Quo Vadis: Hybrid Machine Learning Meta-Model based on Contextual and Behavioral Malware Representations

Dmitrijs Trizna
d.trizna@pm.me
Microsoft

Abstract

We propose a hybrid machine learning architecture that simultaneously employs multiple deep learning models analyzing contextual and behavioral characteristics of Windows portable executable, producing a final prediction based on a decision from the meta-model. The detection heuristic in contemporary machine learning Windows malware classifiers is typically based on the static properties of the sample since dynamic analysis through virtualization is challenging for vast quantities of samples. To surpass this limitation, we employ a Windows kernel emulation that allows the acquisition of behavioral patterns across large corpora with minimal temporal and computational costs. We partner with a security vendor for a collection of more than 100k int-the-wild samples that resemble the contemporary threat landscape, containing raw PE files and filepaths of applications at the moment of execution. The acquired dataset is at least ten folds larger than reported in related works on behavioral malware analysis. Files in the training dataset are labeled by a professional threat intelligence team, utilizing manual and automated reverse engineering tools. We estimate the hybrid classifier's operational utility by collecting an out-of-sample test set three months later from the acquisition of the training set. We report an improved detection rate, above the capabilities of the current state-of-the-art model, especially under low false-positive requirements. Additionally, we uncover a meta-model's ability to identify malicious activity in both validation and test sets even if none of the individual models express enough confidence to mark the sample as malevolent. We conclude that the meta-model can learn patterns typical to malicious samples out of representation combinations produced by different analysis techniques. Furthermore, we publicly release pre-trained models and anonymized dataset of emulation reports.

CCS Concepts: • **Security and privacy → Malware and its mitigation**; • **Computing methodologies → Neural networks**; • **Hardware → Simulation and emulation**.

Keywords: malware, emulation, neural networks, convolutions, reverse engineering

1 Introduction

Machine learning (ML) algorithms have become essential to malicious software (malware) detection in conventional cybersecurity intrusion prevention systems. Such systems can learn common patterns across a vast malware dataset, obtaining a predictive power to classify previously unseen malicious samples. However, there is evidence that contemporary state-of-the-art models lack epistemic capacity due to limited contextual and behavioral awareness [15] since they mostly rely on representations acquired from static properties of the executable files [3, 21].

Human-produced malware analysis typically is based on static and dynamic properties of sample [29]. Static evaluation of malicious specimen provides readily available yet limited insights on its functionality, usually surpassed by a dynamic analysis through sample "detonation" in a controlled environment. However, the collection of Windows portable executable (PE) behavioral patterns through dynamic analysis sufficient for ML algorithms, especially if based on deep learning architectures, poses a significant challenge due to the computational burden of virtualization technology and the necessity to revert operating system setup from the contamination after malware detonation.

We perform dynamic analysis with malware-oriented Windows kernel emulator [18], thus achieving high analysis rates compared to virtualization. Because of data heterogeneity, we consider a composite solution with multiple individual pre-trained modules and a meta-model rather than building a single feature vector with end-to-end trainable architecture. This architecture allows extending the modularity of the decision heuristic with minimal efforts by retraining only a meta-model. In the scope of this publication assessment of hybrid ML architecture relied on three distinct analysis techniques:

- contextual information in the form of a file path on a system at the moment of execution;
- sample behavior expressed as a sequence of Windows kernel API calls;
- static representations obtained from the Windows PE structure.

We expect further work on additional behavioral models like network or filesystem telemetry analysis. Therefore, we release an anonymized emulation dataset publicly.

The lack of publication artifacts is a notorious drawback in any research and contributes to science's reproducibility crisis. Hence we disclose[1] the source code and pre-trained PyTorch [19], and scikit-learn [25] models, as well as provide `scikit-learn`-like [20] API for our model adhering to widely adopted interface of a machine learning objects. To our knowledge, we are the first in the security research community to publish a model that incorporates a single decision heuristic based on (a) contextual, (b) static, and (c) dynamic properties of the PE file.

We collect an out-of-sample dataset three months after model training. We acknowledge that malware classification based on hybrid representations of software yields improved detection performance and reduced false-positive rates against the evolving nature of malevolent logic compared to any individual method capabilities.

This article is structured as follows - Section 2 reviews related work, Section 3 describes dataset and its collection specifics, Section 4 covers architecture of hybrid architecture and data preprocessing, Section 5 reports performance of our model, Sections 6 draws conclusions from empirical observations and outlines future work options.

2 Related Work

Since the idea of malware detection using ML techniques was introduced by Schultz et al. [26], the field in research and industry has grown significantly. The prevailing part of ML malware classification is based on static analysis techniques, ML heuristics are applied to representations acquired from fixed properties of malware files [28, 31]. One of the first neural network applications for malware classification was shown by Raff et al. [21] who presented a *MalConv* model - a "featureless" Deep Neural Network (DNN) that reads raw bytes of executable and proceeds with embeddings and one-dimensional convolution.

Gradient Boosted Decision Tree (GBDT) models achieve notable success in malware classification, specifically the approach introduced by Anderson et al. [3]. Their work is meant to provide a benchmark dataset based on specific, pre-extracted properties from malware files, thus the title: *EMBER* (Endgame Malware BEnchmark for Research). At the same time, the paper includes an evaluation of LightGBM [12] model performance, describing an approach that employs a clever feature engineering phase. Ember representations incorporate domain knowledge into many effective static characteristics of PE files, becoming a *de facto* standard for static feature extraction from PE files in modern malware classification research. An interesting approach is shared by Rudd et al. [24], which utilizes Ember representation vectors to train a feed-forward neural network (FFNN).

ML algorithms are proven fruitful by utilizing dynamic analysis telemetry from malware "detonation" in a sandbox. Generally, dynamic PE analysis methods use API call telemetry to represent PE activity. Rosenberg et al. [23] construct a one-hot encoded vector out of encountered API calls. This approach is the simplest possible and ignores API sequences. Kolosnjaji et al. [14] showed that it is possible to perform API call processing to preserve sequential information. Another example is described by Yen et al. [33], who obtain a behavioral representation based on API call frequency.

The hybrid analysis allows to surpass the limitations of each malware analysis method, and several security research groups provided insights on utilizing the hybrid approach with ML algorithms. For example, Shijo and Salim [27] document a way to construct a feature vector leveraging data from static and dynamic analysis techniques and processed by a single ML model. On the contrary, Ma et al. [17] use an ensemble of different classifiers to perform a hybrid analysis, similarly to modeling techniques proposed in our work, building separate feature sets from static and dynamic analysis telemetry.

Observing malware in a sandbox is costly in terms of required computational resources and execution time. Therefore, it is hard to collect dynamic analysis telemetry in quantities beneficial for most ML algorithms, especially based on deep learning architectures, with conclusions that generalize well across the true distribution of malicious sample properties. For instance, Shijo and Salim [27] evaluate their technique on 997 virus and 490 clean files, Ma et al. [17] use 282 samples, Kolosnjaji et al. [14] have a dataset of 4753 executables, Yen et al. [33] use 4519 files.

Emulators do not require to mobilize full-fledged operating system operations, as they allow getting vast amounts of telemetry reasonably fast, without the need for virtualization infrastructure. Therefore, the utilization of emulators as a telemetry source of ML models for dynamic malware analysis research is promising yet not commonly adopted. To our knowledge, the first occurrence of emulator utilization for system call collection was reported by Athiwaratkun and Stokes in 2017 [4]. Their model resembles recurrent schemes used in Natural Language Processing (NLP). This work is further developed by Agrawal et al. [2] who present a similar architecture adopted for arbitrary long API call sequences acquired with the help of an emulator. Mandiant's data science team performs promising research with emulation-based dynamic analysis. Specifically, Li et al. [16] provides an extended abstract that reports utilization of emulator [18] for hybrid analysis with architecture similar to ours.

The sparsity of work on emulation-based behavior analysis is due to their limitations. Emulation is an abstraction on top of the operating system where the emulator runs, and no direct interaction with hardware happens. Theoretically, the perfect emulator could spoof the logic behind any system call. Nonetheless, kernels like Windows NT incorporate massive functionality, yielding the implausible achievement of one-to-one replicas. Hence real-world emulators implement

[1] https://github.com/dtrizna/quo.vadis

only a subset of all possible kernel manipulations, and sophisticated malware samples can identify a limited emulation environment, preventing detailed behavior analysis.

We report a detailed error rate and data diversity comparison to related virtualization-based work datasets in Section 3.2. We argue that modern Windows kernel emulation has a vast potential in ML malware detectors based on empirical evidence. Emulation reports produce rich and diverse telemetry, bypassing static analysis limitations. It contains a sequence of kernel API calls invoked by the executable and describes manipulations with files or registry entries and attempted network communications.

3 Dataset

The functionality of a hybrid solution presented in this work is based on input data consisting of both (a) armed PE files suitable for dynamic analysis *and* (b) contextual filepath information. The necessity to acquire contextual data yields impossible the utilization of public data collections since for *every* data sample we need to possess both raw PE bytes *and* filepath data on an in-the-wild system.

To the best of our knowledge, none of the known datasets provide contextual information about PE samples with filepath values at the moment of file execution. For example, Kyadige and Rudd et al. [15] rely on a proprietary Sophos' threat intelligence feed and do not release their dataset publicly. The private nature of contextual data is understandable since such telemetry would contain sensitive components, like directories on personal computers.

Therefore, we partner with an undisclosed security vendor for a vast dataset collection, containing both raw PE files and filepaths of samples from personal customer systems that would resemble an up-to-date threat landscape. Data is treated with respect to the privacy policy accepted by all customers. Therefore, we do not publicly release the file path and raw PE dataset. Sensitive data components like usernames or custom environment variables are from telemetry during the pre-processing stage and have not been resembled within model parameters or emulation reports.

3.1 Dataset structure

We collect the dataset in two sessions. The first session forms the foundation of our analysis, consisting of **98 966** samples, `329 GB` of raw PE bytes. 80% of this corpus is used as a fixed training set, and 20% form an in-sample validation set. We pre-train models and investigate our hybrid solution configuration using this data.

The second dataset acquisition session occurred *three months later*, forming an out-of-sample test set from **27 500** samples, about `100 GB` of data. This corpus is used to evaluate the real-world utility of the hybrid model and investigate model behavior on the evolved malevolent landscape.

The PE files in the dataset are tagged by a professional threat intelligence team, utilizing manual and automated reverse engineering tools operated by the malware analysts. The dataset spans seven malware families and benignware, with detailed distribution parameters described in Table 1. All labels except "*Clean*" represent malicious files. Therefore, we collected relatively more "Clean" samples to balance malicious and benign labels in the dataset.

Table 1. Dataset structure and size.

	Train & valid. sets		Test set	
File label	**Size (Gb)**	**Counts**	**Size (Gb)**	**Counts**
Backdoor	30.0	11089	7.4	2500
Clean	127.0	26061	47.0	10000
Coinminer	46.0	10044	11.0	2500
Dropper	36.0	11275	9.0	2500
Keylogger	34.0	7817	9.8	2500
Ransomw.	14.0	10014	4.6	2500
RAT	5.5	9537	2.5	2500
Trojan	40.0	13128	7.1	2500
Total	329	98966	98	27500

Since most malware is compiled as x86 binaries, we focus on 32-bit (x86) images and deliberately skip the collection of 64-bit (x64) images to maintain homogeneity and label balance of the dataset. Furthermore, malware authors prefer x86 binaries because of Microsoft backward compatibility, which allows to execution of 32-bit binaries on a 64-bit system, but not vice versa. The dataset is formed out of executables (`.exe`), and we intentionally omit library PE files (`.dll`).

3.2 Sample emulation

All the samples represented in Table 1 were processed with a Windows kernel emulator. We utilize Speakeasy [18] Python-based emulator released and actively maintained by Mandiant under MIT license. The Speakeasy version used in our tests is 1.5.9. It relies on QEMU [6] CPU emulation framework. We obtained **108204** successful emulation reports, with a mean runtime of 12.23 seconds per report, emulating 90857 samples from training and validation sets, and 17347 samples in the test set.

Unfortunately, some sample emulations were erroneous, primarily due to an invalid memory read of write assembly instructions. However, another common reason for emulation errors is a call of unsupported API function or anti-debugging techniques. Figure 1 shows the error rate across different malware families.

Speculatively, one of the potential drawbacks of the emulated dataset might be its relative sparsity if compared to the live execution of samples in a sandbox. However, empirical evidence shows that our dataset is more diverse than reported by other groups performing similar data acquisition

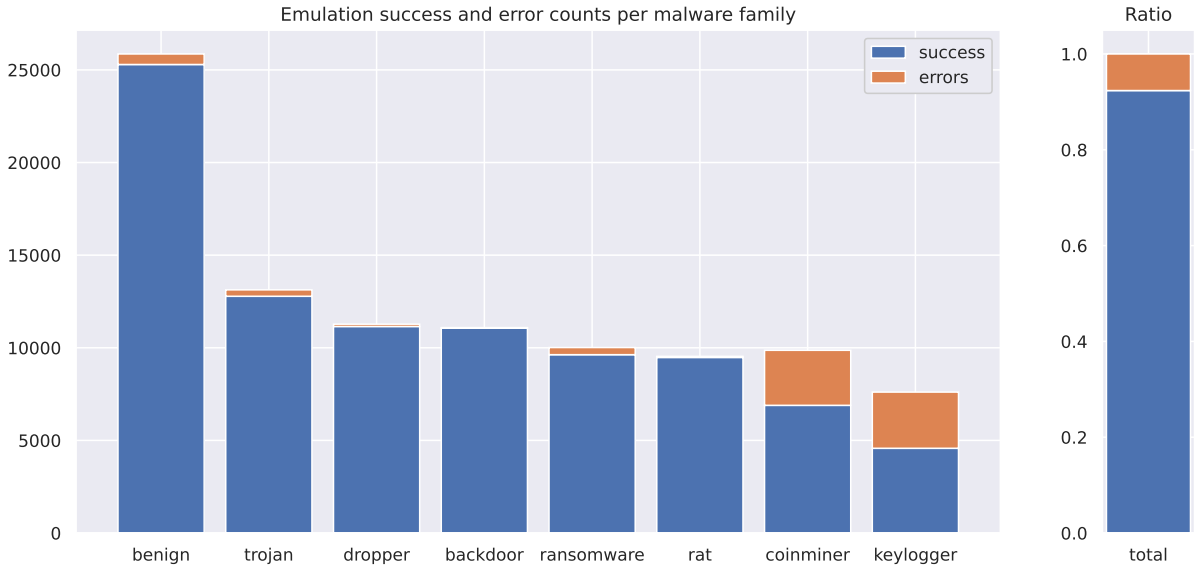

Figure 1. PE emulation error distribution across malware families in in-sample training and validation sets.

Table 2. Dataset diversity preserved and in-sample validation set's F1-score based on choice of top API calls.

Top API calls	100	150	200	300	400	500	600	700
Dataset %	95.53	97.67	98.73	99.48	99.74	99.85	99.91	99.94
Val. F1-score	0.9707	0.9712	0.9725	0.9740	0.9752	0.9747	**0.9759**	0.9754

using full Windows system virtualization. For instance, we acquire **2822** unique API calls within training and validation dataset reports. This behavior is significantly more heterogeneous than in related work datasets - Athiwaratkun and Stokes [4] have a total of 114 unique API calls, Kolosnjaji et al. [14] report 60 unique API calls, Yen et al. [33] have 286 different API calls, Rosenberg et al. [23] have 314 individual API calls. Partially such observation can be described by the larger volume of our dataset since the number of unique calls positively correlates with the number of samples. However, we emphasize this as evidence of emulation technique efficiency. Emulation reports produce rich and diverse telemetry, equivalent by quality to sandboxing for dynamic analysis purposes.

4 System Architecture

A general overview of the hybrid model architecture is visualized in Figure 2. The composite architecture consists of multiple independent models ϕ, that are "fused" together for a final decision produced by a meta-model ψ. Three early fusion models ϕ are:

- file-path 1D convolutional neural network (CNN), ϕ_{fp}
- emulated API call sequence 1D CNN, ϕ_{api}
- FFNN model processing Ember feature vector, ϕ_{emb}

Each early fusion model reports a 128-dimensional vector of representations acquired from input data. Furthermore, all three models' outputs are concatenated together forming an 384-dimensional vector. Therefore, given an input sample x, consisting of raw PE as bytes and its filepath as string, early fusion pass collectively is denoted as:

$$\phi(x) = [\phi_{fp}, \phi_{api}, \phi_{emb}] \in [0,1]^{384}$$

The intermediate vector $\phi(x)$ is passed to a meta-model ψ, which produces the final prediction:

$$\hat{y} = \psi(\phi(x)) \in [0,1]$$

All early fusion networks are pre-trained separately with detailed definition of model configurations and training process in Section 4.3. We want to emphasize that decision to construct a modular system with multiple individually pre-trained components instead of building a single end-to-end trainable architecture is deliberate. First of all, it is shown by Yang et al. [32] that composite neural networks with a high probability surpass the performance of individual pre-trained components.

However, the main reason is the vast potential for expanding a hybrid decision heuristic with complementary modules by only retraining a meta-model. Malicious activity classification problem relies on highly heterogeneous

information sources, beyond raw PE bytes, and such architecture preserves ability on adding heuristics that rely on system logging. At the moment of this publication we already incorporate filepath information, that can be acquired, for instance, from Sysmon[2] telemetry. However, knowledge from within Sysmon data or Speakeasy reports can be extracted further.

4.1 API Call Preprocessing

To acquire a numeric value of API call sequences, we select the top most common calls based on variable vocabulary size V. Preserved API calls are label-encoded, and calls that are not part of the vocabulary are replaced with a dedicated label. The final sequence is truncated or padded using a padding label to a fixed length N.

Table 2 shows the statistics behind dataset diversity and respective model performance. Even though 100 most common calls contain more than 95% API calls within a dataset, experiments show that the model still benefits from relatively large vocabulary size values, so we have chosen $V = 600$ for our final configuration. This observation might be explained by the distribution of API calls per sample. Verbose executables with hundreds of calls bias system call frequency, whereas executables with modest API sequences perform more unique function combinations.

4.2 Path Preprocessing

The first part of path pre-processing includes path normalization since some parts of filepath semantics have variability that is irrelevant for security analysis through the deep learning model. These include specific drive letters or network location if a universal naming convention (UNC) format is used, as well as individual usernames. Therefore, during normalization, we introduced universal placeholders for those path components as presented below:

```
[drive]\users\[user]\desktop\04-ca\8853.vbs
[drive]\users\[user]\appdata\local\file.tmp
[net]\company\priv\timesheets\april2021.xlsm
```

Additionally, it is necessary to parse Windows environment variables to resemble the actual filepath rather than the environment alias used as a variable name. Therefore, we built a variable map consisting of about 30 environment variables that represent specific paths on a system and are used across contemporary and legacy Windows systems. Some examples of a variable map are presented below:

```
r"%systemdrive%": r"[drive]",
r"%systemroot%": r"[drive]\windows",
r"%userprofile%": r"[drive]\users\[user]"
...
```

We perform the encoding of unique letters against the UTF-8 character set. A similar approach was used by Saxe et al. [25] when evaluating URL maliciousness, and by Kyadige and Rudd et al. [15] on a filepath data, using 100 and 150 most frequent UTF-8 bytes, respectively. Rare characters below a frequency threshold are discarded and replaced by a single dedicated label.

4.3 Early Fusion Model Architectures

As mentioned in Section 3, we cannot rely on publicly released malware collections since the model requires contextual information in the form of filepaths at the moment of execution, which is not available publicly. Therefore, to preserve the ability of assessment model performance in comparison to existing research, we can rely only on malware classification models released with pre-trained parameters by other research groups, with further evaluation of those on our dataset. Unfortunately, none of hybrid or dynamic analysis publications [14, 17, 23, 27, 33] provide such artifacts.

Luckily, multiple static analysis publications were accompanied by artifacts in the form of the code repositories [3, 21, 24]. For instance, it is possible directly use Ember LightGBM [12] model, pre-trained on Ember dataset [3] as released in 2019 by Endgame[3]. However, we do not include this model in our composite solution since the decision tree model does not learn representations that can be used by the meta-model, providing only final prediction in scalar form. We still rely on Ember feature extraction scheme [3], but use a FFNN as published by Rudd et al. [24] with three hidden layers or 512, 512, and 128 hidden neurons respectively, all using ELU [9] non-linearity, with layer normalization [5] and dropout [30] rate $p = 0.05$. We retrained FFNN for 200 epochs on 600k feature vectors from Ember training set [3] and 72k samples from our training set.

The analysis of file path and API call sequences can be formulated as a related optimization problem, namely the classification of a 1-dimensional (1D) sequence. We utilize similar neural architecture in both models influenced by Kyadige and Rudd et al. [15], namely embedding layer with a 1D convolutional neural network (CNN) for representation extraction and a fully connected neural network learning classifiers function. We are aware of multiple choices to model sequence classification problems with alternate architectures, such as recurrent neural networks (RNN) [8, 10]. However, as shown in related work on API call sequence classification, both model architectures report similar performance [14, 23], yet it is shown that 1D CNN is significantly less computationally demanding [34].

Encoded input vector x with fixed length N is provided to embedding layer with dimensions H and vocabulary size V. These parameters are subject to hyperparameter optimization. The optimal values for file path model obtained by hyperparameter optimization on validation set are: input

[2] https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

[3] https://github.com/endgameinc/malware_evasion_competition

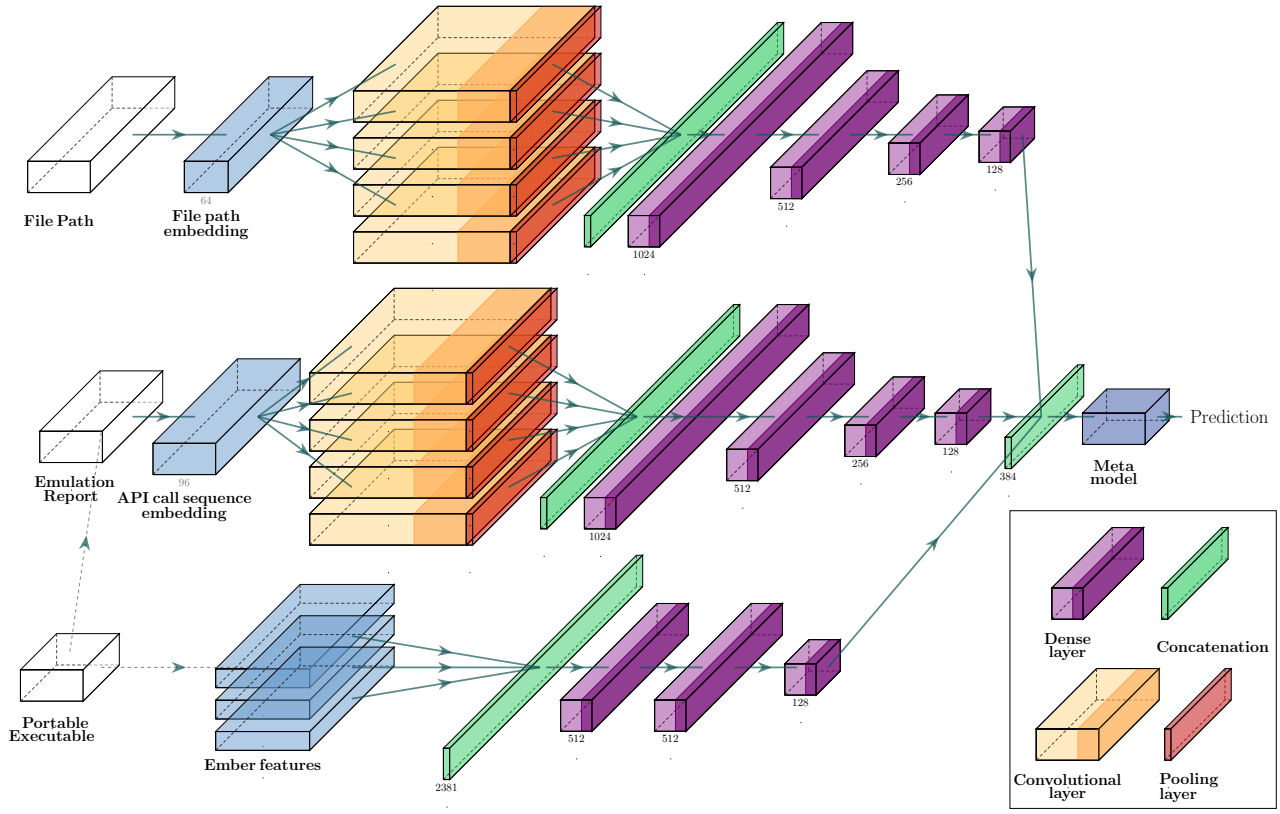

Figure 2. General view of hybrid model architecture with three separate modules.

vector x_{fp} length $N = 100$, embedding dimension $H = 64$, vocabulary size $V = 150$. Respective values for emulated API call sequence model are: input vector x_{em} length $N = 150$, embedding dimension $H = 96$, and vocabulary size $V = 600$. The vocabulary of the file path model is formed out of the most common UTF-8 bytes, and for the API call sequences model, the most common system calls are selected. Both vocabularies are enriched with two labels used for padding and rare characters.

The output of the embedding layer is passed to four separate 1D convolution layers with kernel sizes of 2, 3, 4, and 5 characters, and the number of output channels $C = 128$. With lower C values model underperforms. For instance, having $C = 64$ file path's module validation set F1-score is as low as 0.962, while with $C \in \{100, 128, 160\}$ scores plateau around 0.966.

The output of all four convolution layers is concatenated to a vector of size $4 \times C$ and passed to a FFNN with four hidden layers holding 1024, 512, 256, and 128 neurons. Hidden layers of FFNN are activated using rectified linear unit (ReLU) [1]. The final layer uses a sigmoid activation. Batch normalization [11] is applied to hidden layers of FFNN before the ReLU activation. Additionally, to prevent overfitting, dropout [30] with a $p = 0.5$ rate is applied.

All early fusion networks are fitted using binary cross-entropy loss function:

$$L(x, y; \theta) = -y \log(\phi(x; \theta)) + (1 - y) \log(1 - \phi(x; \theta)).$$

$\phi(x; \theta)$ denotes function approximated by deep learning model given parameters θ, and $y \in \{0, 1\}$ are the ground-truth labels. Optimization is performed using Adam optimizer [13] with 0.001 learning rate and fixed batch size of 1024 samples. We constructed both 1D convolutional networks, Ember FFNN, and training routine using PyTorch [19] deep learning library.

4.4 Meta-Model

The output of early fusion models $\phi(x)$ is used to train the meta-model ψ. Three different architectures types were evaluated, Logistic Regression and FFNN were implemented using `scikit-learn` library [20], and gradient boosted decision tree classifier based on `xgboost` [7] implementation.

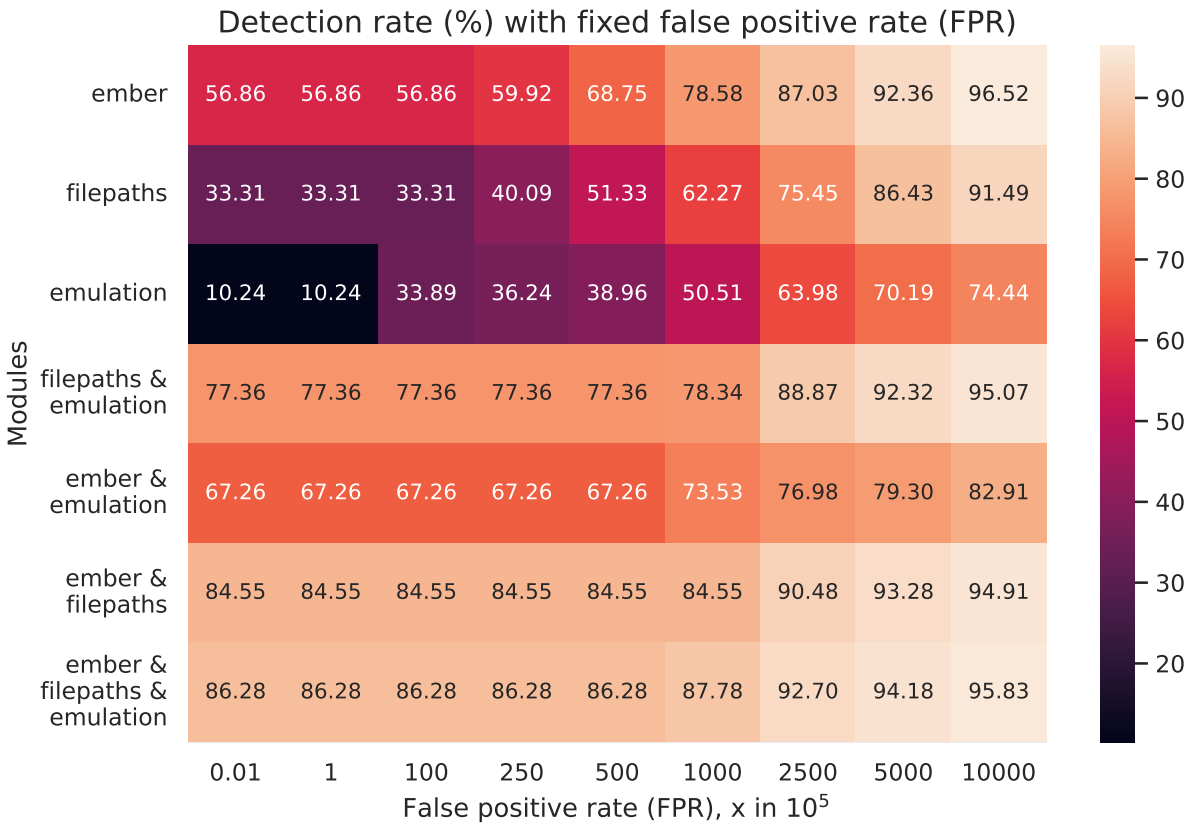

Figure 3. Detection rate (%) on out-of-sample test set with fixed false positive rate based on different combinations of enabled modules in hybrid solution.

Table 3. In-sample validation set metrics assessed against various meta-model architectures.

Model	AUC	F1-score	Recall	Precision	Accuracy	Convergence time
LR	**0.9987**	0.9898	0.9876	0.9920	0.9853	2.52 s
GBDT	0.9986	0.9864	0.9867	0.9862	0.9803	34.92 s
FFNN, 2 layers	0.9973	0.9903	0.9884	0.9923	0.9860	14.30 s
FFNN, 3 layers	0.9965	0.9900	0.9882	0.9918	0.9855	24.52 s
FFNN, 4 layers	0.9957	**0.9904**	0.9881	**0.9926**	**0.9861**	57.48 s
FFNN, 5 layers	0.9954	0.9901	**0.9889**	0.9913	0.9857	36.73 s

Since meta-model ψ performs a relatively complex non-linear mapping $[0,1]^{384} \rightarrow [0,1]$, based on performance metrics from Table 3 we conclude that the fused classification surface is not smooth and presents combinations that utilizes representations from all three feature extraction methods for final decision, which simple models like Logistic Regression are not able to learn. We selected a four layer FFNN with 384, 128, 64, and 16 neurons as a meta-model for our final evaluations since it has close to optimal scores.

5 Experiments and Results

Experiments show that simultaneous utilization of static, dynamic, and contextual information yields significantly better detection rates than individual model performance, especially under low false positive requirements. Such demands are commonly expressed toward machine learning solutions in the security industry. Solutions that do not match low false positive needs are often not allowed to produce alerts for human analysts [22].

Detection rates (%) given fixed false-positive rate (FPR) for the out-of-sample test set are visualised in Figure 3. For

Table 4. Hybrid solution final metrics on validation and test sets with enabled Ember FFNN, API call sequence, and filepath modules using decision threshold of meta-model 0.98.

Metric	Valid. set	Test set
F1-score	0.9900	0.9483
Recall	0.9865	0.9167
Precision	0.9934	0.9822
Accuracy	0.9855	0.9459
AUC	0.9847	0.9485

instance, setting an alert threshold with FPR of only 100 misclassifications in 10^5 cases, individual model detection rates are 56.86%, 33.31%, and 33.89% for Ember FFNN, filepath, and emulation models respectively. However, by combining representations learned from all three processing techniques, the hybrid solution can correctly classify 86.28% from all samples in the test set collected three months after training.

Surprising observations produce filepath and emulation models. Both models individually perform relatively poorly, especially if compared to ember FFNN. A potential explanation behind this observation is the ember FFNN training set consisting of 600k feature vectors from the original Ember publication [3]. Such training corpora produce a much better generalization of "true" malicious PE distribution than our 100k samples reflecting threat landscape in a specific time window.

However, under low false-positive requirements, just a combination of filepath and emulation model, omitting static analysis, outperforms the state-of-the-art Ember feature extraction scheme trained on a much broader dataset, with detection rates 77.36% versus 55.86% given FPR of one sample in 10^5.

Moreover, combining both models result in a detection rate above the cumulative capabilities of individual models, highlighting the hybrid meta-model's superiority over narrow solutions even more. For example, while individual filepath and emulation models detect only 33.31% and 10.24% of samples with FPR of one false alert in 10^5, a combination of them produces a 77.36% detection rate.

This observation holds across both in-sample validation and out-of-sample test sets, collected from divergent systems and in different time frames, allowing us to conclude that this is a general attribute of a hybrid detection heuristic with meta-model rather than an artifact of a specific dataset. Values for in-sample validation set given one misclassification in 10^5 cases are detection rates of 34.46%, 13.52%, and 97.25% for filepath, emulation, and combined heuristics, respectively.

This observation allows to conclude that the meta-model can learn patterns typical to malicious samples out of representation combinations produced by different analysis techniques, like combining a specific API call sequence and filepath n-gram. Each of these representations separately does not produce enough evidence to classify the sample as malicious since it also occurs in benign applications. Therefore, detection happens only by lifting false-positive requirements when both benign and malicious samples are flagged. However, a combination of representations from both filepaths and API calls allows for the meta-model to build a decision boundary in 384-dimensional space to segregate such cases, thus yielding detection rates of more than 40%.

As a result, we see that composite utilization of static, dynamic, and contextual data addresses independent method weaknesses, allowing minimization of FPR and false-negative rates (FNR).

F1-score, Precision, Recall, Accuracy, and AUC scores on all sets are reported in Table 4 with meta-model decision threshold 0.98 that resembled a FPR $\approx$ 0.25% on validation set. While reported results on an in-sample validation set allow concluding that model has little to no overfitting, we still observe a decrease of an out-of-sample test set F1 and AUC scores by $\approx 4-4.5\%$. A drop in detection scores happens despite the same ratio of malware families, and we assume this phenomenon's causality arises from the evolving nature of malevolent logic.

6 Conclusion and Future Work

This work presents a hybrid machine learning architecture that employs the Windows portable executable's static, behavioral, and contextual properties. We performed behavioral analysis on large corpora of executables collected from in-the-wild systems and labeled by a professional threat intelligence team.

We have shown that ML algorithms benefit from hybrid analysis, yielding improved performance, especially under low false-positive requirements. We indeed report exceptional performance by the current state-of-the-art malware modeling scheme based on Ember feature vector [3], which reports detection rates significantly higher than filepath or emulation models individually, as seen in Figure 3. However, combining the Ember model with either filepath or emulation, or both models notoriously improve detection capabilities, under some circumstances like low false positive requirements by almost 30%.

Additionally, we report that a hybrid solution can detect a malevolent sample even if none of the individual components express enough confidence to classify input as malicious. For instance, with FPR of one misclassified case in 10^5, individual filepath and emulation models detect only 33.31% and 10.24% samples. A combination of them produces a 77.36% detection rate, increasing the detection abilities by more than 40% if both models were used together but independently.

This observation holds across in-sample validation and out-of-sample test sets, collected from divergent systems and in different time frames, allowing us to conclude that this is a property of a hybrid solution rather than an artifact

of a specific dataset. We conclude that the meta-model can learn patterns typical to malicious samples out of representation combinations produced by different analysis techniques. Furthermore, this conclusion is supported by the dataset size, significantly larger than in related works on behavioral malware analysis.

We suggest that the positive traits of dynamic and contextual analysis can be extended further. While we represent PE behavior on a system with an API call sequence, not all executable functionality is expressed through API calls. Additional visibility sources might be crucial to minimize ambiguity in the model decision heuristic, and we argue that extending the modularity of composite solutions is a promising research direction. Most of the emulation telemetry was omitted from our analysis.

We publicly release emulated reports of 108204 samples and expect further work in this direction. File system and registry modifications, network connections, and memory allocations may provide crucial information for detection. The architecture of our solution allows us to extend the modularity of the decision heuristic with minimal effort by retraining only the parameters of a meta-model.

Acknowledgments

We express a deep gratitude for Marlon Tobaben, Antti Honkela and Maria Regaki for contributing towards final version of this publication.

References

[1] Abien Fred Agarap. 2019. Deep Learning using Rectified Linear Units (ReLU). arXiv:1803.08375 [cs.NE]

[2] Rakshit Agrawal, Jack W. Stokes, Mady Marinescu, and Karthik Selvaraj. 2018. Robust Neural Malware Detection Models for Emulation Sequence Learning. arXiv:1806.10741 [cs.AI]

[3] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv:1804.04637 [cs.CR]

[4] Ben Athiwaratkun and Jack W. Stokes. 2017. Malware classification with LSTM and GRU language models and a character-level CNN. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ICASSP, New Orleans, LA, USA, 2482–2486. https://doi.org/10.1109/ICASSP.2017.7952603

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. https://doi.org/10.48550/ARXIV.1607.06450

[6] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference* (Anaheim, CA) *(ATEC '05)*. USENIX Association, USA, 41.

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. https://doi.org/10.48550/ARXIV.1412.3555

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). https://doi.org/10.48550/ARXIV.1511.07289

[10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation* 9 (12 1997), 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735

[11] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167 [cs.LG]

[12] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., Long Beach, CA. https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[13] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[14] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. 2016. Deep Learning for Classification of Malware System Call Sequences. In *AI 2016: Advances in Artificial Intelligence*, Vol. 9992. 29th Australasian Joint Conference, Hobart, TAS, Australia, 137–149. https://doi.org/10.1007/978-3-319-50127-7_11

[15] Adarsh Kyadige, Ethan M. Rudd, and Konstantin Berlin. 2020. Learning from Context: A Multi-View Deep Learning Architecture for Malware Detection. In *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, San Francisco, CA, USA, 1–7. https://doi.org/10.1109/SPW50608.2020.00018

[16] Xigao Li, David Krisiloff, and Scott Coull. 2021. Lightweight, Emulation-Assisted Malware Classification.

[17] Xinjian Ma, Qi Biao, Wu Yang, and Jianguo Jiang. 2016. Using multi-features to reduce false positive in malware classification. In *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*. IEEE Information Technology, San Francisco, CA, USA, 361–365. https://doi.org/10.1109/ITNEC.2016.7560382

[18] Mandiant. 2021. Speakeasy: portable, modular, binary emulator designed to emulate Windows kernel and user mode malware. https://github.com/mandiant/speakeasy.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., pytorch.org, 8024–8035.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[21] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2017. Malware Detection by Eating a Whole EXE. arXiv:1710.09435 [stat.ML]

[22] Edward Raff, Bobby Filar, and James Holt. 2020. Getting Passive Aggressive About False Positives: Patching Deployed Malware Detectors. https://doi.org/10.48550/ARXIV.2010.12080

[23] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. 2020. Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers. In *Annual Computer Security Applications Conference* (Austin, USA) *(ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 611–626. https://doi.org/10.1145/3427228.3427230

[24] Ethan M. Rudd, Felipe N. Ducau, Cody Wild, Konstantin Berlin, and Richard Harang. 2019. ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation. arXiv:1903.05700 [cs.CR]

[25] Joshua Saxe and Konstantin Berlin. 2017. eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys. arXiv:1702.08568 [cs.CR]

[26] Matthew Schultz, Eleazar Eskin, F. Zadok, and Salvatore Stolfo. 2001. Data Mining Methods for Detection of New Malicious Executables, In Proceedings 2001 IEEE Symposium on Security and Privacy. *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* 1, 1, 38–49. https://doi.org/10.1109/SECPRI.2001.924286

[27] P.V. Shijo and A. Salim. 2015. Integrated Static and Dynamic Analysis for Malware Detection. *Procedia Computer Science* 46 (2015), 804–811. https://doi.org/10.1016/j.procs.2015.02.149 Proceedings of the International Conference on Information and Communication Technologies, ICICT 2014, 3-5 December 2014 at Bolgatty Palace & Island Resort, Kochi, India.

[28] Rami Sihwail, Khairuddin Omar, and KA Zainol Ariffin. 2018. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *Int. J. Adv. Sci. Eng. Inf. Technol* 8, 4-2 (2018), 1662–1671.

[29] Anuj Soni and Lenny Zeltser. 2021. FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[31] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81 (2019), 123–147.

[32] Ming Chuan Yang and Meng Chang Chen. 2019. Theoretical Investigation of Composite Neural Network. arXiv:1910.09351 [cs.LG]

[33] Yao Saint Yen, Zhe Wei Chen, Ying Ren Guo, and Meng Chang Chen. 2019. Integration of Static and Dynamic Analysis for Malware Family Classification with Composite Neural Network. arXiv:1912.11249 [cs.CR]

[34] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative Study of CNN and RNN for Natural Language Processing. https://doi.org/10.48550/ARXIV.1702.01923