An EMBER Static Malware Analysis Tactical Decision Aid for Incident Responders

Joel Meoak
MSCS Student, Beacom College of Computer and Cyber Sciences
Dakota State University
Madison SD, USA
joel.meoak@trojans.dsu.edu

***Abstract*—There are a range of products available to incident responders for identifying malware, but with the existing solutions, many problems arise when using them as a tactical decision aid for quickly looking at potentially malicious files. Solutions are frequently signature based and, as a result, are reliant on samples being exact matches for previously identified malware as detected by anti-virus (AV) engines. Online options run into confidentiality problems and uploading could alert attackers that the incident response team is investigating their tools. Free sandboxes can be effective but take a significant amount of work to set up and require hardware that might not be easily available in the field. The purpose of this project was to use the Elastic Malware Benchmark for Empowering Researchers (EMBER) dataset to create a portable tactical decision aid for incident responders to quickly identify if a binary is worth examining further. Additionally, this project evaluates the tactical decision aid on more recent malware to include samples from 2022 through 2023 with the goal to determine if the 2018 EMBER dataset provides an adequate training benchmark for modern malware.**

Keywords—Malware Analysis, PE Analysis, Static Analysis, Incident Response, EMBER, Machine Learning

I. Introduction

An incident response team is called in when an institution has identified an adverse event affecting their information technology (IT) systems [1]. This could include a number of different issues, from unauthorized access, modification or destruction of data, or degradation of system availability [2]. Incident responders need to quickly work through the detection and analysis phase of their process in order to contain the incident and prevent further damage to the IT systems. This identification phase is the first goal for responders in the field, and if not done quickly and decisively, the incident can expand and result in greater harm [1]. One the key aspects of this phase is the use of tools to identify malicious binaries or malware that may be present on the affected systems. The tools used for this task must work quickly, be portable for use on a basic laptop in the field or deployable to remote systems, and be able to identify both previously known and unknown malware.

An initial search of malware analysis tools for incident responders leaves any team with no shortage of open source and commercial options available depending on usage, budget, and technical expertise. The existing solutions present many issues for incident response teams working at remote sites or who have limited experience with malware analysis. VirusTotal is the largest free malware repository and users can submit hashes or binaries to determine if anti-virus (AV) engines can detect the sample as malware [3]. Unfortunately, this service does not always work well for new variants of malware and is not available to incident response teams without internet access. Additionally, internet connected services can alert attackers that their tools have been discovered. Cuckoo sandbox is a robust solution used by many teams, but it takes experience to use and requires a dedicated workstation for the sandbox [4].

This project focused on building a tactical decision aid (TDA) that could be used by incident responders in the field. Available open-source options are used as the primary comparison point. The core of the tool is a machine learning (ML) model trained on the 2018 Elastic Malware Benchmark for Empowering Researchers (EMBER) dataset [5]. This dataset was developed by Endgame, Inc, which was later acquired by Elastic, and has been used in multiple academic papers since its release in 2017. The dataset is made up of a collection of portable executable (PE) headers for benign and malicious binaries. The PE format is included in all Windows 32-bit executable (EXEs) and dynamically linked libraries (DLLs) [6]. While not all malware falls into this category, this encompasses the majority of malware [7].

II. Objectives

The goal of this project was to create a static malware analysis tactical decision aid (TDA) for incident responders based on the EMBER dataset and evaluate it against modern malware. The TDA, named EMBER Malware Tactical Decision Aid (EMT), is a framework for employing various machine learning models and is independent of online repositories.

A. Primary Goals

- EMT takes a 32-bit or 64-bit PE binary and provide a recommendation as likely malicious or benign.
- EMT makes a recommendation within one minute.
- EMT does not require internet access to function.
- EMT is portable and usable on Linux systems with a git clone and pip install or on windows systems with Python installed.
- EMT is able to switch between different models for the ML engine and change models automatically for DLL or EXEs.

B. *Secondary Goals*

- Evaluate the performance of different ML models trained on the EMBER dataset against more modern malware using EMT.
- Evaluate the performance of different ML models trained on the EMBER dataset against DLLs versus EXEs using EMT.

III. BACKGROUND AND LITERATURE REVIEW

This section covers the EMBER dataset used in this project, the significant papers associated with the dataset, and an overview of the existing tools for identifying malware.

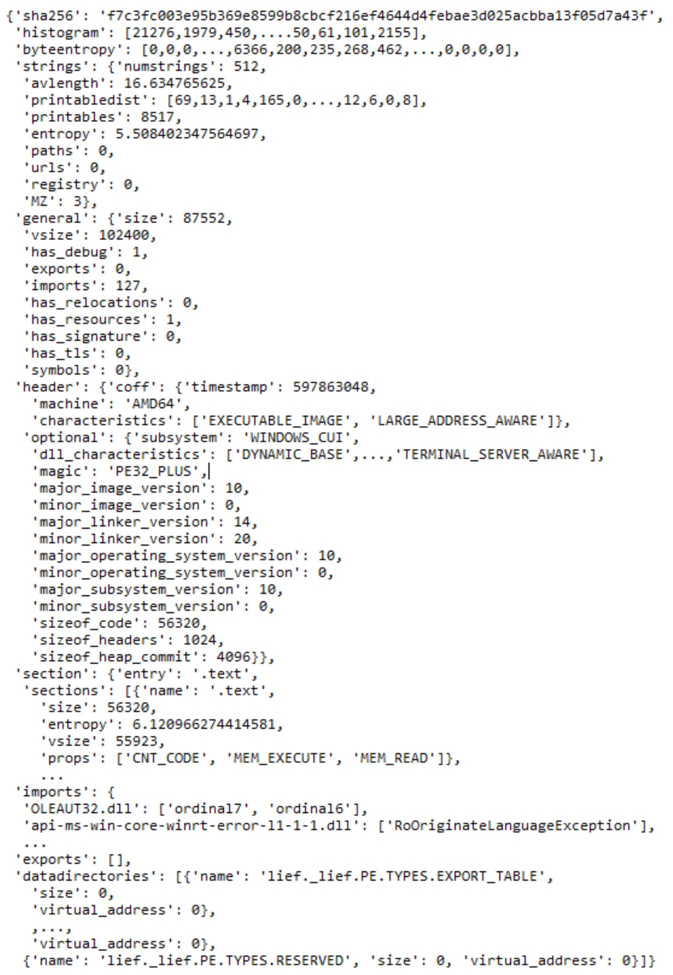
A. *EMBER Dataset*

This project is primarily based on the Elastic Malware Benchmark for Empowering Researchers (EMBER) dataset published in 2017 and then later updated in 2018. The updated EMBER dataset is a collection of one million samples in JSON format (800k for training and 200k for testing), with each line containing the PE header data for one sample [8]. A truncated version of the PE header for the executable bash.exe is shown in Figure 1. Out of the 800k samples for training, 300k are benign, 300k are malicious, and 200k are unlabeled. This project, and most papers on the dataset, focus on the labeled dataset and implementation against supervised models. Besides first seen date and SHA256 hash, the headers are broken into eight groups of raw features of variable length and content. The sections are Byte Histogram, Byte Entropy Histogram, String Extractor, General File Info, Header File Info, Section Info, Imports Info, and Exports Info. Work by Oyama et al in the academic research covers feature importance across these groups [9].

In addition to the dataset, the researchers who developed EMBER built the ember Python module which includes functions to extract the dataset from the multiple json files used to store the dataset, vectorize that dataset based on the eight sections, and a function to extract and vectorize the header of a new PE file so that it can be classified by the ML model trained on the dataset. The feature extractor is implemented with the Library to Instrument Executable Formats (LIEF) module. Once extracted a single executable is represented with 2381 features. The feature count and summary of each section is listed below.

Overview of Section Features (2381 Features) [6]

- Byte Histogram (256): The number of instances of each byte value in the file.
- Byte Entropy Histogram (256): Joint distribution of entropy and bytes values in the file (uniform entropy indicates the file is packed).
- String Extractor (255): A summary of string information to include histogram of characters, number of strings, average string lengths, url count, and path count.
- General File Info (10): General file information to include sizes, has_debug, import count, export count, has_relocations, has_resources, has_signature, has_tls, and symbol count.
- Header File Info (62): Machine architecture, timestamps, characteristics, dll_characteristics, and other optional header info.
- Section Info (255): Information on each file section to include section names, size, vsize, entropy, and properties.
- Imports Info (1280): Imported libraries (DLLs) and the functions imported from each library.
- Exports Info (128): Exported functions for the file.



```
{'sha256': 'f7c3fc003e95b369e8599b8cbcf216ef4644d4febae3d025acbba13f05d7a43f',
 'histogram': [21276,1979,450,....50,61,101,2155],
 'byteentropy': [0,0,0,...,6366,200,235,268,462,...,0,0,0,0],
 'strings': {'numstrings': 512,
  'avlength': 16.634765625,
  'printabledist': [69,13,1,4,165,0,...,12,6,0,8],
  'printables': 8517,
  'entropy': 5.508402347564697,
  'paths': 0,
  'urls': 0,
  'registry': 0,
  'MZ': 3},
 'general': {'size': 87552,
  'vsize': 102400,
  'has_debug': 1,
  'exports': 0,
  'imports': 127,
  'has_relocations': 0,
  'has_resources': 1,
  'has_signature': 0,
  'has_tls': 0,
  'symbols': 0},
 'header': {'coff': {'timestamp': 597863048,
   'machine': 'AMD64',
   'characteristics': ['EXECUTABLE_IMAGE', 'LARGE_ADDRESS_AWARE']},
  'optional': {'subsystem': 'WINDOWS_CUI',
   'dll_characteristics': ['DYNAMIC_BASE',...,'TERMINAL_SERVER_AWARE'],
   'magic': 'PE32_PLUS',|
   'major_image_version': 10,
   'minor_image_version': 0,
   'major_linker_version': 14,
   'minor_linker_version': 20,
   'major_operating_system_version': 10,
   'minor_operating_system_version': 0,
   'major_subsystem_version': 10,
   'minor_subsystem_version': 0,
   'sizeof_code': 56320,
   'sizeof_headers': 1024,
   'sizeof_heap_commit': 4096}},
 'section': {'entry': '.text',
  'sections': [{'name': '.text',
    'size': 56320,
    'entropy': 6.120966274414581,
    'vsize': 55923,
    'props': ['CNT_CODE', 'MEM_EXECUTE', 'MEM_READ']},
    ...
 'imports': {
  'OLEAUT32.dll': ['ordinal7', 'ordinal6'],
  'api-ms-win-core-winrt-error-l1-1-1.dll': ['RoOriginateLanguageException'],
  ...
 'exports': [],
 'datadirectories': [{'name': 'lief._lief.PE.TYPES.EXPORT_TABLE',
   'size': 0,
   'virtual_address': 0},
   ,...,
   'virtual_address': 0},
  {'name': 'lief._lief.PE.TYPES.RESERVED', 'size': 0, 'virtual_address': 0}]}
```

Figure 1: Raw features extracted from bash.exe PE file.

B. *Academic Research*

Since publication of the EMBER dataset, multiple teams have attempted to create ML models to accurately classify malware. In *Malware Detection through Machine Learning Techniques*, the authors were able to achieve accuracy above 99% [10]. They were also able to detect one sample of new malware that, at the time of the study, had only been released 15 days prior. While these results are promising, their implementation did not do extensive testing against newer malware and the research was only conducted one year after the 2018 version of EMBER was released.

More recent work, including a paper in 2020 by C. Galen and R. Steele, looked at the performance of machine learning models based on the 2018 EMBER dataset on malware variants that have been seen after the release of the dataset [11]. They found that a random forest model performed better with newer malware variants, but their work was only two years past the release of the dataset. Their work is another reason why random forests were one of the main models tested in this project.

The paper *Identifying Useful Features for Malware Detection in the EMBER Dataset* by Oyama et al modified the ember module to allow for piecewise extraction of the different sections in order to train models with section groups to gauge importance of each and determine if a reduced feature group is viable for model development [9]. They found that using just General Info, Header Info, and String Extractor, their model was able to achieve an accuracy of 83.7% with just 176 features as compared to the 2381 for the used normally with EMBER. This allowed for 85.5% less training time and reduced the dataset memory usage down to 16.7% of the original. While their research was primarily looking at build a training dataset balanced between accuracy, data size, and training time, it is also important to note that their research showed that all features sections contribute to the accuracy of the final model with inclusion of all eight sections having the highest accuracy. In the use case for this project, training traditional models using all of the sections was not a significant limitation. However, both training time and data size were limitations for neural network models. Future research could be done for using the EMT Framework to train on paired down feature sets outlined by these researchers to train more complex neural models.

C. *Existing Solutions*

The existing solutions can be primarily grouped into three main categories: Online sandboxes, offline sandboxes, and signature repositories.

a) Online Sandboxes: Services like Joe's Sandbox and ANY.RUN offer an excellent free service to users. The privacy policy differs for many of these services, but two main problems are presented regardless of the specifics of the policy. First, uploading binaries to an online sandbox could potentially disclose internal company data [12]. This is especially a danger if the malware in question is a keylogger or was used to gather corporate data. Additionally, benign files uploaded could be proprietary to the company that brought in the incident responders. Second, malicious actors can monitor many online sandboxes, and even if they don't have the ability to monitor the sandbox, the execution of the malware in some sandbox configurations allow the malware to download additional stages from a domain owned by malicious actors, which can alert them that their tool has been found. Alerting malcious actors that their tools have been discovered before the incident is contained could result in them executing ransomware, exfiltrating from the network, or acting to remove their indictors from the network to prevent the incident responders from finding them.

b) Offline Sandboxes: Offline sandboxes don't have the confidentiality problems that are presented with online solutions, but they have their own issues. Cuckoo Sandbox is one of the most popular offline sandboxes used by incident responders. Unfortunately, developer support ended in 2020, so implementation can be difficult and involve technical expertise separate from the core skillset of incident responders [4]. Additionally, these sandboxes are computationally intensive, so suspect binaries must be removed from the infected workstation and moved to an external device with the implemented sandbox. This may not be an issue on an individual level, but scaled across multiple incident responders and thousands of binaries, this task becomes very time-consuming.

c) Signature Repositories: Similar to sandboxes, signature repositories also have online and offline solutions. Online repositories involve many of the same problems of confidentiality and alerts to malicious actors as online sandboxes [12]. More broadly, signature repositories rely on past variants of malware. If the binary is a new variant of malware or is using an obfuscation method to mask the signatures used by the repository, it will not detect the binary as malware.

IV. Methodology and Implementation

This section is broken into three parts: the model training and evaluation against EMBER, the development of the EMT framework for implementing the models, and the evaluation against modern malware and penetration testing tools.

A. *Machine Learning (ML) Engine*

The ML Engine consists of a trained model that is able to evaluate a vectorized PE header to make a determination if the binary should be further investigated as it is likely malware or if it is likely benign. The 10 models implemented for use in the ML Engine were s LightGBM model trained by the EMBER developers; Scikit-Learn's Random Forest Classifier, Decision Tree Classifier, Extremely Random Forest Classifier, Logistic Regression, Gradient Boosting Classifier, and Ada Boost Classifier; Tensor Flow Keras' Sequential Neural Network, Simple Recurrent Neural Network, and Convolutional Neural Network.

Prior to training, the EMBER dataset was extracted and vectorized to normalize the features into a standard format. The training dataset of 800k samples was reduced to 300k malicious labeled samples and 300k benign labeled samples. The testing set consisted on 100k malicious samples and 100k benign samples. After fitting, the models were saved as joblib files using the joblib Python module. Due to the significant size of the dataset, extensive hyperparameter searching was not viable, but this could be an area for future research using more robust hardware. All model training and testing was conducted with an Intel i7-10750H CPU @ 2.60 GHz 6-cores and 32 GBs of DDR4 RAM.

Additionally, while performance against the original EMBER dataset is important, overfitting was a concern, especially when using the model to predict modern malware and penetration testing tools where programming characteristics and malicious trends might differ. During testing, initial models were identifying modern malware if it was implemented as an exe but not as a dll. Data preparation showed that out of the 600k training samples, 507,262 were EXEs and 92,738 were DLLs. Testing also showed that when tested against just DLLs from the

Table 1: Model Performance Against EMBER Test Dataset

Model	Model Notes	Data Subset	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	FPR (%)	FNR (%)
LightGBM	EMBER Provided	Total	97.6	99.1	96.1	97.6	0.8	3.9
LightGBM	EMBER Provided	DLL Only	96.5	99.0	88.9	93.7	0.4	11.1
Decision Tree		Total	91.9	92.0	91.7	91.9	7.9	8.3
Decision Tree		DLL Only	91.9	87.5	84.6	86.0	5.0	15.4
Random Forest		Total	95.4	95.9	94.9	95.4	4.1	5.1
Random Forest		DLL Only	94.4	94.6	86.0	90.1	2.1	14.0
Random Forest	DLL Weight 5 to 1	Total	95.2	95.7	94.7	95.2	3.5	5.6
Random Forest	DLL Weight 5 to 1	DLL Only	95.2	94.9	88.5	91.6	2.0	11.5
Extremely RF		Total	95.4	96.4	94.4	95.4	3.5	5.6
Extremely RF		DLL Only	94.2	95.3	84.4	89.5	1.7	15.6
Extremely RF	DLL Weight 5 to 1	Total	95.3	96.3	94.2	95.2	3.6	5.8
Extremely RF	DLL Weight 5 to 1	DLL Only	94.8	95.4	86.5	90.7	1.7	13.5
Logistic Regression		Total	62.2	59.0	79.9	67.9	55.6	20.1
Logistic Regression		DLL Only	51.2	32.9	63.2	43.3	53.8	36.8
Gradient Boosting		Total	90.2	88.5	92.4	90.4	12.0	7.6
Gradient Boosting		DLL Only	88.4	81.0	79.2	80.1	7.7	20.8
AdaBoost Classifier		Total	95.1	95.8	94.3	95.1	4.1	5.7
AdaBoost Classifier		DLL Only	94.7	94.6	86.8	90.5	2.0	13.2
Sequential	8 Dense, 128 epoch	Total	65.3	78.0	42.5	55.0	12.0	57.5
Sequential	9 Dense, 32 epoch	Total	60.6	57.5	81.4	67.4	60.2	18.6
Convolutional		Total	64.2	61.9	73.6	67.3	45.3	26.4
Simple RNN		Total	63.8	58.7	92.9	72.0	65.2	7.1

EMBER testing dataset, the models trained on both had similar accuracies but lower recall. To help balance this, Random Forest and Extremely Random Forest were each trained with five to one weighting on DLLS from the training set as a form of oversampling without expanding the data size.

Table I shows a summary of the tests run against each model, the specific parameters used for each model, and the performance metrics of accuracy, recall, precision, F1 score, False Postive Rate (FPR), and False Negative Rate (FNR) for each test. The majority of the models perform well on the total dataset with the standard metrics above 90%. These scores reduced across the board when tested agaisnt only the DLLs with slighly stronger DLL classification performance from the weighted models. Out of the traditional models, logistic regression was the only one that performed poorly and was not implmented as a ML engine for modern malware testing. The neural network models all performed poorly with accuracies around 60-65% and extremely high either FPR or FNR. The implementation of more expansive neural network models, trained on more sophisticated hardware, could be an area for future research. Due to the low accuracy scores, the four neural network models were not implemented in the final modern malware and penetration tool testing.

B. *EMT Framework*

The EMT framework provides a Python-based command line utility for using one or more ML engines against one or more executables. The primary framework is included as a single Python file with a recommended directory structure for running the models. This can be run on any Linux systems or any Windows system with Python installed as long as its dependencies are met. Its dependencies include modules standard with any Python distribution such as os or joblib as well as more specialized Python modules to include ember, scikit-learn, lief, and the libraries for any additional models used which could tensorflow or lightbgm. In both cases a separate joblib file for each model is required and can be from the standard list trained for this project or can include custom models trained separately.

The EMT framework is scalable as both a simple tactical decision aid to classify single executables with a single model or can classify all executables in a directory using all trained models available and then store those results in a csv format. This allows for more expansive testing in future research and easy evaluation of new models. EMT runs as looping command line utility with execution in either normal mode or advanced mode. Advanced mode includes testing with multiple models and DLL detection where the framework can change models by using the lief module to determine if the file is a DLL or executable and then change to the appropriate model such as one of the ones trained with a weighted toward DLL identification. The EMT User Guide is included as Appendix G.

C. *Modern Malware and Penetration Tool Testing*

Since this section involved working with current malware, extreme caution was implemented in the evaluation phase. While the model does not run the malware itself, accidental execution of malware that is not encrypted is always a concern. All testing was done inside a VMware Workstation Pro virtual machine running Kali Linux.

Malware testing was done in groupings that were involved in specific attacks. 124 samples were evaluated as part of the EMT malware testing. The malware samples came from attacks in 2022 and 2023 attributed to UNC2589/Bleeding Bear which targets Ukraine [13]. Additionally, the EMT was run against 42

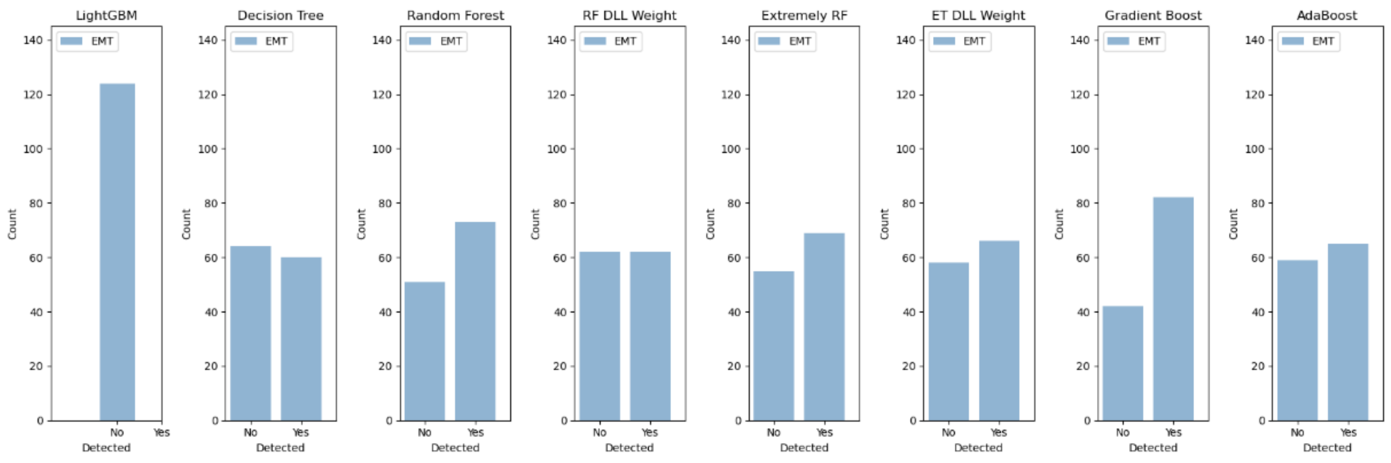


Figure 2: EMT Malware Detection

penetration testing tools which have been used by many Advanced Persistent Threats (APTs) in the past. A summary of all of this testing is included in the results and a more detailed breakdown including hashes of each sample tested against is included in Appendix A. Additionally, the malware detection rate for each attack and tool was compared to VirusTotal detection of the same samples as a comparison point [3].

For the penetration testing tool portion, 42 version of Metasploit's Meterpreter payload were created using the domain example[.]com as the callback domain and 443 or 80 as the port, aligned with payload type [14]. Metasploit is the most popular penetration testing framework with Meterpreter as the primary payload used in many tests. Each instance was created using msfvenom and included 12 stageless (all in one) payloads and 30 stagers (callback beacons or port binder). The variations included changing payload type between bind/callback payload type; x86 or x64 architecture; networking protocol; and EXE, EXE Service (uses service style API calls), or DLL file type.

V. Results and Discussion

A. *Modern Malware Performaance*

Figure 2 shows a summary of the model performance against the 124 samples. For each of the 124 samples tested, Virus Total detected the sample as malicious. VirusTotal scoring was created using the VirusTotal API to retrieve scores for files based on the SHA256 hash and determine if at least 50% of the anti-virus engines in Virus Total identified the sample [3]. Details of this code can be found in Appendix D. This data shows that the majority of models performed similarly with two exceptions: Sklearn Gradient Boosting Classifier (GBC) and LightGBM. The GBC model detected significantly more of the malware samples than any of the other models. The LigthGBM model was provided with the EMBER dataset and had the highest accuracy of any model during testing against the original EMBER dataset. It is likely that this model is overfitted to the original dataset and as such is not able to extrapolate to newer malware variants.

Table 2 shows a summary of the accuracy metrics for each model against the modern malware testing set. While the initial accuracy calculated shows the GBC model to have a significantly higher accuracy, it is important to also consider FPR. The modern malware testing does not include benign samples which is one of the distinguishing features of the EMBER dataset since many benign samples are proprietary. Without benign samples, the FPR from EMBER testing is used in order to extrapolate an adjusted accuracy. The formula *Adj Acc = (Acc * 124 + (1 - FPR) * 124) / 248* was used which extrapolates the accuracy of a dataset using the 124 malware samples and 124 benign sample using the FPR previously calculated. The adjusted accuracy takes the high FPR for the GBC model into account and the resulting scores show that Random Forest Classifier (RF) model had the highest adjusted accuracy of 77.4%. While this is not an extremely high accuracy, the problem of malware detection is very difficult and the strong identification abilities of VirusTotal against this dataset likely is due to the well documented and signatured nature of the attacks this malware was pulled from.

Table 2: Modern Malware EMT Testing Metrics

Model	Testing FPR (%)	Accuracy (%)	Adj Acc (%)	Detect Count
LightGBM	0.8	0.0	49.6	0/124
Decision Tree	8.0	48.4	70.2	60/124
Random Forest	4.1	58.9	77.4	73/124
RF DLL Weight	3.5	50.0	73.2	62/124
Extremely RF	3.5	55.6	76.1	69/124
ET DLL Weight	3.6	53.2	74.8	66/124
Gradient Boost	12.0	66.1	77.0	82/124
AdaBoost	4.1	52.4	74.2	65/124

The modern malware dataset included 89 EXEs and 36 DLLs. While the EMBER testing set metrics showed weaker performance by the models on DLLs as compared to EXEs, that was not the case for the modern malware testing. Table 3 shows the results of testing against only the DLLs. While the RF and Extremely RF models again showed strong results, in this case the GBC model out performed all of the others with an adjusted accuracy of 92.0%

Table 3: Modern Malware DLL EMT Testing Metrics

Model	Testing FPR (%)	Accuracy (%)	Adj Acc (%)	Detect Count
LightGBM	0.4	0.0	49.8	0/36
Decision Tree	5.0	44.4	69.7	16/36
Random Forest	2.1	69.4	83.7	25/36
RF DLL Weight	2.0	63.9	81.0	23/36
Extremely RF	1.7	63.9	81.1	23/36
ET DLL Weight	1.7	66.7	82.5	24/36
Gradient Boost	7.7	91.7	92.0	33/36
AdaBoost	2.1	69.4	83.7	25/36

B. *Penetration Testing Tool Performance*

The EMT detection metrics for the penetration testing tools can be seen in Table 4. In this portion of the testing, the GBC model performed far stronger than any model in both detection of 33/42 and in adjusted accuracy after accounting for the FPR with a score of 83.3%. The RF model trained with weighting toward DLLs performed very close to GBC, with a much lower false positive rate, so using this model or a variation of it might be a more balanced option.

Table 4: Penetration Testing Tool Evaluation Metrics

Model	Testing FPR (%)	Accuracy (%)	Adj Acc (%)	Detect Count
LightGBM	0.8	0.0	49.6	0/42
Decision Tree	8.0	45.2	68.6	19/42
Random Forest	4.1	47.6	71.8	20/42
RF DLL Weight	3.5	59.5	78.0	25/42
Extremely RF	3.5	45.2	70.9	19/42
ET DLL Weight	3.6	52.4	74.4	22/42
Gradient Boost	12.0	78.6	83.3	33/42
AdaBoost	4.1	50.0	73.0	21/42

VI. Conclusion

In this project, 13 models were trained against the EMBER dataset and the top eight models were implemented in the EMT framework. The EMT framework was then run against 124 modern malware samples and 42 instances of Metasploit's Meterpreter payload [13]. With modern malware testing 7/8 models showed adequate performance with adjusted accuracy extrapolated from the EMBER testing FPR between 70.2-77.4%. The outlier model of LightGBM which is packaged with the EMBER dataset as a pretrained model did not detect any samples showing it is overfit for newer malware. The Sklearn's Random Forest Classifier achieved the highest adjusted accuracy of 77.4%. With the penetration testing tool portion, the same seven models showed adequate performance with adjusted accuracies between 68.6-83.3%. The pretrained LightGBM again did not detect any of the penetration testing payloads. Sklearn's Gradient Boosting Classifier achieved the highest adjusted accuracy of 83.3% on the 42 samples but it should be noted that this model has a high FPR rate in EMBER testing so implementation of the Random Forest with DLL weighting might be a more balanced model to implement.

This project shows that despite six years since it's updated publication in 2018, the EMBER dataset is still relevant for training new machine learning models. Unfortunately, the changes in modern malware and penetration testing tools since the release of the dataset cause some models trained on the dataset to be overfit for use in new malware detection. The dataset includes code for adding new samples to the training data and future research could expand on the dataset with more recent malware to help with this issue. Due to the larger size of the dataset (600k training samples), it will take a large number of new samples to adequately allow for training on newer malware designs, even if oversampling is used on the new samples. Additionally, the testing in this project was not extensive as compared to the EMBER dataset so bias toward specific malware may change the accuracies observed in either direction. Future research could look at additional modern malware and penetration testing tools as well as use more powerful hardware to train neural network models, potentially with a paired down feature list.

The EMT framework consisting of one Python file and four Jupyter notebooks provide an easy mechanism to train and implement new models and evaluate them against new malware samples. The main Python file for the framework, along with all of the notebooks for training, evaluating, and post-processing the data, are included in the Appendix of this paper, along with a user guide and an xlsx document containing the SHA256 hashes of all of the modern malware samples and Meterpreter payloads tested against in this project.

References

[1] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, Computer security incident handling guide (National Institute of Standards and Technology), NIST Series SP 800-61, Rev 2, Aug. 2012. doi:10.6028/nist.sp.800-61r2

[2] M. Nieles, K. Dempsey, and V. Y. Pillitteri, An introduction to information security, (National Institute of Standards and Technology), NIST Series SP 800-12 Rev 1, Jun. 2017. doi:10.6028/nist.sp.800-12r1

[3] "How it works," VirusTotal, https://docs.virustotal.com/docs/how-it-works (accessed Feb. 19, 2024).

[4] "What is cuckoo?" Cuckoo Sandbox, https://cuckoo.readthedocs.io/en/latest/introduction/what/ (accessed Feb. 19, 2024).

[5] H. Anderson and P. Roth, "EMBER: an open dataset for training static pe malware machine learning models," arXiv:1804.04637 [cs.CR], April 2018, Available: https://arxiv.org/pdf/1804.04637.pdf

[6] Microsoft, "PE format - win32 apps," Microsoft Learn, https://learn.microsoft.com/en-us/windows/win32/debug/pe-format (accessed Feb. 19, 2024).

[7] J. Leyden, "64-bit malware threat may be itty-bitty now, but it's only set to grow," The Register, https://www.theregister.com/2017/05/24/64bit_malware/ (accessed Feb. 19, 2024).

[8] P. Roth, "EMBER Improvements," Presented at CAMLIS 2019. Available: https://www.camlis.org/2019/talks/roth

[9] Y. Oyama, T. Miyashita, and H. Kokubo, "Identifying useful features for malware detection in the ember dataset," *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, Nov. 2019. doi:10.1109/candarw.2019.00069

[10] A. Amer and N. A. Aziz, “Malware detection through Machine Learning Techniques,” *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no. 5, pp. 2408–2413, Oct. 2019. doi:10.30534/ijatcse/2019/82852019

[11] C. Galen and R. Steele, “Evaluating performance maintenance and deterioration over time of machine learning-based malware detection models on the ember pe dataset,” *2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS)*, Dec. 2020. doi:10.1109/snams52053.2020.9336538

[12] P. Arntz, “Why you shouldn’t automate your virustotal uploads,” Malwarebytes, https://www.malwarebytes.com/blog/news/2022/04/why-you-shouldnt-automate-your-virustotal-uploads (accessed Feb. 19, 2024).

[13] M. Parkour, “Malware arsenal used by Ember Bear (aka UAC-0056, Saint Bear, UNC2589, Lorec53, ta471, Nodaria, nascent Ursa, LorecBear, Bleeding Bear, and Dev-0586) in attacks targeting Ukraine.” contagio. (n.d.). https://contagiodump.blogspot.com/2023/02/malware-arsenal-used-by-ember-bear-aka.html

[14] Metasploit Unleashed: About the Metasploit meterpreter. OffSec. (n.d.). https://www.offsec.com/metasploit-unleashed/about-meterpreter/ (accessed Apr. 10, 2024).

[15] “Virustotal API V3 overview,” VirusTotal, https://docs.virustotal.com /reference/overview (accessed Apr. 20, 2024).

APPENDIX

A. *Malware and Penetration Testing Tool Performance Report (XLSX Excel Workbook)*

Appendix A is a single document containing a more detailed report on every malware and penetration testing sample tested against in this project including the attack/group the sample is associated with, the SHA256 hash of the file, the file type, and VirusTotal score.

B. *EMT Model Trainer Notebook*

Appendix B is the training notebook used to create all of the models for use in the ML engine including the specific parameters used within each of the traditional models and the layers used with the neural network models. This model provides a blueprint for creating future models for use with the EMT framework.

C. *EMT Model Evaluator Notebook*

Appendix C is the notebook used the evaluate all of the models against the EMBER testing set that were trained as part of the ML engine phase. It contains all of the metrics included in Table I to a higher level of precision, the confusion matrices for each model.

D. *VirusTotal Integrator Notebook*

Appendix D is the notebook used to integrate the Virus Total scores for each of the samples used in testing. This notebook takes in a csv report created by the EMT framework and implements the Virus Total Python module (vt-py) and Virus Total REST API to automatically pull reports on each sample and update the csv file [15].

E. *EMT Post Processing Notebook*

Appendix E is the notebook used to post process the csv files created using the EMT framework. It has portions to process data that is likely to already be on Virus Total like malware samples that are publicly available as well as code to handle data from evaluations of samples like penetration testing tools which will have unique hashes since they are generated to callback to specific domains.

F. *EMT Framework Main Python Code*

Appendix F is the Python code for the EMT framework which runs as a command line utility to evaluate one or more PE files using one or more models and print the results to the screen and save in csv format.

G. *EMT Framework User Guide*

Appendix G is the User Guide for the EMT Framework including instructions to install, implement new models, and run against new PE files.