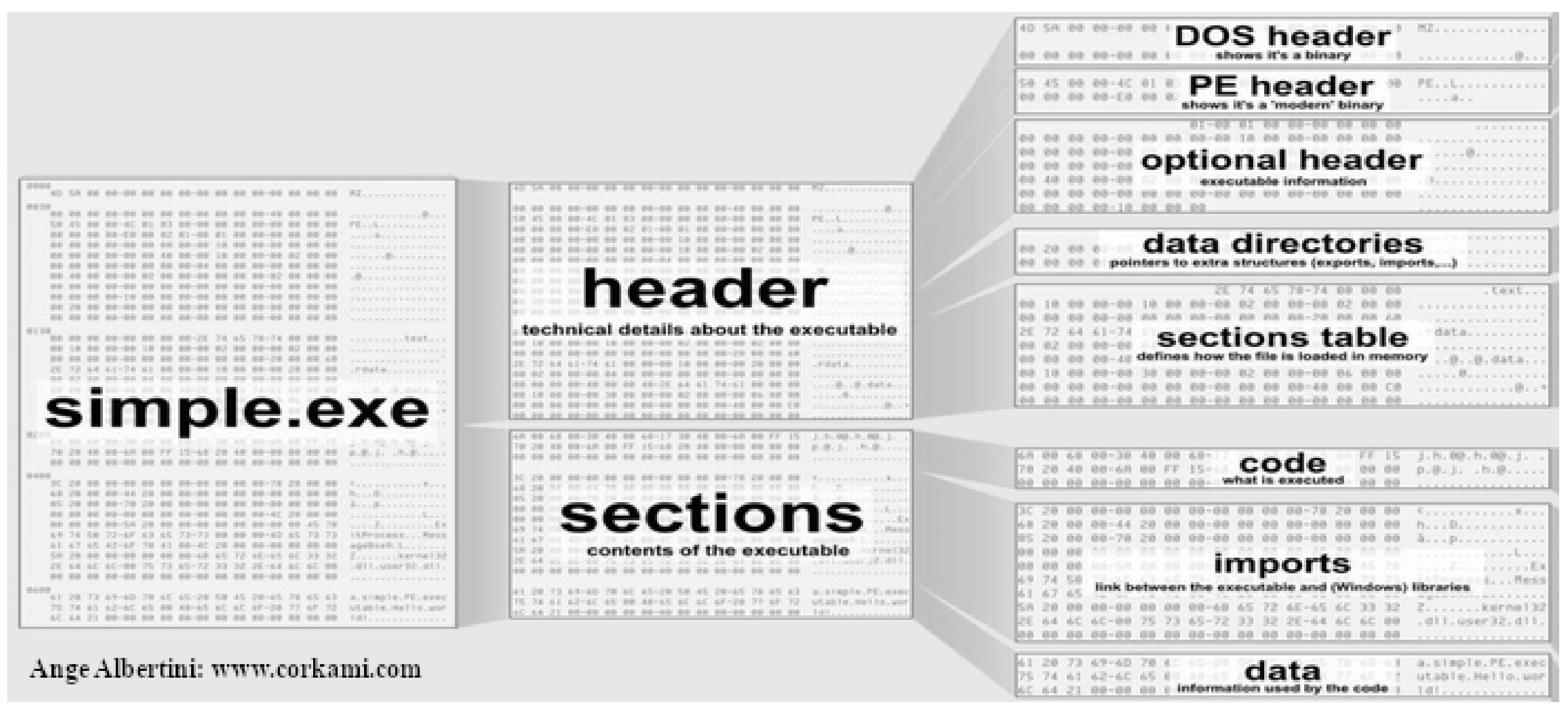
ABSTRACT

The research project, *Feature Extraction and, Analysis of Binaries for Classification*, provides an in-depth examination of the features shared by unlabeled binary samples, for classification into the categories of benign or malicious software using several different methods. Because of the time it takes to manually analyze or reverse engineer binaries to determine their function, the ability to gather features and then instantly classify samples without explicitly programming the solution is incredibly valuable. It is possible to use an online service; however, this is not always viable depending on the sensitivity of the binary. With Python3 and the Pefile library, we can gather the necessary features to begin choosing different classifier models from the Scikit-learn library for machine learning. This all addresses the issue of local automated classification, and we present several different classifier models, datasets and methods that allow for the classification of unknown binaries with a high degree of accuracy for predicting malware and benignware.

BACKGROUND

Traditionally, the analysis of PE32 executables is done through the process of static or dynamic analysis. For static analysis this means using software like IDA or Radare2, interactive dissemblers and debuggers, to read the file and display the contents of their individual structures without executing the program; whereas dynamic analysis requires the analyst to use a debugger or volatility tools to capture artifacts such as live memory, process trees, or file IO during execution of the program. Each of these methods has its own benefits; however, only static analysis was used for feature extraction for ease of use. Later results will demonstrate that solely static analysis is needed for accurate classification.

One way to simplify this format structure is with the following breakdown of a PE32 file, which shows what kind of information can be used:


simple.exe
header
technical details about the executable
sections
contents of the executable
DOS header
shows it's a binary
PE header
shows it's a 'modern' binary
optional header
executable information
data directories
pointers to extra structures (exports, imports,...)
sections table
defines how the file is loaded in memory
code
what is executed
imports
link between the executable and (Windows) libraries
data
information used by the code
Ange Albertini: www.corkami.com

Any of the extracted information is meaningless without a person to study the contents, which is why machine learning methods are implemented. Machine learning is, in essence, the practice of getting computers to act without being explicitly programmed to make those decisions or actions. Classification is a type of machine learning that uses given data points to predict the class called labels or categories. These predictive models are used to approximate a mapping function from input variables to discrete output variables. They are evaluated using different methods like: precision and recall, receiver operating characteristics (ROC) curve, f1 score, or even k-fold validation. By applying these mathematical concepts to malware identification, we can express the relationships between samples as binary classifiers, such as benign

METHODOLOGY

While several different datasets were used over the course of the project, the final results were calculated using a combination of binaries taken from Hybrid Analysis and from VirusShare. The binaries taken from Hybrid Analysis were obtained via the associated API, and were composed of samples that were previously analyzed for users and determined to be malicious and benign. As a form of preprocessing - all samples gathered were verified as PE32 format executable by examining the magic bytes field, the first two bytes of an executable binary, as 'MZ'. Once binaries were obtained, a Python script was utilized to extract a variety of features.

To obtain workable data from these binaries, we created a Python script using the Pefile Python3 library to extract a wealth of features. By feeding each sample as a buffered input we can further parse each section using the same structured format. This includes the size of the imports directory, the sum of imports from all imported DLLs, as well as the imports' ordinals. Similarly, the extraction of any features from the PE32 samples follows the format of calling the Pefile function 'pe' with the appropriate class identifier for the requested field.

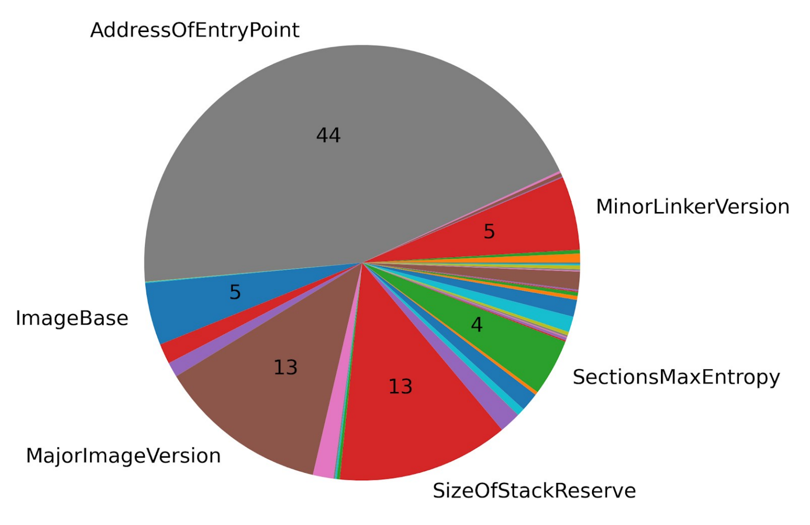
The following are the specific properties extracted from the PE header for training:

Name	MD5	Machine	SizeOfOptionalHeader	Characterisitcs	MajorLinkerVersion
MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	SizeOfUninitializedData	AddressOfEntryPoint	BaseOfCode
BaseOfData	ImageBase	SectionAlignment	FileAlignment	MajorOperatingSystemVersion	MinorOperatingSystemVersion
SizeOfImage	SizeOfHeaders	Checksum	Subsystem	DllCharacteristics	SizeOfStackReserve
SizeOfStackCommit	SizeOfHeapReserve	SizeOfHeapCommit	LoaderFlags	NumberOfRvaAndSizes	SectionsNb
SectionsMeanEntropy	SectionsMinEntropy	SectionsMaxEntropy	SectionsMeanRawsize	SectionsMeanVirtualsize	SectionsMinVirtualsize
SectionsMaxVirtualsize	ImportsNbDLL	ImportsNb	ImportsNbOrdinal	ExportNb	ResourcesNb
ResourcesMeanEntropy	ResourcesMinEntropy	ResourcesMaxEntropy	ResourcesMeanSize	ResourcesMinSize	ResourcesMaxSize
LoadConfigurationSize	VersionInformationSize	Benign			

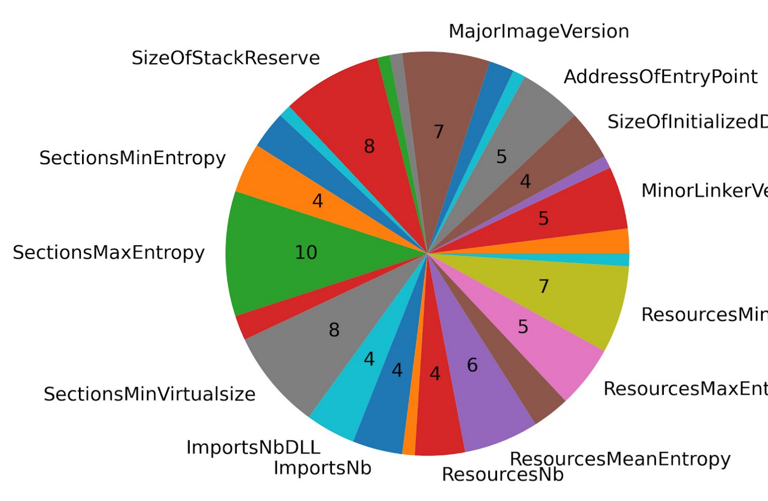
Models used were tuned and parameterized using the default guidelines given by Scikit-learn, an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.

RESULTS & CONCLUSION

Many of the datasets utilized were heavily skewed towards malicious samples. For example, one such dataset contained 28,393 malicious samples, while only containing 966 labeled benign samples. Due to this large imbalance we used undersampling. The data was adjusted prior to division into testing/training sets by taking *min(m,b)* samples from each class, where *m* is the class of samples labeled to be malicious, and *b* is the class of samples labeled to be benign in nature. In general, while this did improve the macro accuracy of several models, its primary purpose was to improve the precision and recall scores of the benign class, as some models' benign data was misclassified as malicious. To compare the effectiveness of undersampling at redistributing the reliance of the models, we produced the following graphs:



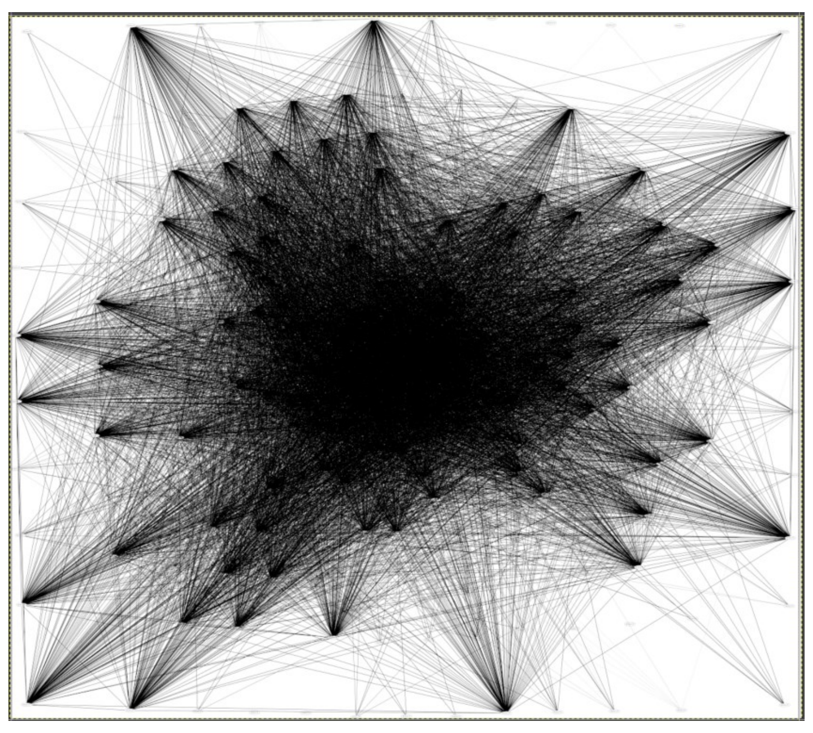
Random Forest feature ranking w/o undersampling



Adaboost feature ranking w/ undersampling

Here it can be seen how imbalanced the models we created were before undersampling the dataset. By evening the samples chosen from both classes, the models we employed were more resilient against inconsistencies of heavily relied upon features like AddressOfEntryPoint.

The following graph was an attempt to demonstrate an attribution of different families of backdoor malware samples, by creating one node for each sample and edges between those nodes, representing a relationship between the two nodes. In this instance, the relationships were determined by scoring the jaccard distance similarities, with a threshold of 0.8, of the extracted import strings, although the clusters are convoluted and could have been represented better with a technique that weighs the edges between nodes like, KMeans with ChineseWhispers clustering. The main takeaway is that anomalous samples with fewer common relations are shown grouped at an equal distance from the center.



The following chart shows the overall performance evaluation for the models used:

Name	Macro Accuracy	Malicious Precision	Malicious Recall	Malware F1	Benign Precision	Benign Recall	Benign F1
HistGradientBooster	100%	100%	100%	100%	99%	100%	100%
HistGradientBooster - w/ Undersampling	99%	100%	98%	100%	98%	100%	96%
AdaBoost	100%	100%	100%	100%	99%	100%	100%
AdaBoost - w/ Undersampling	99%	100%	99%	X	99%	100%	X
RandomForest	100%	100%	100%	100%	100%	99%	100%
RandomForest - w/ Undersampling	100%	100%	99%	X	99%	100%	X
LogisticRegression	96%	99%	100%	99%	85%	55%	67%
LogisticRegression - w/ Undersampling	98%	100%	98%	99%	98%	100%	99%

In conclusion, the research demonstrates that the combination of static analysis, for feature extraction, with basic supervised machine learning methods is able to achieve a macro accuracy.

FUTURE WORK

The majority of this research, however, was hindered by the lack of verifiable benign (negative) software samples. Given the focus for analysis and identification of malware (positive), that difference in availability is almost incomparable because the lack of data is insufficient to correctly form the right bias when training models; this is often referred to as overfitting and underfitting, and there are many methods to combat this issue.

One example would be to use Positive Unlabeled (PU) learning. Given a set of samples of a particular class P, called the positive class, and a set of unlabeled samples U, which contains both class P and non-class P, called the negative class instances, the goal is to build a binary classifier to classify the test set T into two classes, positive and negative, where T can be U. In this case though, the positive examples are malicious and the set of unlabeled samples can contain both malicious and benign samples. This would allow future research to be completed without the limits caused by a lack of verified, benign samples previously needed by models like, random forest or logistic regression with undersampling.