Learning from Context: A Multi-View Deep Learning Architecture for Malware Detection

Adarsh Kyadige*, Ethan M. Rudd † and Konstantin Berlin‡
Sophos AI
Reston, Virginia
Email: *adarsh.kyadige@sophos.com, †ethan.rudd@fireeye.com, ‡konstantin.berlin@sophos.com

***Abstract*—Machine learning (ML) classifiers used for malware detection typically employ numerical representations of the content of each file when making malicious/benign determinations. However, there is also relevant information that can be gleaned from the *context* in which the file was seen which is often ignored. One source of contextual information is the file's location on disk. For example, a malicious file masquerading as a known benign file (e.g., a Windows system DLL) is more likely to appear suspicious if the detector can intelligibly utilize information about the path at which it resides. Knowledge of the file path information could also make it easier to detect files which try to evade disk scans by placing themselves in specific locations. File paths are also available with little overhead and can seamlessly be integrated into a multi-view static ML detector, potentially yielding higher detection rates at very high throughput and minimal infrastructural changes.**

In this work, we propose a multi-view deep neural network architecture, which takes feature vectors from the PE file content as well as corresponding file paths as inputs and outputs a detection score. We perform an evaluation on a commercial-scale dataset of approximately 10 million samples – files and file paths from user endpoints serviced by an actual security vendor. We then conduct an interpretability analysis via LIME modeling to ensure that our classifier has learned a sensible representation and examine how the file path contributes to change in the classifier's score in different cases. We find that our model learns useful aspects of the file path for classification, resulting in a 26.6% improvement in the true positive rate at a 0.001 false positive rate (FPR) and a 64.6% improvement at 0.0001 FPR, compared to a model that operates on PE file content only.

***Index Terms*—Static PE Detection, File Path, Deep Learning, Multi-View Learning, Model Interpretation**

I. Introduction

Commercial Portable Executable (PE) malware detectors consist of a hybrid of static and dynamic analysis engines. Static detection – which is fast and effective at detecting a large fraction of malware – is usually first employed to flag suspicious samples. It involves analyzing the raw PE image on disk and can be performed very quickly, but it is vulnerable to code obfuscation techniques, e.g., compression and polymorphic/metamorphic transformation [1].

Dynamic detection, by contrast, requires running the PE in an emulator and analyzing behavior at run time [2]. When dynamic analysis works, it is less susceptible to code obfuscation, but takes substantially greater computational capacity and time to execute than static methods. Moreover, some files are difficult to execute in an emulated environment, but can still be statically analyzed. Consequently, static detection methods are typically the most critical part of an endpoint's malware prevention (blocking malware before it executes) pipeline.

Static detection methods have seen performance advancements recently, thanks to the adoption of machine learning [3], where highly expressive classifiers, e.g., deep neural networks, are fit on labeled data sets of millions of files. When these classifiers are trained, they use the static file content as input but no auxiliary data. We note, however, that dynamic analysis works well *precisely because of auxiliary data* – e.g., network traffic, system calls, etc.

In this work, we seek to use file paths as orthogonal input information to augment static ML detectors. File paths are available statically, without any additional instrumentation of the OS. By including the file path as an auxiliary input, we expect to be able to combine information about the file with information about how likely it is to see such a file in that specific location, and identify common directory hierarchies and naming patterns associated with known malware and benign files.

We focus our analysis on three models:

- The baseline file content only (*PE*) model, which takes only the PE features as input and outputs a malware confidence score.
- Another baseline file path content only (*FP*) model, which takes only the file's file paths as input and outputs a malware confidence score.
- Our proposed multi-view PE file content + contextual file path (*PE+FP*) model, which takes in both the PE file content features and file paths, and also outputs a malware confidence scores.

A schematic of the three models is shown in Figure 1.

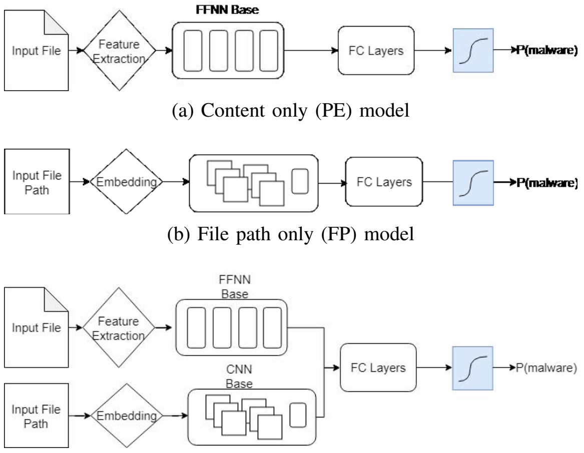
We conduct our analysis on a time-split dataset collected from a large anti-malware vendor's telemetry, and find that our classifier trained on both file content and the contextual file path yields statistically significantly better results across the ROC curve and particularly in low false positive rate (FPR) regions.

The contributions of this paper are as follows:

1) We obtain a realistic carefully curated data set of files and file paths from a security vendor's customer end-

† Work for this paper was performed while at Sophos.
A. Kyadige* and E. M. Rudd† contributed equally to this work

DOI 10.1109/SPW50608.2020.00018



(a) Content only (PE) model

(b) File path only (FP) model

(c) Multi-view (PE + FP) model

Fig. 1: Schematic of the three approaches that we compare in this paper. (a) A baseline conventional deep learning model for PE malware detection makes malicious/benign determination using a numeric summary of file content as input. (b) A contextual baseline which utilizes convolutions over just the file path at which the PE resides. (c) Our novel multi-view architecture, which fuses representations derived from both a content input and a contextual input to make its malicious/benign determination.

points (rather than a malware / vendor label aggregation service).

2) We demonstrate that our multi-view PE+FP malware classifier performs substantially better on our dataset than a model that uses the file contents alone.
3) We extend Local Interpretable Model Agnostic Explanations (LIME) [4] to our PE+FP model, and use it to analyze how file paths contribute to a model's malware/benign decision.

The remainder of this manuscript is structured as follows: Section II covers important background concepts and related work. Section III discusses data set collection and model formulation. Section IV presents an evaluation comparing our novel multi-view approach to a baseline content-only model of similar topology and an interpretability analysis of our model. Section V concludes.

II. Background and Related Work

In this section, we describe how machine learning is commonly applied to static PE detection and how our approach differs, in a high level sense, by providing contextual information as an auxiliary input. We then present related work in other machine learning domains.

A. *Static ML Malware Detection*

Machine learning has been applied in the computer security domain for many years now [5], but disruptive performance breakthroughs in static PE models using ML at the commercial scale are a more recent phenomenon. Commercial models typically rely on deep neural networks [6] or boosted decision tree ensembles [7] and have been extended to other static file types as well, including web content [8], [9], office documents [10], and archives [10].

Most static ML for information security (ML-Sec) classifiers operate on learned embeddings over portions of files (e.g., headers) [11], learned embeddings over the full file [12], or most commonly, on pre-engineered numerical *feature vectors* designed to summarize the content from each file [6], [13]–[19]. Pre-engineered feature vector representations tend to be more scalable, and quickly distill content from each file while preserving useful information. There are a number of ways to craft feature vectors, including tracking per-byte statistics over sliding windows [6], [18], byte histograms [7], [18], ngram histograms [13], treating bytes as pixel values in an image (a visualization of the file content) [13], [18], opcode and function call graph statistics [18], symbol statistics [18], hashed/numerical metadata values [6], [7], [18] – e.g., entry-point as a fraction of the file, or hashed imports and exports, – and hashes of delimited tokens [10], [19]. In practical applications, several different types of feature vectors extracted from file content are often concatenated together to achieve superior performance.

B. *Learning from Multiple Sources*

Related research in static ML malware detection using deep neural networks has examined learning from multiple sources of information but the approaches are fundamentally different from ours: Huang et al. [20] and Rudd et al. [21] use multi-objective learning [22], [23] over multiple auxiliary loss functions which they found increased performance on the main malware detection task. Both of these works use metadata as auxiliary target labels during training to provide additional information to the model, and use a single input when deployed to make the classification decision.

Our approach utilizes *multiple* input types/modalities – one which describes the content of the malicious sample, in the form of a PE feature vector similar to [6], and another which feeds the raw string of characters to a to a character embedding layer (similar to [8]) which provides information on where that sample was seen. This technique is a type of *multi-view learning* [24]. As the name might suggest, the majority of applications of multi-view learning are in computer vision, where the multiple views *literally* consist of views from different input cameras/sensors or different views from the same camera/sensor at different times.

In the ML-Sec space, we could find only two approaches which specifically reference themselves as *multi-view*: namely [25], in which Narayanan et al. applied multiple kernel learning over dependency graphs for Android malware classification and [26], in which Bai et al. used multi-view ensembles for PE malware detection. While these approaches are in some ways similar to ours, we are the first, to our knowledge, to perform multi-view modeling for malware detection at a commercial scale using exogenous contextual fed to a deep neural network in conjunction with file content features.

III. Implementation Details

In this section we present implementation details of our approach, including the data collection process for obtaining PE files and file paths from customer endpoints, our featurization strategy, and the architectures of our multi-view deep neural network and comparison baselines.

A. Dataset

For our experiments, we collected three distinct datasets from the telemetry of a prominent anti-malware vendor: A training set, a validation set and a test set. The training set comprised of 9,148,143 samples which were first seen between June 1 and November 15 2018, out of which 693,272 samples were labeled malicious. The validation set consisted of 2,225,094 distinct samples seen between November 16 and December 1 2018, out of which 85,041 were labeled as malicious. Finally, the test set had 249,783 total samples, seen between Jan 1 to Jan 30 2019, out of which 38,767 were labeled as malicious.

Malicious/benign labels for these files were computed using a criterion similar to [6], [8], but combined with additional proprietary information to generate more accurate labeling.

B. Feature Engineering

In order to use file paths as input to a neural network model, we first convert the variable length strings into numeric vectors of fixed length. We accomplished this using a vectorization scheme similar to [8], by creating a lookup table keyed on each character with a numeric value (between 0 and the character set size) representing each character. In practice, we implemented this table as a Python dictionary. Guided by statistics from our telemetry and early experimentation, we trimmed file paths to the last 100 characters. Features for file paths shorter than 100 characters were padded with zeroes. For the character set, we consider the entire Unicode character set, but limit the vocabulary to the 150 most frequent characters. See Appendix **??** for further discussion.

As features for the content of the PE files, we used floating point 1024-dimensional feature vectors consisting of four distinct feature types, similar to [6].

In total, we represent each sample as two feature vectors: a PE content feature vector of 1024 dimensions and a contextual file path feature vector of 100 dimensions.

C. Network Architectures

Our multi-view architecture is shown in Figure 2. The model has two inputs, the 1024 element PE content feature vector, $\mathbf{x}_{PE}$, and the 100 element file path integer vector, $\mathbf{x}_{FP}$, as described in Section III-B. Each distinct input is passed through a series of layers with their own parameters, θ_{PE} and θ_{FP}, for PE features and FP for filepath features respectively, and are jointly optimized during training. The outputs of these layers are then joined (concatenated) and passed through a series of final hidden layers – a joint output path with parameters θ_O. The final output of the network consists of a dense layer followed by a sigmoid activation.

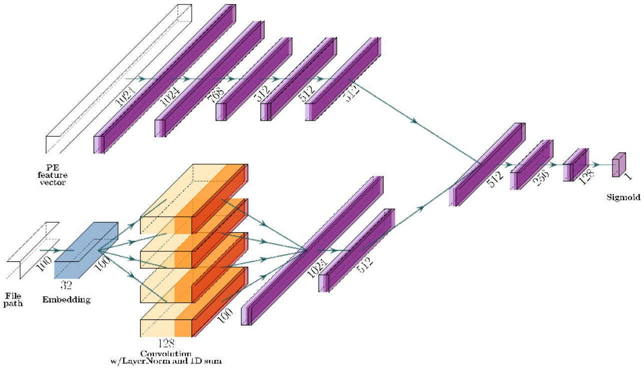


Fig. 2: The neural network model we use in our experiments. Each of the unlabeled blocks contains a fully connected layer, followed by Layer Normalization and a Dropout Layer. In experiments where we train the file paths and PE features individually, the respective input and associated input branch is used and the other branch is removed from the model definition.

The PE input arm θ_{PE} passes $\mathbf{x}_{PE}$ through a series of blocks consisting of four layers each: a Fully Connected layer, a Layer Normalization layer implemented using the technique described in [27], a Dropout layer with a dropout probability of 0.05, and an Rectified Linear Unit (ReLU) activation. Five of these blocks are connected in sequence with dense layer sizes 1024, 768, 512, 512 and 512 nodes respectively in order.

The file path input arm θ_{FP}, passes $\mathbf{x}_{FP}$ – a vector of length 100 – into an Embedding layer that converts each character of the filepath into a 32 dimensional vector, resulting in a 100x32 embedding tensor for the entire filepath. This embedding is then fed into 4 separate convolution blocks, that contain a 1D convolution layer with 128 filters, a layer normalization layer and a 1D sum layer to flatten the output to a vector. The 4 convolution blocks contain convolution layers with filters of size 2, 3, 4 and 5 respectively that process 2, 3 4 and 5-grams of the input file path. The flattened outputs of these convolution blocks are then concatenated and serve as input to two dense blocks of size 1024 and 512 neurons (same form as in the PE input arm).

The outputs from the fully connected blocks from the PE arm and the file path arm are then concatenated and passed into the joint output path, parameterized by θ_O. This path consists of dense connected blocks (same form as in the PE input arm) of layer sizes 512, 256 and 128. The 128D output of these blocks is then fed to a dense layer which projects the output to 1D, followed by a sigmoid activation that provides the final output of the model.

The PE only model is just the PE+FP model but without the FP arm, taking input $\mathbf{x}_{PE}$ and fitting θ_{PE} and θ_O parameters. Similarly, the FP model is the PE+FP model but without the

TABLE I: Mean and standard deviation true positive rates (TPRs) on the test set for false positive rates (FPRs) of interest. Results were aggregated over five training runs with different weight initializations and minibatch orderings. Best results, shown in **bold**, consistently occurred when using both feature vectors from the file and contextual file path as inputs.

		PE+FP	PE	FP
TPR	10^{-5} FPR	**0.398** ± 0.083	0.208 ± 0.086	0.02 ± 0.022
	10^{-4} FPR	**0.558** ± 0.009	0.339 ± 0.059	0.233 ± 0.04
	10^{-3} FPR	**0.693** ± 0.005	0.547 ± 0.007	0.522 ± 0.003
	10^{-2} FPR	**0.922** ± 0.006	0.889 ± 0.008	0.711 ± 0.003
	10^{-1} FPR	**0.978** ± 0.005	0.972 ± 0.007	0.927 ± 0.003
Overall AUC		**0.992** ± 0.001	0.990 ± 0.002	0.968 ± 0.003

PE arm, taking input $\mathbf{x}_{FP}$ fitting θ_{FP} and θ_O paramters. The first layer of the output subnetwork is adjusted appropriately to match the output from the previous layer.

We fit all models using a binary cross entropy loss function. Given the output of our deep learning model $f(\mathbf{x};\theta)$ for input $\mathbf{x}$ with label $y \in \{0,1\}$, and model parameters θ the loss is:

$$L(\mathbf{x}, y; \theta) = -y\log(f(\mathbf{x};\theta)) + (1-y)\log(1-f(\mathbf{x};\theta)). \quad (1)$$

Via an optimizer, we solve for $\hat{\theta}$ the optimal set of parameters that minimize the combined loss over the dataset:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{M} L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}; \theta), \quad (2)$$

where M is the number of samples in our dataset, and $\mathbf{y}^{(i)}$ and $\mathbf{x}^{(i)}$ are the label and the feature vector of the ith training sample respectively.

We built and trained our models with the Keras framework [28], using the Adam optimizer with Keras's default parameters and 1024 sized minibatches. Each model is trained for 15 epochs, which we determined was enough for the results to converge.

IV. Experiments and Analysis

In our experiments, we trained three different types of models: two baseline models (PE and FP) and one multi-view model (PE+FP). To get a statistical view of model performance, we trained five models of each type, with different weight initialization per model, different minibatch ordering, and different seeds for dropout. This allows us to assess not only relative performance comparisons across individual models (as is standard practice), but also mean performance and uncertainty across model types. Training multiple models also tells us important information about the stability of each model type under different initialization.

A. Performance Evaluation

Results for the three model types, evaluated on the test set – PE+FP, PE, and FP – are shown in Figure 3 as ROC curves and are also summarized in tabular form in in Table I. Recall that these results (mean and standard deviation) were assessed over five runs.

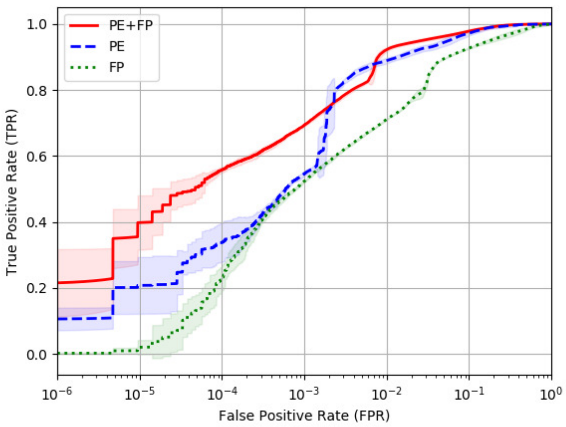


Fig. 3: Mean ROC curves and standard deviations for our PE+FP model (red solid line), a PE model (blue dashed line), and an FP model (green dotted line).

We see that the multi-view (PE+FP) model substantially outperforms the content-only model in terms of net AUC and across the vast majority of the ROC curve, dipping slightly below the PE baseline between 10^{-2} and 10^{-3} FPR, an effect which could potentially be alleviated with a larger training set. At lower FPRs, the performance improvements from the PE+FP model compared to both baselines is substantial. Specifically, we see that there is a 27% increase in True Positive rate for the PE+FP model as opposed to the PE model at 10^{-3} FPR, and a 64% increase at 10^{-4} FPR. This increase is also accompanied by a reduction in variance of performance, making the PE+FP model a better choice in terms of both stability and overall detection performance. As expected, the filepath only (FP) model that looks only at context consistently performs the worst, with an overall mean AUC of 0.968, compared to a mean AUC of 0.992 for the multi-view (PE+FP) model and a mean AUC of 0.990 for the content-only (PE) model.

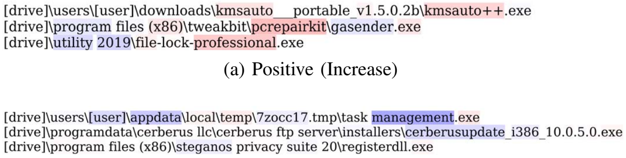
Note that the TPR/FPR metrics that we use to evaluate detection are invariant to the ratio of malicious to benign samples in our test set. When we deploy this model in practice, we can use this TPR/FPR ROC curve to re-calibrate the detector to select a threshold associated with an FPR that we can handle (e.g., 10^{-3}), rather than the presumed default 0.5 threshold. Hence it is encouraging that our model performs well at low FPR regions as well. In practice, we deploy our models at an FPR of 10^{-3} since that aligns with our capacity to manually analyze and correct false positives produced by the model.

At very low FPRs ($< 10^{-4}$) the variance in the TPR increases. This is due to inherent measurement noise at low FPRs: an FPR of 10^{-5} means that $1/100,000$ benign samples were falsely labeled as malicious, which is the same order of magnitude as the number of benign samples in our dataset, providing little support for the numerical interpolation used to generate these ROC curves. However, the improvement of the combined model is still substantially larger than the statistical uncertainty for the relevant 10^{-3} to 10^{-4} FPR regions.

There are two reasons to believe that our test set is more challenging than than real deployment distributions. The first reason is that ML detectors are never deployed by themselves, and are instead guard-railed by signers, prominent file hashes, and AV signature whitelisting. Most of the prominent FP issues can be suppressed using these whitelist approaches. The second reason, is that we removed any previously seen PE file from test set, even it has a new file path. In the raw telemetry, we observed that most executed files are actually not new. Thus, our evaluation reflects the realistic capability of our respective classifiers to detect novel malware.

B. LIME Analysis

To ensure that our multi-view model has learned meaningful content from PE file paths, we pick one of our trained models and employ Local Interpretable Model-Agnostic Explanations (LIME) [4] to samples from the test set. Using LIME, we were able to generate weights for each token in a sample filepath, which represented the impact that the token had on the final classification outcome.



```
[drive]\users\[user]\downloads\kmsauto___portable_v1.5.0.2b\kmsauto++.exe
[drive]\program files (x86)\tweakbit\pcrepairkit\gasender.exe
[drive]\utility 2019\file-lock-professional.exe
```

(a) Positive (Increase)

```
[drive]\users\[user]\appdata\local\temp\7zocc17.tmp\task management.exe
[drive]\programdata\cerberus llc\cerberus ftp server\installers\cerberusupdate_i386_10.0.5.0.exe
[drive]\program files (x86)\steganos privacy suite 20\registerdll.exe
```

(b) Negative (Decrease)

Fig. 4: Example file paths from our LIME analysis with (a) positive and (b) negative ground truth labels. The path tokens are highlighted based on the Lasso weights, as computed by the LIME model. The token weights can be directly interpreted as either making the overall malware score higher (red weights) or lower (blue weights). Darker red and blue shades correspond to greater magnitude weights, while lighter shades correspond to smaller magnitudes. White color corresponds to no impact.

We visualized these computed Lasso model weights for several interesting examples in Figure 4, by overlaying the computed weights on top of the file path string. In the first positive example we can see that that the token "`kmsauto`" is being identified as a maliciousness indicator by our PE+FP model. KMS Auto is legally dubious Microsoft product activator, and this file is identified as "PUA:Win32/AutoKMS" by Microsoft. Similarly, in the second positive example our PE+FP model gave high score to "`pcrepairkit`". Repair kits are often questionable software products that usually contain spyware or malware.

On the other hand, in the several negative examples we can see that management tools are being down-weighted by the PE+FP model, as compared to the PE model. Management tools are notoriously difficult to distinguish from spyware, as their functionality is basically the same, the only difference is the intent of the user. In this case, using filepath information provided us more context for the detection, thus allowing more accurate identification by the PE+FP model. We note that these are a few interesting examples, and that the relative contributions of tokens also have a non-linear dependence on the file content itself. For example, when we kept the same path for the first negative example, but replaced the file with a randomly chosen malicious file, the importance of the token "`management`" was significantly reduced.

Finally, we performed an aggregate LIME analysis to identify prominent tokens throughout our dataset. We performed isotonic regression to calibrate the sigmoid outputs of the PE and the PE+FP model, and identified 200 samples which had the highest score variation between the two models - 100 samples which saw the largest increase in score and 100 samples which saw the largest decrease in score over the baseline. We then aggregated LIME parameter weights across tokens and normalized by token frequency, looking at tokens of highest and lowest weights for the selected 200 samples. The top 10 tokens which increased and decreased response are shown in Table II.

For malicious samples, we see that the tokens of highest weight consisted of strings with randomized content, that were not cryptographic digests, perhaps an attempt at obfuscation. The remaining high-weight token, `setup` is perhaps indicative of an infected installer. Tokens with large negative weights consist of common looking benign software names, as one might expect. Of the benign samples that we assessed, tokens that increased response tended to have very short length, e.g., "`t`", "`d`", and "`z`", very high or very low entropy, e.g., "`219805786`" and "`xxxxx`", and have "miner" in their names, e.g., "`miner`", "`mineropt`" – indicating the likely presence of a (benign) cryptocurrency miner, potentially downloaded by the user voluntarily. It is not surprising that the string "miner" increased response as many types of malware and potentially unwanted benignware steal CPU cycles to mine cryptocurrency. With respect to tokens that most attenuated the response, they appear to be components of standard software. Interestingly, "`setup`" tends to attenuate response for the benignware that we analyzed, indicating that the behavior of tokens depends on their contextual location within the file path. Note that, as LIME involves fitting a classifier per sample, this analysis is limited only to the samples that we analyzed. However, it suggests that our neural network is learning to extract useful contextual information from file paths; not just mere data artifacts.

C. Practical Deployment and Adversarial Attacks

In recent years, some research on the use of Deep Learning models in Malware detection has highlighted that they are vulnerable to adversarial attacks. Adversarial weakness plagues all known detection models, including the original PE content only model that we compare the PE+FP model to in our research, as well as all signature detection methods

TABLE II: Tokens and corresponding weights from our LIME analysis that most amplified and attenuated responses for malicious and benign samples. For the malicious samples analyzed, tokens that resulted in greatest increase and greatest decrease in classification score are shown in (a) and (b). Corresponding tokens for the benign samples are shown in (c) and (d).

(a) Increase (Malicious)

Token	Weight
2786	7.436
4327	5.854
8o0sdtwhrxkz	4.213
28pygyuokzwwn	3.826
wfzctyetugjwxxuy	3.736
3015798005	3.592
setup	3.313
jzljumnkfaapzpqq	3.183
whyovxk3mplt6	3.167
1467	2.219

(b) Decrease (Malicious)

Token	Weight
onv2k	-6.677
computerz	-6.433
westlake	-5.565
editor	-5.13
printingtools	-4.738
videodecodesdk	-3.687
placar80	-3.663
movavistatistics	-3.556
enterprise	-3.488
jarvee	-3.401

(c) Increase (Benign)

Token	Weight
miner	9.369
z	8.163
2639	6.876
mineropt	6.507
2198205786	6.28
systemprofile	4.26
xxxxx	4.193
t	3.916
d	3.812
namespace	3.441

(d) Decrease (Benign)

Token	Weight
msi61f0	-8.04
part	-7.022
ciscosparklauncher	-6.642
sesinaci	-4.738
clientinst	-4.445
safesenderslist	-4.443
setup	-4.389
sd	-4.147
wim	-4.06
ie8shims	-3.996

commonly used by the AV industry. Thus, in this section, we focus on the practical aspects of deploying the PE+FP model, and acknowledge that if an attacker gains white box access to any of our neural network models, then evading them would be trivial at that point and defending against such an attack would be impossible. Our goal for deploying a context model like our PE+FP model is to detect already existing in the wild attacks that are missed by the PE only model, rather than to replace the PE model.

We propose that the PE+FP model be deployed along side the PE model (not instead of the PE model), as well as all other currently existing layers of defense, such as static and dynamic signature-based detection, whitelisting and blacklisting, etc. This ensures that the model is not the only line of defense, and reduces black box access of the model to the attacker.

As part of a layered defense, bypassing the PE+FP model becomes more difficult. There are multiple constraints that the attacker faces when trying to evade a deployed PE+FP model by modifying the file paths. The attacker is often limited by file permissions. In cases where the malicious file needs to impersonate a benign file, modifying the file path is not even an option. Modifying the file path may even make a malicious file less effective, by making it less likely that a user will click on the file, or making it more likely to be included in a disk scan. Additionally, the attacker has to be aware not to trigger alerts in endpoint detection systems while moving malware to arbitrary paths in the file system. Finally, for attackers to be able to find an evasive file path, they would need to query the model several times with the same file and different file paths, which would be easy to catch and block.

In order to evade detection by the PE content only model, the only constraint the attackers have is to ensure that the PE file compiles and is able to execute the malicious code in it. The rest of the file is completely in the attacker's control to modify. However, as we demonstrated in Section IV-B, the PE+FP model learns to dynamically attribute importance to parts of the file path based on the content of the PE file supplied to it, and vice versa. As a result, finding a Universal Bypass for the combined model becomes a lot harder than for the PE content only model, because changing just one of inputs does not consistently change the output, and changing both with a limited query budget to find the right joint bypass is hard.

V. Conclusion

We have demonstrated that deep neural network malware detectors can benefit from incorporating contextual information from file paths, even when this information is not inherently malicious or benign. Adding file paths to our detection model did not require any additional endpoint instrumentation, and provided a statistically significant improvement in the overall ROC curve, throughout relevant FPR regions. The fact that we measured the performance of our models directly on a customer endpoint distribution suggests that our multi-view model can practically be deployed to endpoints to detect malware.

The LIME analysis that we conducted in Section IV-B demonstrates that the multi-view model learns to distill contextual information suggestive of actual malicious/benign concepts; not merely statistical artifacts of the dataset, though as we observed, it can learn such artifacts as well.

In addition to endpoint deployment, another potential area where this research can be applied is in an Endpoint Detection and Response (EDR) context, where the outputs of our model can be used to rank file events on disk based on how suspicious they seem. Interestingly, techniques like LIME also have applications in this context. Using explanations derived from LIME or similar approaches, analytic tools could be created that allow users that are not malware/forensics experts to perform some degree of threat hunting. Importance highlighting, like we illustrated in Figure 4, is not only a useful visualization for the user, but is also an alternative to the nearest neighbor/similarity visualization approach that does not reveal Potentially Identifiable information (PII) of other users.

Acknowledgment

This research was funded by Sophos PLC.

References

[1] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, 2007, pp. 421–430.

[2] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, p. 6, 2012.

[3] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017.

[4] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 1135–1144.

[5] E. Rudd, A. Rozsa, M. Gunther, and T. Boult, "A survey of stealth malware: Attacks, mitigation measures, and steps toward autonomous open world solutions," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 1145–1172, 2017.

[6] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE, 2015, pp. 11–20.

[7] H. S. Anderson and P. Roth, "Ember: An open dataset for training static pe malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.

[8] J. Saxe and K. Berlin, "expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys," *arXiv preprint arXiv:1702.08568*, 2017.

[9] J. Saxe, R. Harang, C. Wild, and H. Sanders, "A deep learning approach to fast, format-agnostic detection of malicious web content," *arXiv preprint arXiv:1804.05020*, 2018.

[10] E. M. Rudd, R. Harang, and J. Saxe, "Meade: Towards a malicious email attachment detection engine," *arXiv preprint arXiv:1804.08162*, 2018.

[11] E. Raff, J. Sylvester, and C. Nicholas, "Learning the pe header, malware detection with minimal domain knowledge," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 121–132.

[12] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] M. Mays, N. Drabinsky, and S. Brandle, "Feature selection for malware classification." in *MAICS*, 2017, pp. 165–170.

[14] M. Hassen, M. M. Carvalho, and P. K. Chan, "Malware classification using static analysis based features," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–7.

[15] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 3854–3861.

[16] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 2017, pp. 239–248.

[17] B. N. Narayanan, O. Djaneye-Boundjou, and T. M. Kebede, "Performance analysis of machine learning and pattern recognition algorithms for malware classification," in *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*. IEEE, 2016, pp. 338–342.

[18] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proceedings of the sixth ACM conference on data and application security and privacy*. ACM, 2016, pp. 183–194.

[19] J. Drew, T. Moore, and M. Hahsler, "Polymorphic malware detection using sequence classification methods," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 81–87.

[20] W. Huang and J. W. Stokes, "Mtnet: a multi-task neural network for dynamic malware classification," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 399–418.

[21] E. M. Rudd, F. N. Ducau, C. Wild, K. Berlin, and R. Harang, "Aloha: Auxiliary loss optimization for hypothesis augmentation," *arXiv preprint arXiv:1903.05700*, 2019.

[22] R. Caruna, "Multitask learning: A knowledge-based source of inductive bias," in *Machine Learning: Proceedings of the Tenth International Conference*, 1993, pp. 41–48.

[23] E. M. Rudd, M. Günther, and T. E. Boult, "Moon: A mixed objective optimization network for the recognition of facial attributes," in *European Conference on Computer Vision*. Springer, 2016, pp. 19–35.

[24] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *arXiv preprint arXiv:1304.5634*, 2013.

[25] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "A multi-view context-aware approach to android malware detection and malicious code localization," *Empirical Software Engineering*, pp. 1–53, 2018.

[26] J. Bai and J. Wang, "Improving malware detection using multi-view ensemble learning," *Security and Communication Networks*, vol. 9, no. 17, pp. 4227–4241, 2016.

[27] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv e-prints*, p. arXiv:1607.06450, Jul 2016.

[28] F. Chollet *et al.*, "Keras," 2015.