A CROWDSOURCING-BASED ANALYTICAL ENGINE FOR VIRUS AND MALWARE DETECTION USING ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Zonglin Zhang[1], Marisabel Chang[2]

[1]Portola High School, 1001 Cadence, Irvine, CA 92618
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

In recent years, cybersecurity has grown increasingly salient in people's lives [8]. With the spread of various new malware, the security risks of executable network installation packages are dramatically increasing, so problems persist, rising with the growth of web users. This research work, aimed at a Crowdsourcing-based Analytical Engine for Virus and Malware Detection, prevents malware by examining MS Windows Portable Executable (PE) headers. YARA, a database from Kaggle, and data extracted from actual malware files were combined to create a final dataset [9]. Comparing each section of the PE header to improve the detection accuracy, the final absolute accuracy is between 98% and 99%, and the front end displays the final prediction results through PythonGUI.

KEYWORDS

AI, Machine learning, Cybersecurity

1. INTRODUCTION

Computers, a cross-generation product, have brought countless people a convenient life. Still, when individuals use their computers to operate millions of files, they cannot identify the security by the file name. According to statistics provided by the AV-Test Institute, the types of malware worldwide number far more than one billion. At least 5 billion computers are attacked by malware yearly; some serious ones can severely cause system crashes. When malware is running on a computer, it destroys system data indefinitely. Even if the malware is deleted, permanent damage has already occurred to the computer, including but not limited to crashes, freezes, data loss, and file corruption. Therefore, unrestricted malware will also cause serious losses to network assets. There are also many malicious links for one to open and then download and run. I have also encountered the same thing in my experience. After running, the malicious code in the file destroys the computer system, a key reason why I conduct this research [10]. Over time, the types of malware will only increase, diversifying the types of hazards as well. There is no doubt that the virus will bring permanent damage to your computer.

There are also many other ways to detect viruses. One method was proposed by Rabia Tahir, in which he created a virtual machine [5]. Although you can run files in the virtual machine to test security without posing a risk to the original machine, this method takes a long time to test the files. It would generally involve having to run each file individually to see whether it would .

David C. Wyld et al. (Eds): SOEN, SIPP, PDTCA, ITE, CCSIT, NLPCL, DaKM, BIGML, AISC -2023
pp. 219-226, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.131219

Occasionally, the virtual machine fails or even crashes when attempting to login and operate the virtual machine as a result of faulty. There is another method similar to mine, which also applies machine learning and PE Header, but this method only extracts three different sections, so the prediction is not very accurate [3].

In this research, I mainly use machine learning. First, I extracted the PE header of 5212 files from the Kaggle database as my training data and added malware data from the YARA database to populate my entire database [4]. Then, I imported the file to test it, extracted the 61 different PE Headers in the file, and tested it with my database to predict whether the file is malware. In addition, I've created a framework with PythonGUI in conjunction with this malware detection, an interface using simple buttons to import files or folders. The final forecast results are displayed on the second page of the frame. This approach has several advantages, the first being that it saves time testing files compared to creating a virtual machine. The second advantage is that you can test whether the file has a virus without running the file, which achieves the function of predicting harm in advance. I believe my method is simpler and more convenient. For a virtual machine to test a 1MB file, it takes at least 3 minutes; in addition to the file size, there may be unavoidable issues that come with running potentially harmful files. But for the same file of the same size, my method only takes 5 seconds to predict the result.

To test my results for this project, I tested 15 different models; a few of the models that were tested include GaussianNB, AdaBoostClassifier, LogisticRegression, SVM, DecisionTreeClassifier, and LinearDiscriminantAnalysis. As Python has an active community with an abundance of different packages and libraries to choose from, the models were relatively easy to find and implement into my code for testing. For the experiment, the models would go through 100 different files. Of these files, 50 of them are completely safe to open and leave on the computer, while the other 50 are known malware files. With each model, the total number of files that were correctly determined to be safe or malware would be used as the percentage to be recorded in a table. With this experiment, the goal is to see whether the analytical engine can properly incorporate malware detection models within itself and be a useful tool for people to use.

This paper is guided by the following structure: Section 2 discusses the challenges that I encountered when an insufficient dataset caused faulty reports; Section 3 is an explanation of the method I propose to detect malware, and how I created the framework of an app that makes this method accessible to a wider audience; Section 4 experiments is how I chose the final training model that yielded highest malware detection accuracy. In Section 5, I compare my method with those of others, and how my solution differs from the ones pre-existing. In Section 6, a conclusion will summarize the work, look back at what areas the application is lacking in, and provide possible paths for future improvement.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. The malware database found in Kaggle only had data of 5212 files

Several complex challenges were experienced in building this program, firstly because the malware database found in Kaggle only had data of 5212 files. The correct rate in the last step of prediction did not reach an accuracy of 100%, but a rate of 98-99%, so the chance of false results was still as high as 1-2%. False results appear once after every 10-20 files tested. The reason for this is that the PE header of many unknown files may not match the one in the database, resulting in the AI not having enough data to predict the security of the files. Hence, the lack of information in the tracing database became my first challenge in creating this project.

2.2. The constant evolution of viruses

The second challenge is the constant evolution of viruses and their subsequent rapid mutations [11]. If the database is not updated in real time, the accuracy of the prediction will be drastically reduced, because similar virus files may be similar to ordinary files in predicting false results. Therefore, if the database cannot be updated every second, the usefulness of the project will slowly decline over time. The similarity of files also requires data from a large number of files to discriminate and avoid incorrect results in many tests. There are countless types of viruses, particularly notorious ones being Trojan horses and worms. The speed of evolution and spread of these two viruses far exceed the speed of updating my database; this concern is further complicated by the absolute advantage of the variety possessed by virus types.

2.3. No framework for users to use

The third challenge is that there is no framework for users to use my malware detection application. The project only uses Python to make scripts and does not create a web page or app to dematerialize. If this project is to be brought to the public in the future, an entity framework is essential, but Python has a disadvantage. Unlike the Swift language, Python cannot be directly translated into software because Python is mainly used in back-end development [12]. Therefore, in the experience of presenting the entity, using the Python language requires a more intricate software development method. This is also the difficulty and inconvenience of this project when presenting it to the public. To solve this, the PythonGUI framework was used to create a user interface for users of the application to interact with.

3. SOLUTION

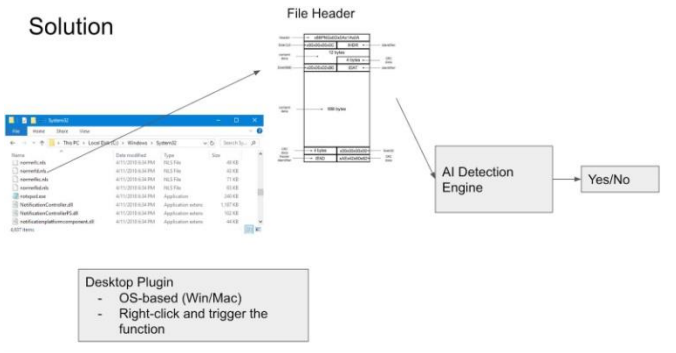

Figure 1. Overview of the solution

PE header is an executable file for 32-bit and 64-bit versions of Windows operating systems, where the file seals the executable information on the computer; for example, you download an executable file with the extension .exe, .ax, .acm, etc. A user can successfully run the file when you download it and double-click to open it. FIrst, I find a data extraction from Kaggle with information from 5212 files, which was used for training. The hexadecimal in each file was converted to a number. Each PE header section should be equal to the header of the database file to avoid iterable expectations [2]. I chose GaussianNB, AdaBoostClassifier, LogisticRegression, SVM, DecisionTreeClassifier, and LinearDiscriminantAnalysis. After saving the model, the converted number can be compared with the information in the database. Finally, machine learning can predict the final result.

3.1. Establishing the UI frontend

Figure 2. The malware detector's user interface

The UI is created using PythonGUI, a framework from Python. The interface features a simple title that states "Malware Detector", followed by three buttons that represent the different options that a user can choose from to detect malware. The options are as follows:
Open File: select a specific file
Open Folder: select a single folder for the files within itScan PC: enter all .exe files on the computer

3.2. Database and Training

I used the database from Kaggle and combined it with YARA (a tool mainly used for malware research and detection). The final database formed a table with the PE information of 5525 different files to have the model distinguish between benign "0" and malware files "1". The information of these tested files is stored in the backend. I first chose 6 different models: GaussianNB, AdaBoostClassifier, LogisticRegression, SVM, DecisionTreeClassifier, and LinearDiscriminantAnalysis. Finally, I chose the AdaBoostClassifier with an F1 score of 99.3% as my model for training with the database.

3.3. Extracting the PE Headers

All the different headers are extracted from the user input's files. I removed most of the DOS section header because the headers in this section are the same value in all the .exe files that exist, so this makes the DOS header an abandoned PE header.

3.4. Prediction and UI Features

Using the trained model, the final prediction accuracy is between 98-99.3%. The results of the tests are all reproduced in the UI, each with the pattern displaying "The file [file name] is [Malware or Benign]". If the user clicks on the small plus sign below the UI to expand the second page, the contents include the history of detection and the trash button to removemalware files.

4. EXPERIMENT

Six models were trained with a database to determine which would be the best-performing model to use in the application; these models were GaussianNB, AdaBoostClassifier, LogisticRegression, SVM, DecisionTreeClassifier, and LinearDiscriminantAnalysis [13]. The database was created by combining a pre existing database of 5212 files from Kaggle with malware files from the YARA database. An additional database made of 100 files consisting of 50 harmless and 50 malware files was also created specifically to be tested in the application using the option to scan a folder. With both databases, there is a large enough sample size to account for any variability.

Table 1. Model type, F-1 score, and accuracy

Model Type	F-1 Score	Accuracy in Application
AdaBoostClassifier	99.3%	98%
LogisticRegression	98.2%	97%
GaussianNB	98.6%	95%
SVM	97.3%	95%
DecisionTreeClassifier	97.6%	93%
LinearDiscriminantAnalysis	96.4%	92%

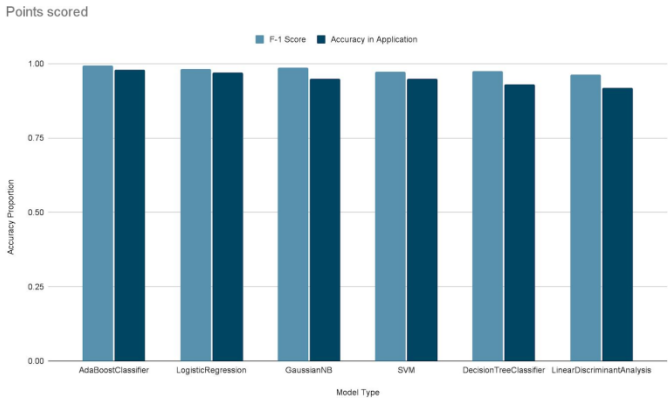

Figure 3. Points scored

The results of the tests indicate that the AdaBoost Classifier has the highest accuracy, both with the F-1 score from the larger dataset tested in raw Python code and the proportion of files correctly identified from the smaller dataset tested through the application. Every model had a fairly high rate of accuracy, as all of the models were over 90% accurate with identifying files across both datasets. An interesting observation to make based on the graph and table is that the models all perform better on the dataset composed of files from Kaggle and YARA than they do with the dataset in a folder with 50 harmless and 50 malware files. A possible explanation for this is that the second dataset is much smaller than the first dataset and contains particularly difficult files to correctly identify; this could reinforce the need for a large sample size to reduce variability.

According to the results of the experiment, AdaBoostClassifier was the most accurate one when testing it on the dataset with 50 malware and 50 benign files. This meets expectations, as this is the same model that performed the best when training with the larger dataset. As AdaBoostClassifier appears to be the most consistently accurate model, it would be the most suited to be incorporated into the full-release version of the application. Since the issue of which model to use is resolved, other aspects of the application can be focused on in the future.

With the completion of the experiment, the application is confirmed to function properly and provide the correct accuracy. Although the application had been tested before this experiment during its development, it was done using much smaller samples that do not necessarily represent whether the application is outputting correct information. After testing the dataset, it roughly matches up with the F-1 scores when training the data for each model.

5. RELATED WORK

With the growing amount of malware that has been developed, it is essential that computer science researchers develop countermeasures to defend computers and networks from them. A related work created by Bazrafshan et al. analyzed several malware detection methods and described the benefits and downsides of each one [6]. The related work is similar to this work in that the central topic involves malware detection and analyzing different methods to do so. The related work emphasizes methods such as API Calls, N-Grams, and OpCodes that are regarded as state-of-the-art malware detection methods. On the other hand, this work features six other malware detection classifiers and places a greater emphasis on machine learning.

New generation malware is using more clever and invasive methods of taking over the computer, which can make it harder to detect and more difficult to rid a computer or network of. Malware detection is an important countermeasure to ensure that people's computers and data stay safe. Another related work presents a systematic overview of malware detection approaches forfurther studies [1]. The related work and this work share the similarity of focusing on malware and how to prevent it from damaging people's computers and data. However, while the related work primarily provides a look at how malware detection and prevention can potentially be improved, this work is based on the testing of applications and how well they do.

A third related work tackles the same issue of malware detection but does so specifically for Android applications. Android malware seems to grow in number just as Android applications do, and reviews currently exist regarding several approaches to detecting Android malware with machine learning. The work seeks to expand on these reviews by exploring other aspects of these approaches [7]. This work and the related work both have the topic of malware detection in common, but the related work is targeted specifically toward malware that infects Android devices while this work tests the ability of malware detection methods to detect Windows malware from files in particular.

6. CONCLUSIONS

A tool to detect malware was created in the form of an application. Python was used as the programming language for the back end of the application, and PythonGUI is a framework used to create a user interface for the front end of the application. The application features three possible buttons to choose from, which each has its own function. The first button will prompt the user to choose an individual file from the PC to check whether it is malware or not. The second button will scan the entire PC for all of its .exe files and determine whether any of the selected files are malware. The last button will prompt the user to select a folder to see whether

the files within it are malware. An experiment was performed using the application as aninterface to test six different models. There were two datasets for this experiment. The first dataset was created by combining a dataset of 5212 files from Kaggle with the YARA dataset to have a combination of harmless files and malware files. After testing each model through raw Python code, AdaBoostClassifier obtained the highest F-1 score [15]. The second dataset was tested as a scanned folder of 100 files; within the folder, 50 files are harmless and the other 50 files are known malware. The purpose of this experiment was to determine which model would be the best one to use in the tool. Among the six models that were tested, the results determined that the AdaBoostClassifier had the highest accuracy [14]. Therefore, this was chosen to be used in the model that was used in the application. The experiment also serves to prove that the application is a suitable interface for the application.

A possible limitation of the application is the lack of features and design in the application. While the application is arguably only for one purpose and does not necessarily need new features, a single-feature tool that does not offer anything unique or special may cause people to eventually move to a different tool that is more versatile and feature-rich. The simple design may make the application very intuitive and easy for newcomers to use, but the design may give off the impression of a very unpolished and unrefined application, which could discourage many potential users from giving the application a chance.

If I continue to work on this malware detection engine in the future, I plan to improve the design of the application. Rather than a white plain background, a light-colored pattern for the background could be a relatively quick design change that will give the application much more polish. Furthermore, adding even more models to the application and directly allowing the user to choose which model to use would be another way of potentially improving the application.

REFERENCES

[1] Aslan, Ömer Aslan, and Refik Samet. "A comprehensive review on malware detection approaches." IEEE Access 8 (2020): 6249-6271.
[2] Dawahdeh, Ziad E., Shahrul N. Yaakob, and Ali Makki Sagheer. "Modified ElGamal elliptic curve cryptosystem using hexadecimal representation." Indian Journal of Science and Technology 8.15 (2015): 1-8.
[3] Raff, Edward, Jared Sylvester, and Charles Nicholas. "Learning the pe header, malware detection with minimal domain knowledge." Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security. 2017.
[4] Dadi, Temesgen Hailemariam, et al. "DREAM-Yara: an exact read mapper for very large databases with short update time." Bioinformatics 34.17 (2018): i766-i772.
[5] Tickoo, Omesh, et al. "Modeling virtual machine performance: challenges and approaches." ACM SIGMETRICS Performance Evaluation Review 37.3 (2010): 55-60.
[6] Bazrafshan, Zahra, et al. "A survey on heuristic malware detection techniques." The 5th Conference on Information and Knowledge Technology. IEEE, 2013.
[7] Liu, Kaijun, et al. "A review of android malware detection approaches based on machine learning." IEEE Access 8 (2020): 124579-124607.
[8] Craigen, Dan, Nadia Diakun-Thibault, and Randy Purse. "Defining cybersecurity." Technology Innovation Management Review 4.10 (2014)..
[9] Bojer, Casper Solheim, and Jens Peder Meldgaard. "Kaggle forecasting competitions: An overlooked learning opportunity." International Journal of Forecasting 37.2 (2021): 587-603.
[10] Bayer, Ulrich, et al. "Dynamic analysis of malicious code." Journal in Computer Virology 2 (2006): 67-77.
[11] Grompe, Markus. "The rapid detection of unknown mutations in nucleic acids." Nature genetics 5.2 (1993): 111-117.

[12] Wilde, Michael, et al. "Swift: A language for distributed parallel scripting." Parallel Computing 37.9 (2011): 633-652.
[13] Pushpakumar, R., et al. "A Novel Approach to Identify Dynamic Deficiency in Cell using Gaussian NB Classifier." 2022 7th International Conference on Communication and Electronics Systems (ICCES). IEEE, 2022.
[14] An, Tae-Ki, and Moon-Hyun Kim. "A new diverse AdaBoost classifier." 2010 International conference on artificial intelligence and computational intelligence. Vol. 1. IEEE, 2010.
[15] Takahashi, Kanae, et al. "Confidence interval for micro-averaged F 1 and macro-averaged F 1 scores." Applied Intelligence 52.5 (2022): 4961-4972.