2018 IEEE Symposium on Security and Privacy Workshops

HeNet: A Deep Learning Approach on Intel® Processor Trace for Effective Exploit Detection

Li Chen
Security and Privacy Lab, Intel Labs
Hillsboro, OR 97124

Salmin Sultana
Security and Privacy Lab, Intel Labs
Hillsboro, OR 97124

Ravi Sahita
Security and Privacy Lab, Intel Labs
Hillsboro, OR 97124

***Abstract*—This paper presents HeNet, a hierarchical ensemble neural network, applied to classify hardware-generated control flow traces for malware detection. Deep learning-based malware detection has so far focused on analyzing executable files and runtime API calls. Static code analysis approaches face challenges due to obfuscated code and adversarial perturbations. Behavioral data collected during execution is more difficult to obfuscate but recent research has shown successful attacks against API call based malware classifiers. We investigate control flow based characterization of a program execution to build robust deep learning malware classifiers.**

HeNet consists of a low-level behavior model and a top-level ensemble model. The low-level model is a per-application behavior model, trained via transfer learning on a time-series of images generated from control flow trace of an execution. We use Intel® Processor Trace enabled processor for low overhead execution tracing and design a lightweight image conversion and segmentation of the control flow trace. The top-level ensemble model aggregates the behavior classification of all the trace segments and detects an attack. The use of hardware trace adds portability to our system and the use of deep learning eliminates the manual effort of feature engineering.

We evaluate HeNet against real-world exploitations of PDF readers. HeNet achieves 100% accuracy and 0% false positive on test set, and higher classification accuracy compared to classical machine learning algorithms.

I. Introduction

According to recent security threat reports, one million malware variants hit the Internet every day while new malware samples found in Q3 2017 increased to a record high of 57.6 million [1][2]. Traditional signature-based malware detection techniques are growing ineffective as the malware authors use code obfuscation, polymorphism, etc. to automate writing new malware variants and zero-day exploits to create new malware. Hence, the security community are pursuing machine learning-based detection approaches to adapt to the rapidly changing malware ecosystem. Neural networks able to automatically learn features have been proven to detect and classify complex malware with high accuracy [3][4]. While significant progress in deep learning makes it promising for automated malware analysis, further work is needed to build *robust* and *scalable* malware classifiers.

Existing deep learning-based malware detection and classification systems learn features from static analysis of an executable file or runtime behavioral analysis. The static analysis approaches build neural networks using header, instruction opcodes, or raw bytes of an executable file and classify the file before execution [5][6][3]. The dynamic analysis approaches model user or kernel API call sequences over time for a program execution [4][7][8][9][10]. Despite their potential, the neural network malware classifiers are susceptible to *adversarial inputs*. By crafting adversarial malware samples [11] or API call sequences [12][13], an attacker can mislead the malware classifier and achieve high misclassification rate without affecting the malware functionality. Generally, it is a lot more challenging to perturb dynamically gathered features. The API call-based dynamic features, however, have been shown to be vulnerable to black-box attacks. The proposed attacks are effective against many classifiers including recurrent neural network variants, feed forward deep neural networks, and traditional machine learning classifiers. To be noted, API call-based malware analysis requires significant manual feature engineering to select the API calls most relevant to malware execution.

We investigate more sophisticated dynamic features to build a robust deep learning malware classifier. In this paper, we describe a malware detection system powered by HeNet, a hierarchical ensemble neural network, that classifies the control flow trace of program execution. Control flow analysis has been widely used in security to mitigate memory corruption attacks by enforcing pre-specified security policies over a program execution [14]. HeNet, however, is the first malware detection approach to apply deep learning on fine-grained control flow traces. We build a per-application deep-learning behavioral model to distinguish benign and malicious executions of the application. The key observation underlying our work is that malware exploits temporarily or persistently execute malicious code in the context of a victim process and hence, change the original control flow of the application.

We utilize Intel® Processor Trace (Intel® PT), a low overhead tracing capability in the CPU, to get the complete control flow audit of a program execution. Intel® PT records the non-deterministic control flow transfers, contextual, timing, etc. information in highly compressed packets. The control flow packets combined with the program binary can be used to reconstruct the exact sequence of instructions executed but the post-processing takes a long time due to the high volume of trace data. We, however, notice that the control flow packets can be used as an encoded representation of the control flow transfers over a program execution. The time series of control flow packets can be classified using

DOI 10.1109/SPW.2018.00025

deep neural networks for malware detection [15][16]. In this work, we explore the image representation of a control flow trace and convert the payload of control flow packets to a stream of pixels. While we do not claim image as the optimal transformation of a trace, exploratory data analysis and visual inspection of the control flow images from benign and malicious samples provide the foundation for our proposal. Image representation also provides a compact way to extract full information from control flow. Utilizing low level trace packets eliminates manual feature engineering and makes our system portable across different OSes and hardware.

We train a deep neural network on the sequence of control flow images to build an application behavioral model. To accelerate the training on a large number of images, we use deep transfer learning [17][18]. Each trace is represented by a time series of images whose label is classified by the low-level model. An ensemble model is imposed on top to aggregate the behavior classifications over the entire trace.

The contributions of this paper are summarized as follows:

(1) We develop the first (to the best of our knowledge) deep learning approach to malware detection using fine-grained control flow traces of an application execution. We present the architecture of HeNet, a hierarchical ensemble neural network, that classifies a control flow trace to detect malware.

(2) We utilize Intel® PT, a low overhead tracing hardware in commodity processors, to collect control flow trace. The low level trace based features make HeNet extensible and portable to various systems.

(3) We represent the control flow trace as a time series of images and define behavioral anomaly detection as an image classification problem. We apply transfer learning to achieve high accuracy and faster training on large number of trace images.

(4) We evaluate HeNet against real-world exploitations of Adobe® Reader 9.3 on Windows® 7 32 bit. The experimental results show 100% accuracy to classify malicious and benign PDFs and 0% false positive. We show that HeNet produces much higher classification accuracy and lower false positive rate compared to classical machine learning algorithms.

The rest of the paper is organized as follows: Sec. II discusses the threat model and a brief background on Intel® PT. Sec. III presents the design overview of our anti-malware system. Sec. V presents HeNet architecture in detail. HeNet performance results are reported in Sec. VI. We discuss the related work in Sec. VII and conclude in Sec. VIII.

II. Background and Threat Model

A. Threat Model

We intend to detect malware attacks by classifying control flow transfers over an application execution. Hence, we consider the attacks that change the execution flow of a benign application. This threat model covers a wide range of malware attacks, including control flow attacks such as Return-Oriented-Programming (ROP), Jump-Oriented-Programming (JOP) attacks, fileless malware that run under the cover of benign applications, zero-day attacks.

HeNet learns the control flow model of an application execution from the sequence of control flow packets produced by Intel® PT. Hence, we assume that the Intel® PT hardware is trusted so that the generated trace is complete and accurate. We also assume that the integrity of trace is protected in memory and while in use for decoding, training, and classification.

B. Intel® Processor Trace (Intel® PT)

Intel® PT is a low overhead debugging hardware that captures the **complete** execution trace of monitored applications. It captures information about application execution on each hardware thread using dedicated hardware so that the trace can be post-processed to reconstruct exact program flow. This capability is particularly useful for baremetal malware analysis, targeted to track evasive malware that can otherwise bypass software instrumentation based malware detection.

Intel® PT captures control flow, timing, and other contextual information in highly compressed trace packets. We utilize the Target IP (TIP) and Taken Not-Taken (TNT) packets to characterize an application execution. TIP packets record the target address of indirect branches, exceptions, interrupts. TNT packets track the direction of conditional branches within basic blocks. A TNT packet contains a number of conditional branches, where each bit corresponds to a conditional branch. If a conditional branch is taken during execution, a bit of value 1 is recorded; 0 otherwise.

III. Malware Detection System Overview

Figure 1 shows the system overview of malware detection using HeNet on Intel® PT trace. We train HeNet, consisting of a low-level behavior and top-level ensemble model, to build a benign/malicious classification model of an application execution. During testing, we apply the HeNet behavior model to detect anomalous trace/execution segments and its ensemble model to decide on an attack over the full trace.

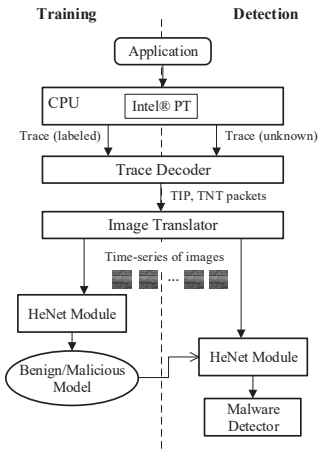


Figure 1. System overview of malware detection using Intel® PT and HeNet

We configure Intel® PT to monitor an application at runtime and generate execution trace. We use both benign and malicious traces of an application to train HeNet. We refer to

a trace as *benign* when the benign version of an application is executed and as *malicious* when the application is exploited to conduct an attack. Given a raw PT trace, the *Trace Decoder* decodes the trace packets and extracts the sequence of TIP and TNT packets. A data pre-processing component, *Image Translator*, converts the TNT and TIP packets to a 1-D array of pixels. Due to the high volume of trace data, it is computationally infeasible to build a neural network on the whole pixel array. Instead, we segment the pixel array into n sub-arrays and establish a low-level behavior model as a part of the hierarchical structure of HeNet. The *Image Translator* converts the execution trace to a time series of images. The *HeNet Module* performs transfer learning on the sequence of trace images to learn a benign/malicious behavior model.

At testing, we perform segmentation and image conversion of a given trace, as similar to training. The *HeNet Module* applies the behavior model on each image to classify the execution behavior in that segment, and uses top-level ensemble to aggregate the behaviors learned from all the segments. The aggregated probability is fed into a *Malware Detector* which detects an attack if the probability exceeds a threshold.

IV. Control Flow to Characterize Execution

We utilize the TIP and TNT packets in Intel® PT trace to characterize an execution. We design a lightweight image conversion technique that converts the payload of a TIP/TNT packet to a stream of pixels. A pixel value is of [0, 255].

A. Taken Not-Taken (TNT) packet conversion

There are two variants of TNT packets: (1) Short TNT packet of 1 byte, (2) Long TNT packet of up to 8 bytes. The short TNT packet has the least significant bit as 1-bit header and may contain from 1 to 6 TNT bits. The long TNT packet has the least significant 2 bytes as 2-byte header and may contain up to 47 TNT bits. The last valid TNT bit is followed by a trailing 1, or *stop-bit*. If the TNT packet is not full, the *stop-bit* moves up and the trailing bits are filled with 0's.

We convert the short TNT packet to an unsigned 1-byte value representing 1 pixel intensity. For long TNT packets, we remove the headers and find the length of the packet payload. We convert each byte of payload to a pixel value. Hence, a long TNT packet may generate from 1 to 6 pixels.

B. Target IP (TIP) packet conversion

The TIP packet has 1 byte header, where the most significant 3 bits specify how the target address of an indirct branch is compressed in payload. Due to compression, the size of target IP payload may vary from 0 to 8 bytes.

We remove the TIP packet header and extract the target IP payload. Using the IP payload and last saved IP, we reconstruct the full target address of the indirect branch. We use auxiliary runtime binary load information to find the binary which the target address belongs to. We identify each binary with a 1 byte ID. To normalize the impact of ASLR, we calculate the offset of the target address from the binary base address. We represent the target address as a <binary ID, offset> pair and convert the 1 byte ID and 4 byte offset to an array of 5 pixels.

V. HeNet: Hierarchical Ensemble Neural Network

In this section, we present the detailed architecture of HeNet, designed for Intel® PT trace. HeNet consists of a low-level deep learning model to classify benign/malicious behavior of trace segments, and a high-level ensemble model to aggregate behaviors learned from all the trace segments.

A. Segmentation and Image Conversion

After the pixel conversion of TNT and TIP packets as described in Section IV, an execution trace is represented as a one-dimensional pixel array. We denote the pixel array by $X \in [0, 255]^L$, i.e., X has length L and each element is within 0-255. Next, we divide X into n segments, where each trace segment is of length m^2. If L is not a multiple of m^2, we pad the last segment with 0's. The choice of m depends on the network architecture.

Each segment is mapped to a two-dimensional $m \times m$ array and represented as a gray-scale image. Thus, X, the pixel representation of a trace, is transformed to a time series of images $(I_1, I_2, \cdots, I_n)$, where each $I_k \in [0, 255]^{m \times m}$. Note that the sequence-to-image transformation only resizes the data shape and no information is lost from resizing.

B. Low-Level Behavior Model

The low-level model in HeNet is a Deep Neural Network (DNN) to classify the behavior of a trace segment as benign or malicious. As we use the image representeation of a trace segment, we apply deep learning to extract the textural features of the trace images. Visual similarities among the benign trace segment images and among the malicious trace segment images motivate us to frame the problem as an image classification problem.

1) Training Dataset: The time series of trace images are the training dataset for the low-level DNN model. We utilize the original label of the Intel® PT trace to label the sequence of images. Noisy ground truth, however, imposes a challenge here. In supervised learning, the labels of training data are assumed to be absolutely correct. Our ground truth is noisy, since a malicious trace may contain benign execution until the attack happens. Hence, if a trace X^i is malicious, we first identify the time of attack k and then label $(I^i_k, I^i_{k+1}, \cdots, I^i_n)$ as malicious. For a benign trace X^i, all the segmented images $(I^i_1, I^i_2, \cdots, I^i_n)$ are labeled as benign.

2) Training via Transfer Learning: Intel® PT generates a large volume of data and segmentation makes the low-level sample size even larger. Hence, training a neural network from scratch may not be the optimal solution. We perform transfer learning [17] to train the behavior model. We transfer the knowledge learned from the state-of-the-art DNNs on large image sets and retrain the layers suitable for our classification.

Currently, HeNet framework supports transfer learning using a variety of networks including Inception [19][20][21] and VGG [22] networks, pre-trained on ImageNet datasets. The resizing parameter m is set to an appropriate value. The channels of gray-scale images are augmented by replicating

the gray channel into RGB channels. Our empirical analysis shows that the choice of a particular transfer learning network does not make significant difference in classification performance. HeNet, however, shows higher accuracy and lower false positive rates compared to classical machine learning algorithms and neural networks trained from scratch. Transfer learning also helps HeNet achieve faster model convergence (within 10 epochs in our experiments).

C. Top-level Ensemble Model

Given an execution trace, represented as image sequence $(I_1, I_2, ..., I_n)$, the low-level model produces corresponding behavioral probabilities $(p_1, p_2, ..., p_n)$. The ensemble model aggregates the probabilities by computing the average, $\overline{p} = \frac{1}{n}\sum_{i=1}^{n} p_i$. We compare $\overline{p}$ with a pre-specified threshold to decide whether the execution corresponds to an attack or not.

HeNet is summarized in Algorithm 1.

Algorithm 1 HeNet on Intel® PT

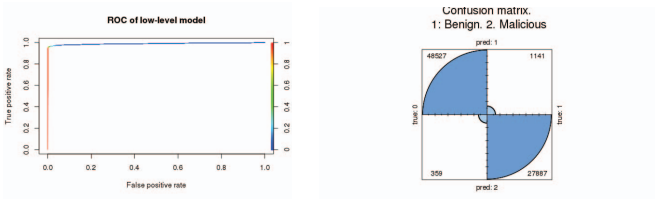
1: **Goal:** Malware detection.
2: **Input:** Intel® PT trace, resizing parameter m.
3: **Step 1** Convert TNT and TIP packets to pixels of value [0, 255]. Denote pixel array by X.
4: **Step 2** Slice X into n segments, where $n = \frac{L}{m^2}$. L is the length of X and a segment is of length m^2.
5: **Step 3** Convert each segment to an image of $m \times m$. Denote an image by I_k with $k \in \{1, 2, \cdots, n\}$.
6: **Step 4** Apply low-level deep learning behavior model on I_k and obtain probability of each segment as p_k.
7: **Step 5** Predict the label of X by averaging $(p_1, ..., p_n)$.

VI. Performance Evaluation

In this section, we present the performance results of HeNet in detecting real-world attacks. We evaluate HeNet with respect to malware detection accuracy and false positive rate. We compare the performance of HeNet low-level model with a neural network trained from scratch and three classical machine learning algorithms.

A. Experimental Setup

To show the effectiveness of our malware detection system, we use ROP attacks against Adobe® Reader 9.3 in Windows® 7 32 bit. In our experiments, we use a set of 348 benign and 299 malicious PDF samples. We, however, acknowledge that capturing the complete control flow graph of an application is a challenging task and plan to train HeNet with larger dataset to increase model coverage. We utilize Intel® VTune™ Amplifier driver to collect Intel® PT trace and develop a decoder to extract TNT and TIP packets. Out of total 647 traces, 53 are set aside for out-of-sample testing. From the remaining traces, we choose train-validation-test ratio as $0.8 : 0.1 : 0.1$. We set the resizing parameter m to 224 and the gray-scale images are of size 224×224. Thus, the training and validation sets contain 419075 benign and 276110 malicious trace images and the test set have 48886 benign and 29028 malicious images.



(a) ROC of behavior model. The area under curve is 0.989

(b) Plot of confusion matrix. Accuracy is 0.981 and false positive rate is 0.0073.

Figure 2. Performance of low-level behvaior model

B. Experimental Results

1) Low-level behavior model performance: To train the low-level model, we utilize Inception-BN at iteration 126 as the pre-trained model for transfer learning with learning rate at 0.05 and batch size at 100. We retrain the last fully connected layer and the softmax layer on trace segment images. Figure 2a shows the ROC curve of the low-level behavior model. The area under the curve is 0.989. Figure 2b shows that the behavior model achieves 0.981 accuracy and 0.0073 false positive rate on the test set. Thus, HeNet achieves low false positive rate without sacrificing accuracy. Utilizing transfer learning, the training converges within 10 epochs.

Next, we examine the features extracted from the low-level behavior model using transfer learning. We randomly select 22% of benign and 55% of malicious behavior images from test set, feed them to the trained low-level model, and compute the global average pooling. Then we apply t-distributed stochastic neighbor embedding (t-SNE) [23] on the global average pooling and obtain the first three dimensions. Figure 3 shows the correlation plot (top), t-SNE scatter plots (lower-triangular), t-SNE density plots (diagonal), correlation (upper triangular), and label box-plots (upper triangular). As we see, the first three t-SNE dimensions are well separated. The correlation among the malicious trace images and correlation among the benign trace images are positive. The correlation among the malicious and benign trace images is negative.

We present the performance of the low-level model when trained via Inception transfer learning vs. trained from scratch. We note that training a neural network from scratch takes significantly long time. Hence, we resize the trace images to low-resolution images of size 28×28 with gray-scale channel and select LeNet [24] to train from scratch. Since LeNet architecture is much shallower, training per epoch does not take too long, yet the model converges much slower.

We also compare the performance of HeNet with classical machine learning algorithms including random forest, naive Bayes, and k-nearest neighbor on high and low resolution vectorized images. For this comparison, we sub-sample 17% of the training data for 5 times and compute the average accuracy and false positive. For 224×224 images, we apply principal component analysis to reduce dimension from 50176 to 1000.

Table I presents the comparative analysis of HeNet low-level model, LeNet low-resolution model, nearest neighbor, and random forest on both high and low resolution images.

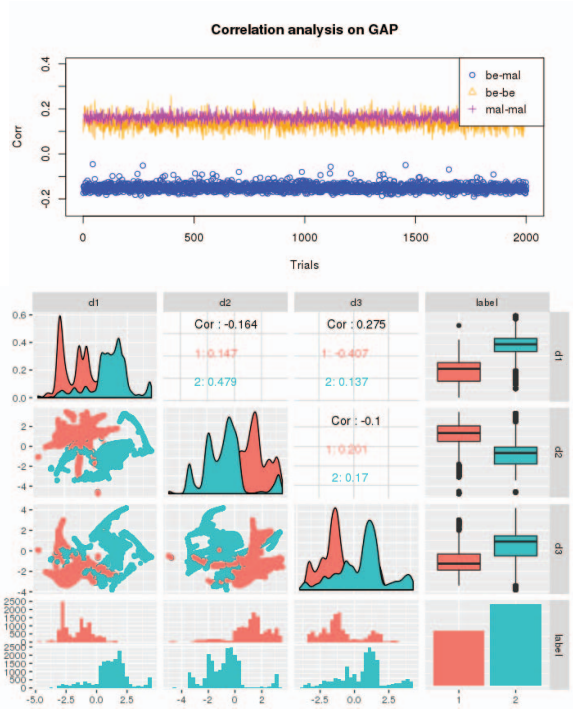


Figure 3. Visualization of the features learned from HeNet low-level model.

Table I
COMPARISON OF HENET WITH OTHER CLASSIFIERS

Behavior model	Accuracy	FPR	Resolution
HeNet low-level model	0.98	0.0073	224×224×3
Low-resolution model	0.90	0.1	28×28×1
3-nearest neighbor + PCA	0.80	0.05	$224^2 \rightarrow 1000$
Random forest + PCA	0.63	0.02	$224^2 \rightarrow 1000$
3-nearest neighbor	0.75	0.4	28^2
Naive Bayes	0.82	0.23	28^2

Among all the algorithms, the HeNet low-level model achieves the highest behavior classification accuracy and lowest false positive rate. The low-resolution model is trained on 28×28 grey-scale images, resized from the 224×224 images. The performance degradation of the low-resolution model may be due to much shallower architecture compared to Inception-v1, fewer image channels, or information loss from image resizing. The low-resolution model, however, may provide a performance baseline for the deep learning image classifiers. We further investigate the low-resolution model with deeper architecture on three-channel images in Sec VIII. We expect the machine learning algorithms will have performance improvement if more domain expertise is leveraged for feature extraction. On the other hand, it highlights the advantage of deep learning in greatly saving the cost of feature engineering.

2) Top-level ensemble model performance: The ensemble model aggregates the low level execution behaviors by computing an average of the behavior probabilities. To determine whether an execution trace is benign or malicious, we set a threshold of 0.5. On our test set, we obtain 100% detection accuracy and 0 false positive rate.

The ensemble probabilities of being benign for the 31 benign test traces have a mean of 0.98 and a minimum of 0.96. The p-value of a two-sided Wilcoxon rank sum test on the ensemble probabilities of being benign for the 31 benign traces and 22 malicious traces is 4.3×10^{-15}, indicating statistically distinguishable prediction scores. Figure 4 shows the density of ensemble benign probability across benign and malicious test cases. Thus, HeNet achieves high confidence classification at both low-level behavior model and top-level ensemble model.

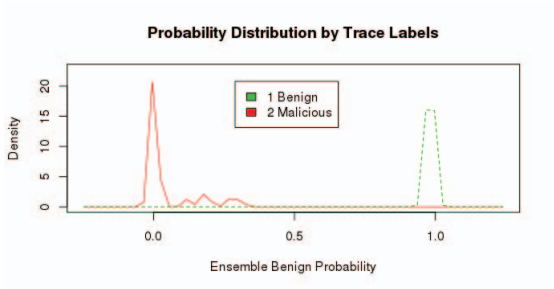


Figure 4. Density of the ensemble benign probabilities. HeNet produces high confidence predictions.

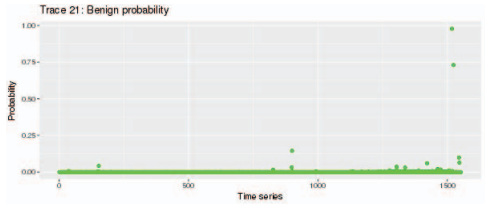
C. Time Series of Behavior Probabilities

The aggregated probability produced by the ensemble model enables malware detection with high confidence. For the traces with high ensemble probability, the time series of behavioral probabilities are relatively smooth. Refer to Figure 5a as an example. We, however, examine a malicious test trace which has a 0.62 probability of being malicious. Here, the decision confidence is lower compared to other test cases. Figure 5b shows the execution time-series and reveals that the behaviors towards 20% from the tail were classified as benign. From manual analysis, we find that the attack invokes a lot of $strcmp()$ function calls in a loop. Since $strcmp()$ is a commonly used function, the malicious execution segments are misclassified as benign causing a lower detection confidence.

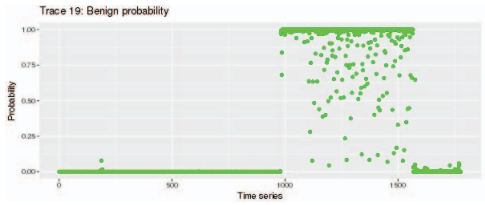
VII. RELATED WORK

There has been little work so far in developing robust deep learning malware classifiers. Existing deep learning approaches use static or dynamic features to construct neural networks for malware detection or classification. The static features are obtained by analyzing an executable file. Neural network malware classifiers have been trained on static features including byte/entropy histogram, imports, raw header bytes and other metadata of Portable Executable files or several million byte long executable files [7][3][5]. Despite this success, Grosse et al. showed that such malware detectors can be bypassed by adversarially crafted malware samples [11].

Dynamic analysis approaches are more robust to obfuscation. Dahl et al. proposed a multi-class malware neural network classifier trained on random projections of features including API call tri-grams, API parameter, and file strings [4]. Pascanu et al. trained recurrent neural nets on API call information for both malware detection and classification [25]. Kolosnjaji et al. combine convolutional and recurrent neural



(a) An example of smooth probabilities. The trace is malicious and the probabilities of being benign are mostly 0.



(b) A malicious trace with anomalous behavior. We discover that towards the tail of execution, a system function is called in a loop, causing the low-level model to misclassify the time period as benign.

Figure 5. Time series behavior probabilities.

networks to optimize malware classification [10]. API call based approaches require significant domain knowledge and feature engineering. More importantly, an attacker can effectively bypass sequential API feature based malware detectors by generating sequential adversarial examples [12][13].

Classical machine learning algorithms have been applied on static binaries, converted to images, for multi-family malware classification [26][27].

VIII. Conclusion and Future Work

HeNet is the first deep learning approach to classify Intel® PT generated fine-grained control flow traces for malware detection. HeNet is portable across various software systems due to using hardware execution trace based features. HeNet achieves high accuracy and faster training as it builds application behavior model by using transfer learning on a time-series of images, generated from control flow trace. The experimental results show that HeNet detects ROP attacks agianst Adobe® Reader with 100% accuracy and 0% false positive.

This work is a step towards deep learning designed on application execution traces for robust malware detection. To further improve HeNet, it is desired to conduct cross validation on the top-level threshold, granularity analysis of top-level model, exploration of different ensemble schemes, and experiments with diverse applications. Below, we discuss a few important future directions.

A. Ensemble Behavior Model

To achieve high accuracy and faster training, the HeNet low-level behavior model is trained via transfer learning on high resolution trace images. We refer to the model as *high-resolution model*. We investigate an ensemble of *high-resolution model* and *low-resolution model*, a model trained from scratch on low resolution images, for further performance improvement. We train ResNet from scratch with 20 layers

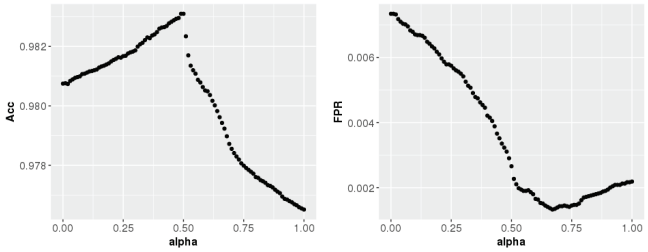


Figure 6. Accuracy and FPR for various convex combination parameter α. Optimal accuracy and FPR of HeNet and low-res model is at $\alpha = 0$ and $\alpha = 1$, respectively.

Table II
Performance of Ensemble Behavior Model

Behavior model	Accuracy	FPR	Resolution
Ensemble low-level model M_e	0.983	0.0029	High-low res
HeNet low-level model M_h	0.981	0.0073	224×224×3
ResNet Low-resolution model M_l	0.976	0.0022	28×28×3

for three-channel images of size 28×28, learning rate of 0.05, and batch size of 100. We choose the model at 20th epoch. Note that HeNet's behavior model converges within 10 epochs. We apply convex combination on the *low-resolution* and *high-resolution model* so that the new model is $M_e = \alpha M_l + (1-\alpha)M_h$, where $\alpha \in [0,1]$ is selected to optimize metrics including accuracy, false positive rate, sensitivity, and area under the curve [28]. In Figure 6, we consider an optimal convex combination parameter $\alpha \in [0.48, 0.68]$ for both desired accuracy and FPR. Selecting $\alpha = 0.5$, model M_e has 0.9831 accuracy, higher than that of M_l and 0.0029 FPR, lower than that of M_h. We plan to further investigate this model.

B. Adversarial Machine Learning

Studies show that small adversarial purturbations, even unnoticable by human eyes, may lead to misclassification by deep learning image classifiers [29][30][31]. As we assume end-to-end trace security, to fool the HeNet behavior classification, the attacker goal is to inject malicious control flow transfers at runtime in a way that also keeps the application operational. Determinstic control flow integrity methods are shown to be vulnerable due to the weakness of control flow graph and security policies [32][33]. HeNet learns the legitimate control flow behavior of an application from training on dynamic traces. As we use both conditional and indirect branches, the behavior model is fine-grained. Hence, it becomes a difficult challenge for the attacker to effectively manipulate the control flow. We, however, will investigate whether an attacker can bypass the HeNet ensemble model exploiting threshold based decision. We will also study adversarial perturbations to exploit HeNet under certain constraints.

C. Different Data Representations

Beyond pixel conversion, TNT/TIP packets may be utilized to extract other useful features. We plan to investigate indirect branch target addresses to build adversary resilient deep learning malware classifier.

ACKNOWLEDGMENT

We would like to thank David Durham at Intel Labs for his insightful discussion and valuable feedback.

REFERENCES

[1] R. Behrends, L. K. Dillon, S. D. Fleming, and R. E. K. Stirewalt, "Internet security threat report," Symantec, Tech. Rep., April 2017.

[2] N. Minihane, F. Moreno, E. Peterson, R. Samani, C. Schmugar, D. Sommer, and B. Sun, "Mcafee labs threat report," McAfee Labs, Tech. Rep., December 2017.

[3] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," *ArXiv e-prints*, oct 2017.

[4] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *IEEE Intl. Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 3422–3426.

[5] E. Raff, J. Sylvester, and C. Nicholas, "Learning the PE Header, Malware Detection with Minimal Domain Knowledge," *ArXiv e-prints*, Sep. 2017.

[6] A. Davis and M. Wolff, "Deep learning on disassembly data," in *BlackHat USA*, 2015.

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

[8] G. Kim, H. Yi, J. Lee, Y. Paek, and S. Yoon, "Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems," *CoRR*, vol. abs/1611.01726, 2016.

[9] W. Huang and J. W. Stokes, *MtNet: A Multi-Task Neural Network for Dynamic Malware Classification*. Springer International Publishing, 2016, pp. 399–418.

[10] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, *Deep Learning for Classification of Malware System Call Sequences*, 2016, pp. 137–149.

[11] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *CoRR*, vol. abs/1606.04435, 2016.

[12] W. Hu and Y. Tan, "Black-box attacks against RNN based malware detection algorithms," *CoRR*, vol. abs/1705.08131, 2017.

[13] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against rnns and other api calls based malware classifiers," *ArXiv e-prints*, July 2017.

[14] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 4:1–4:40, 2009.

[15] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018.

[16] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 1578–1585.

[17] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[18] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[21] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, 2017, pp. 4278–4284.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *IEEE Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 1916–1920.

[26] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*. ACM, 2011, p. 4.

[27] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. ACM, 2011, pp. 21–30.

[28] L. Chen, D. C. Comer, C. E. Priebe, D. Sussman, and J. C. Tilton, "Refinement of a method for identifying probable archaeological sites from remotely sensed data," in *Mapping Archaeological Landscapes from Space*. Springer, 2013, pp. 251–258.

[29] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[30] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," *arXiv preprint arXiv:1610.08401*, 2016.

[31] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," *arXiv preprint arXiv:1608.04644*, 2016.

[32] N. Carlini, A. Barresi, M. Payer, D. Wagner, and T. R. Gross, "Control-flow bending: On the effectiveness of control-flow integrity," in *USENIX Security Symposium*, 2015, pp. 161–176.

[33] N. Carlini and D. Wagner, "ROP is still dangerous: Breaking modern defenses," in *USENIX*.