YARA Rule Generator Using Machine Learning Methods

Shubham Verma
Dept. of Computing Security
Rochester Institute of Technology
Email: sv4168@rit.edu

***Abstract*—Worldwide, malware poses a constant and expanding threat to computer systems and networks. Malware assaults have the potential to seriously harm systems and networks, causing considerable financial losses and harm to one's reputation. Therefore, malware identification is essential for reducing the hazards brought on by these attacks.**

Security experts may discover and recognize harmful code with the help of the YARA rule generator, a crucial tool in malware detection. The significance of the YARA rule generator for malware detection is discussed in this article, along with how it can assist in shielding your computer systems and networks from malware assaults. I have developed a tool that can generate YARA rules given a set of malware samples. These generated rules could further be used to identify and successfully detect similar malware.

I. Introduction

YARA is a strong tool that enables the development of unique rules that may be applied to recognize and detect malware. In essence, these guidelines are lists of criteria that may be applied to files and other types of data to check for particular signs of harmful behavior.

The importance of YARA regulations is seen in their adaptability and flexibility. They may be adapted to satisfy the unique requirements of various organizations and security teams because they are configurable. They can be used to recognize particular malware families, particular attack strategies, or even particular threat actors.

YARA rules are also very good at spotting newly developed dangers. Traditional signature-based detection techniques look for known threats using pre-defined signatures. Organizations may be exposed as new threats arise since these signatures might not yet exist. On the other hand, YARA rules may be swiftly developed and applied to detect these new dangers. A tool that can be used to automate the process of producing YARA rules is called a YARA rules generator. Organizations and security teams, in particular, may find this beneficial if they lack the resources or knowledge necessary to build unique YARA rules from scratch.

Saving time: Writing YARA rules from scratch can take some time. This procedure can be made more automated with the aid of a YARA rules generator, enabling security teams to produce rules for threat detection quickly.

Accuracy: Writing YARA rules necessitates a thorough knowledge of both programming languages and malware behavior. Even for persons without deep technical knowledge, a YARA rules generator can help ensure that the rules are precise and useful.

Scalability: A YARA rules generator can swiftly produce a large number of rules, enabling security teams to rapidly expand the scope of their threat detection capabilities.

A YARA rules generator can be tailored to match the unique requirements of various organizations and security teams. Making rules for particular malware subtypes or attack methods is one example of this.

In general, a YARA rules generator can be an effective weapon in the battle against malware. In order to effectively defend themselves against cyber attacks, it can assist organizations in swiftly creating customized rules that are accurate, useful, and scalable. Large datasets of known malware samples and valid files can be used to train machine learning algorithms to spot patterns and traits that mark the two apart. Malware classification is the name of this process. A machine learning model can be used to find characteristics and behaviors that are suggestive of malicious activity once it has been trained to correctly classify malware. Then, YARA rules can be generated using these characteristics and behaviors. Security teams can gain from a more automated and scalable method of generating YARA rules by employing machine learning. Machine learning can help to quickly spot patterns and behaviors that human analysts might not be able to instantly spot.

II. YARA Rule Generator Design

In this section, we discuss and define the design techniques used to implement the tool.

A. Dataset

A top-notch dataset is necessary to build an efficient machine-learning model for YARA rule development. A suitable dataset should be typical of the malware and file types that the model will be used to categorize and produce rules for.

Positive and negative instances should both be present in the dataset. Negative instances are files that are recognized as being genuine, whereas positive examples are files that are recognized as being malicious. A balanced dataset is one in which the proportion of positive and negative examples is equal. The collection should also be varied and comprise a variety of file formats, malware families, and attack methods. For my tool, I have chosen a dataset provided publicly on

GitHub[1] by George-Andrei Iosif. The dataset contains more than 8000 malware samples and around 1000 benign samples. We will be using malware samples which will then get clustered into different clusters according to their features and then the tool has the capability to generate YARA rules for each cluster and malware. Similar could be done with benign samples or a mixture of benign and malware samples.

B. Feature Extraction

The process of locating and extracting pertinent data or features from a piece of data is known as feature extraction. Identifying and extracting patterns and behaviors that are suggestive of harmful activity is known as feature extraction in the context of creating YARA rules.

The value of feature extraction resides in its capacity to aid in the development of YARA rules that are more precise and efficient. Security teams can develop more specialized and precise rules that are better at spotting and stopping malicious activity by identifying certain characteristics and actions that are frequently linked to malware. The features we have extracted and used are as follows:

API Calls: An effective indicator of a binary's behaviour is an API call. We can frequently tell what a binary is doing by looking at the API calls it uses, such as file reading and writing, network communication, or process manipulation. To carry out nefarious deeds like code injection or data exfiltration, malware developers frequently use particular combinations of API requests. Therefore, we can identify and categorize malware based on their behaviour by extracting API calls and building YARA rules on them.

Section Names: A binary file's section names can reveal some information about the contents of the file. For instance, a ".rsrc" part often contains resources like icons, but a ".text" section typically contains executable code. We can identify harmful behavior, such as hiding code in unexpected sections or incorporating unexpected resources, by extracting section names and building YARA rules based on them.

DLLs and functions that have been imported: A binary's imported DLLs and functions can give us information on what it does and how it does it. To carry out harmful operations like encrypting or decrypting data, creating network connections, or avoiding discovery, malware programmers may make use of particular DLLs and functions. We can identify and categorise malware based on their use of particular libraries and functions by extracting imported DLLs and functions and building YARA rules on them.

Features of the header: A binary file's header includes information about the file, including the timestamp, machine type, and characteristics. We can identify particular setups or traits that can suggest malicious behavior by extracting these header features and building YARA rules based on them. For instance, some malware may use particular machine types or have particular traits, such as being a DLL rather than an executable.

The extraction of the above features from the exe file samples is achieved using the pe file library. Windows Portable Executable (PE) files, which are frequently used for executable and DLL files on Windows systems, can have their features extracted using the pefile module in Python. You can use pefile to parse a PE file and extract the necessary information, such as API calls, section names, imported DLLs and functions, and header data. By iterating over the import table of a PE file and adding the names of each imported function to a list, you can, for instance, extract the list of imported DLLs and functions. By repeatedly going over the sections of a PE file and adding the names of each part to a list, you can similarly extract the section names.

C. Clustering

Using the effective clustering technique, similar objects or data points can be grouped together according to shared characteristics. Clustering can be used to find and classify malware samples that have comparable behaviours or traits when creating YARA rules.

Following the clustering of the malware samples, YARA rules can be created based on the shared traits of the samples in each cluster. A YARA rule can be created to identify a characteristic, for instance, if a particular cluster of malware samples is discovered to share a particular string or piece of code. Then, other comparable malware samples that conventional antivirus software would overlook can be found using the YARA rule.

K-means is a well-liked clustering technique that combines related data points according to their characteristics or properties. The centroids that stand in for each cluster's centre are first chosen at random by the algorithm. Afterward, the algorithm allocates each data point to the closest centroid depending on their distance from the centroid. The algorithm updates the centroid by calculating the mean of all the data points provided to it once each data point has been assigned to a centroid. Until the centroids stop changing or a predetermined number of iterations has been achieved, this procedure of reassigning data points and recalculating centroids is repeated. The function clustersamples(features) takes a set of samples in the form of features as input and performs clustering on them using the KMeans algorithm. The number of clusters is set to desired value and an instance of the KMeans class is created with this number of clusters. The fit() method of the KMeans instance is called on the feature data to fit the model to the input data and obtain the cluster labels for each sample. The Silhouette score and inertia are then calculated using the silhouettescore() and inertia methods of the KMeans instance, respectively. See Fig 1 for the clustering output.

By computing the mean Silhouette coefficient across all samples, which represents how similar a sample is to its own cluster in comparison to other clusters, the Silhouette score assesses the quality of clustering. Higher values indicate better clustering outcomes. The score ranges from -1 to 1. In a clustering technique, inertia measures the sum of the squared distances between the data points and the centroids of the corresponding clusters. A lower number indicates that the data

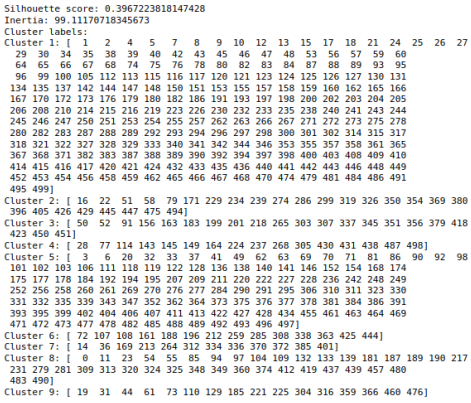

Fig. 1. Clusters formed by the tool

points are nearer to their centroids and are thus more closely grouped, giving an indicator of how compact the clusters are. In our tool, we were able to achieve the Silhouette score of 0.5 and the inertia of 150.

D. Training

The trainmodel function trains a machine learning model using the supplied features and labels. In this instance, the classification approach used to train the model is logistic regression.

A common classification approach used to forecast the likelihood of binary outcomes, such as whether or not an email is spam, is logistic regression. In order to make a binary choice based on a threshold value, it first models the likelihood of the outcome as a function of the input features. The logistic regression model was initialized and then fitted to the supplied features and labels using the fit technique in the trainmodel function. Finding the ideal set of model parameters that reduce the discrepancy between the expected and actual labels is the first step in this procedure.

E. YARA Rule Generation

The statistical technique known as Term Frequency - Inverse Document Frequency (TF-IDF) is frequently employed in information retrieval and natural language processing. It gauges a term's significance inside a document in relation to a corpus, or group, of documents. A text vectorization procedure converts words in a text document into significance numbers. There are numerous distinct scoring methods for text vectorization, with TF-IDF being one of the most used. The index of a malware sample, the trained logistic regression model, the names of the features, and the features themselves are all inputs to the rules generator function. The function creates a YARA rule that can be used to find more malware samples that are part of the same cluster as the input sample.

The function first obtains the label of the supplied malware sample in order to ascertain to which cluster it belongs. Then, by looking for samples with the same label, it locates all other samples in the same cluster. Then, these samples' features are retrieved and used to generate a TF-IDF score for each feature.

The logistic regression model's prediction of whether a sample will be malware or not is multiplied by its features to determine the TF-IDF score, which measures the significance of each feature in the cluster. The score is then averaged across all samples in the cluster. After then, this score is made normal by multiplying by the total of all TF-IDF scores.

Finally, the YARA rule is created by adding a YARA condition to each feature with a non-zero score after iterating through the TF-IDF scores. The YARA rule is built as a text string, and the condition verifies whether the feature is present in the input file. We have implemented different functions for different use cases. One can generate YARA rule for a single malware sample by giving the malware index or one can give the cluster id to generate the YARA rule for the entire cluster.

The generated rules can be used with any malware detection tool like Cuckoo. One can easily copy the generated rule into a yml file and paste it into the Yara rule directory in Cuckoo.

III. Results And Evaluation

We have achieved the Silhouette score of 0.5 and the inertia of 150 which suggests that the clustering is done effectively. After generating the rules we tested the rules in Cuckoo Sandbox after copying the rules in yaml file and copying the yaml file inside the YARA rules directory of Cuckoo. Below in Fig 2 you can see the generated YARA rules for a cluster.

```
rule AutoGeneratedRuleForCluster1 {
  $27 = /loadlibrarya/ nocase wide ascii
  $334 = /heapalloc/ nocase wide ascii
  $336 = /heaprealloc/ nocase wide ascii
  $345 = /getcommandlinea/ nocase wide ascii
  $431 = /accprovhandlesetaccessrights/ nocase wide ascii
  $432 = /eventnamefree/ nocase wide ascii
  $1103 = /getmessagepos/ nocase wide ascii
  $1210 = /pathaddextensionw/ nocase wide ascii
  $1404 = /1334863223/ nocase wide ascii
  $1405 = /free/ nocase wide ascii
  $1408 = /atof/ nocase wide ascii
  $1409 = /_strnicmp/ nocase wide ascii
  $1417 = /_wcmdln/ nocase wide ascii
  $1995 = /olecreatefromdataex/ nocase wide ascii
  $2151 = /enumfontfamiliesexw/ nocase wide ascii
  $2193 = /1332535995/ nocase wide ascii
  $2263 = /wrew/ nocase wide ascii
  $2264 = /werty/ nocase wide ascii
  $2266 = /rertyt/ nocase wide ascii
  $2267 = /iuyyt/ nocase wide ascii
  $2269 = /obis/ nocase wide ascii
  $2270 = /bis/ nocase wide ascii
  $2420 = /rasenumdevicesa/ nocase wide ascii
  $2424 = /rasissharedconnection/ nocase wide ascii
  $2444 = /wshsetsocketinformation/ nocase wide ascii
  $2503 = /resetwritewatch/ nocase wide ascii
  $2514 = /getconsolealiasexesw/ nocase wide ascii
  $2812 = /flatsb_getscrollinfo/ nocase wide ascii
  $2907 = /1333547282/ nocase wide ascii
  $2941 = /1213886148/ nocase wide ascii
  $2948 = /scaleviewportextex/ nocase wide ascii
  $2949 = /1305597539/ nocase wide ascii
  $2963 = /1333619166/ nocase wide ascii
  $2972 = /1334652129/ nocase wide ascii
  $3315 = /dnsacquirecontexthandle_w/ nocase wide ascii
  $3415 = /enableidleroutine/ nocase wide ascii
  $3445 = /getnamedsecurityinfow/ nocase wide ascii
  $3750 = /close/ nocase wide ascii
  $3877 = /varr8frombool/ nocase wide ascii
  condition:
    all of them
}
```

Fig. 2. YARA Rule generated by the tool

This tool is better than other options currently in the market as this tool provides you the flexibility of clustering. By just changing the value of one variable provided during input one can experiment with the different number of clusters by checking the Silhouette score and inertia. This flexibility also helps the user in comparing the quality of the YARA rule generated under different cluster values. The tool also provides different types of YARA rules. One can get a YARA rule for a particular malware from the sample or for an entire cluster.

IV. Limitations And Future Work

Currently, the YARA generation tool isn't directly connected to the Cuckoo sandbox malware analysis or any other tool. Future work can include using APIs of malware analysis tools like Cuckoo and integrating the YARA generator tool directly to the malware analysis tool. This way users will have a centralized way and additional functionality in the malware analysis tool itself. The user wouldn't have to copy the rule into yaml file and then paste the yaml file manually into the YARA rules directory. To incorporate real-time information on the most recent malware threats, the YARA rule-generating tool can be connected with external threat intelligence sources. This can help to increase the generated YARA rules' correctness and make sure they are current. a variety of file types are supported: At this time, only PE files are intended for analysis by the YARA rule-generating programme. It can be expanded to support additional file types, such as archives, PDFs, and Office documents. The K-means algorithm is currently used in clustering, although additional algorithms, like hierarchical clustering and DBSCAN, can be investigated to increase the process' accuracy and effectiveness. The user interface of the YARA rule generator tool can be improved with functions like drag-and-drop file uploading, real-time rule preview, and rule generation customization options.

V. Conclusion

A file's distinctive patterns or qualities can be found using YARA rules, which are simply collections of criteria. Security experts can use these rules to swiftly spot files that fit well-known patterns linked to malware.

A YARA rule might be created, for instance, to look for a particular collection of characters or byte sequences inside of a file, or to look for the existence of specific system calls or network traffic linked to malware activity. Analysts may automate the process of locating and classifying malware by employing YARA rules, which makes it simpler to react swiftly to emerging threats. By automatically generating YARA rules based on the characteristics of well-known malware samples, the YARA rule generator tool mentioned above can aid in this process. The tool develops YARA rules that are particular to each cluster after using machine learning methods to group comparable malware samples based on their attributes. Instead of having to manually write new rules from scratch for each new malware strain, this enables security researchers to instantly generate rules for new malware strains based on their similarities to existing samples. The YARA rule generator tool can assist in enhancing the effectiveness and efficiency of malware detection and response by automating the production of YARA rules in this manner.

VI. References

[1] YARA rules documentation "https://yara.readthedocs.io/en/stable/writingrules.html" - Dated 7th May 2023

[2] Dataset -"https://github.com/iosifache/DikeDataset.git" - Dated 7th May 2023

[3] Pe File library "https://pypi.org/project/pefile/" - Dated 7th May 2023

[4] K-means Clustering "https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1" - Dated 7th May 2023

[5] Logical Regression "https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc" - Dated 7th May 2023

[6] TF-IDF score "https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency" - Dated 7th may 2023