Project Specifications Report

CleaverWall

Arda Barış Örtlek

Yekta Seçkin Satır

Ali Emre Aydoğmuş

Onur Korkmaz

Selahattin Cem Öztürk

Supervisor: Özcan Öztürk

Jury Members: Erhan Dolak, Tağmaç Topal

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

Malware detection is a trending topic since the 1980s and it is becoming more significant due to the immense increase in malware programs. Even though there is use of different approaches, most of the open-source anti-malware programs rely upon signature based methods. CleaverWall optimizes this method commonly used in the industry by developing an application that totally depends on machine learning to eliminate the disadvantages of signature based detection methods. The application has three stages, the extraction of features to put into the model, the classification using a machine learning model, and the operations after the detection of malware. We are planning to produce two versions of the products, the first will be a web service and the second will be a Windows application.

The details of CleaverWall, its constraints, professional and ethical issues, and functional and non-functional requirements will be explained in the rest of this report.

1.1 Description

CleaverWall is an anti-malware mechanism to detect whether a portable executable is malicious, if so, to classify the malware type. For the classification process, a set of malware classifiers trained with various machine learning and deep learning techniques will be used. Static and dynamic analysis will be conducted to create different feature vectors for the models. Those different classifiers will work collaboratively to decrease the false positive occurrences.

In static analysis, we have different approaches. The first approach consists of disassembling a portable executable and collecting information from that .asm file. Operation codes, registers, symbols, sections, miscellaneous and Windows API calls will be analyzed to create features [1], [2]. Capstone [3] and pefile [4] modules will be utilized to form disassemble scripts. Moreover, portable executable headers include valuable information such as SizeOfCode and AddressOfEntryPoint [5]. We aim to extend our feature vector by adding new features from PE headers. Feed Forward Neural Network structure will be used for this model.

The second approach is to represent an executable file as a grayscale image by interpreting every byte as one pixel in an image, ranging from 0 to 255. This approach reduces malware classification problem to image classification problem on which Convolutional Neural Networks are very efficient. Studies show that images of the same malware family are similar to each other while they are distinct from images of other families [6], [7], [8].

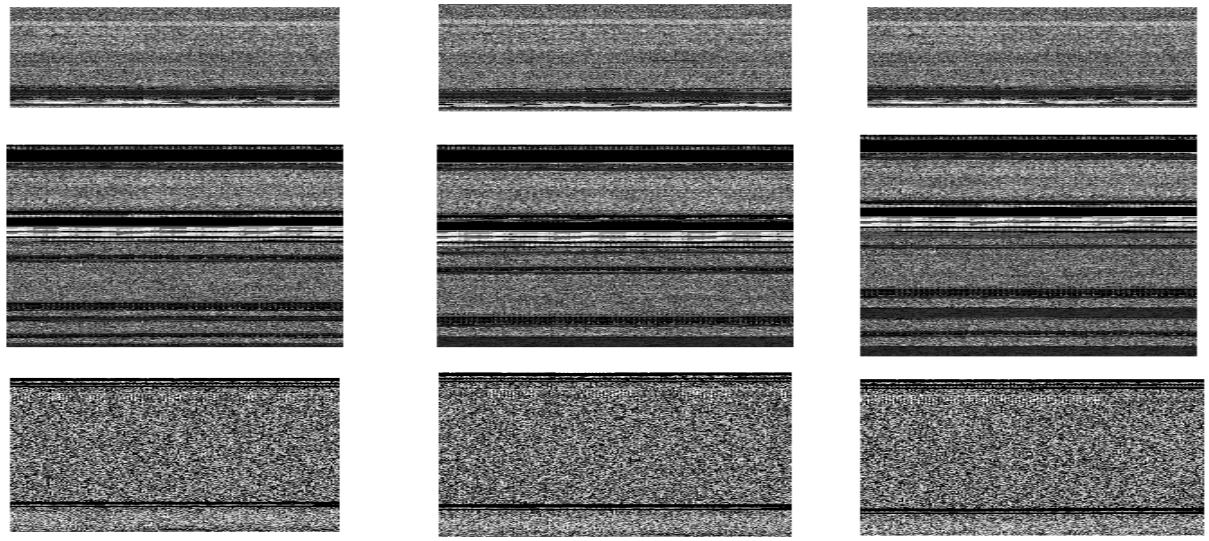

Figure 1: The first row represents Yuner.A samples, the second row represents VB.AT samples and the third row represents Skintrim.N samples [9].

The third approach is to combine models discussed in the first and the second approach and form a multimodal malware classifier. Output of 2 different layers fed with different inputs will be concatenated into a single layer. We observed similar approaches [10], [11] and decided to implement this specific approach to increase the evaluation score of the static classification.

In dynamic analysis, a portable executable's Windows API call sequence will be analyzed to create the feature vector [12]. For this purpose, an executable will be run on a sandbox for a small amount of time. After that, the sandbox will return information regarding the API call sequence. Cuckoo Sandbox [13] and a server where the sandbox can operate will be utilized. For the model, Feed Forward Neural Network or XGBoots will be selected depending on their performance.

Dataset for training the models will be taken from VirusShare [14]. Academic API of VirusTotal [15] will be utilized to label the dataset. Moreover, we aim to enrich our dataset from the executable files that users upload to the server when using Web Client.

Since CleaverWall is an open source project, we initially do not aim to compete with premium services. However, open source anti-malwares such as ClamAV and other applications using only signatures based detection methods perform poorly. Therefore, our goal is to surpass them on both evaluation and performance metrics due to our machine learning approach. The innovation type that we want to implement is product performance and it is incremental. As we improve our application, we want to change the case that reliability is expensive.

1.2 Constraints

1.2.1 Implementation Constraints

- Source code will be controlled through git, on a Github repository.
- The desktop and server-side applications will be written in Python. For the desktop application, C scripts may be written as needed.
- Cuckoo Sandbox will be utilized.
- Capstone will be used as the disassembler tool.
- Tensorflow [16] framework will be used for machine learning operations.
- The simple web client will be written in Javascript.

1.2.2 Economic Constraints

- Current economic extent of the project is limited to the budget of the project members. Future iterations of the project may include investment funds and/or grants from third parties.
- Realistically, to be able to compete with current products on the market, the project will need the mentioned economic support mostly for server and licensing costs, which are necessary for good user experience and project maintenance. Those are for now provided by academic and free/trial licenses.

1.2.3 Sustainability Constraints

- New data users share has to be fed to the models.
- There will be no micropayments for accessing any application features.

1.2.4 Resource Constraints

- The server has to be scalable enough to answer multiple requests to do costly operations on sandbox environment
- Even though the system will not be signature-based, data is needed to train machine learning models. Economic constraints limit the quantity of data to be used for model training. The data are now gathered from VirusShare and they will be labeled using the Academic API of VirusTotal.

1.3 Professional and Ethical Issues

1.3.1 Professional Issues

- We will cite every academic paper and third party organizations that benefit us.

1.3.2 Ethical Issues

- We need to get user's permission for adding the features of the executable to the dataset.
- We will not share any data that is uploaded by any user.

2. Requirements

2.1 Functional Requirements

2.1.1 User Functionalities

Users can:

- Scan a portable executable file through the desktop or the web app,
- Scan a windows directory,
- Demand further analysis with other heavier machine learning models,
- View the detailed log of scan results,
- Schedule an auto-scan process for a specific file path,
- Determine options such as quarantine or deletion for a scanned portable executable file which is detected as malicious.

2.1.2 System Functionalities

The server can:

- Accept portable executable files,
- Apply static and dynamic analysis on them,
- Return the results,
- Save the obtained data to a database.

Desktop application can,

- Use models of different weights to scan a portable executable file statically,
- Scan directories,
- When needed, send files to server for further analysis,
- Display the results,
- Quarantine detected malwares & delete them on demand,
- Present a UI.

Web client can:

- Display a UI that accepts portable executable files,
- Display the results,
- Display information about the system.

2.2 Non-Functional Requirements

2.2.1 Usability

- Our graphical user interface will be clear and intuitive to increase the ease of usage.
- We will ensure that the graphical user interface styles of the desktop application and the web server are similar to each other so that our customers will not have a hard time when they switch the service that they are using.

2.2.2 Security and Privacy

- We will ensure that the uploaded files will be safe against cyber attacks.
- We will ensure that our classification model can not be disturbed by outside forces.
- Newly obtained and saved data to improve the machine learning model should not be disturbed by outside forces.

2.2.3 Reliability

- We will ensure that our classification model can classify mainstream malware families.
- By using the new saved data, our classification model should be able to increase its accuracy.

2.2.4 Scalability

- The server should be able to analyze multiple files that are uploaded by different users concurrently.

2.2.5 Maintainability

- The mean time to restore the system following a system failure on the server side must not be greater than 10 minutes. Mean time to restore the system includes all corrective maintenance time and delay time.

2.3 Risks

- There is a risk that both the desktop application and the web client can not be implemented before the deadline. For that case, we may have to choose one of them for implementation.
- Public datasets have imbalanced distribution which might cause high false positive rates despite high training accuracy rates.

3.References

[1] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, “Novel feature extraction, selection and fusion for effective malware family classification,” *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016.

[2] M. A. U. of Ballarat, M. Alazab, U. of Ballarat, U. of B. V. Profile, S. V. U. of Ballarat, S. Venkatraman, P. W. U. of Ballarat, P. Watters, M. A. D. University, M. Alazab, D. University, T. A. N. University, U. of Technology, and O. M. V. A. Metrics, “Zero-day malware detection based on supervised learning algorithms of API call signatures: Proceedings of the ninth Australasian Data Mining Conference - volume 121,” *DL Hosted proceedings*, 01-Dec-2011. [Online]. Available: https://dl.acm.org/doi/10.5555/2483628.2483648. [Accessed: 16-Oct-2022].

[3] Capstone, “The ultimate disassembly framework,” *Capstone*, 08-May-2020. [Online]. Available: https://www.capstone-engine.org/. [Accessed: 16-Oct-2022].

[4] Erocarrera, “Erocarrera/pefile: Pefile is a python module to read and work with PE (portable executable) files,” *GitHub*. [Online]. Available: https://github.com/erocarrera/pefile. [Accessed: 16-Oct-2022].

[5] Karl-Bridge-Microsoft, “PE format - win32 apps,” *Win32 apps | Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format. [Accessed: 16-Oct-2022].

[6] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images,” *Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec '11*, 2011.

[7] D. Gibert, C. Mateu, J. Planes, and R. Vicens, “Using convolutional neural networks for classification of malware represented as images - journal of computer virology and hacking techniques,” *SpringerLink*, 27-Aug-2018. [Online]. Available: https://link.springer.com/article/10.1007/s11416-018-0323-0. [Accessed: 16-Oct-2022].

[8] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, “Malware classification with deep convolutional Neural Networks,” *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018.

[9] “Grayscale images of malware samples belonging to different malware ...” [Online]. Available: https://www.researchgate.net/figure/Grayscale-images-of-malware-samples-belonging-to-different-malware-families-in-BIG-2015_fig3_358487073. [Accessed: 16-Oct-2022].

[10] D. Gibert, C. Mateu, and J. Planes, “Hydra: A multimodal deep learning framework for malware classification,” *Computers & Security*, 12-May-2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820301462. [Accessed: 16-Oct-2022].

[11] T. G. Kim, B. J. Kang, M. Rho, S. Sezer, and E. G. Im, “A multimodal deep learning method for Android malware detection using various features,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.

[12] Y. Ki, E. Kim, and H. K. Kim, “A novel approach to detect malware based on API call sequence analysis,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 659101, 2015.

[13] “Automated malware analysis,” *Cuckoo Sandbox - Automated Malware Analysis*. [Online]. Available: https://cuckoosandbox.org/. [Accessed: 16-Oct-2022].

[14] *VirusShare.com*. [Online]. Available: https://virusshare.com/. [Accessed: 16-Oct-2022].

[15] *Virustotal*. [Online]. Available: https://www.virustotal.com/gui/home/upload. [Accessed: 16-Oct-2022].

[16] “Tensorflow,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/. [Accessed: 16-Oct-2022].