Device Behavioral Profiling for Autonomous Protection using Deep Neural Networks

Sandeep Gupta
University of Trento, Trento, Italy
sandeep.gupta@unitn.it

Bruno Crispo
University of Trento, Trento, Italy
bruno.crispo@unitn.it

2023 IEEE Symposium on Computers and Communications (ISCC) | 979-8-3503-0048-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCC58397.2023.10218275

***Abstract*—Demand for autonomous protection in computing devices can not go unnoticed with an enormous increase in cyber attacks. Consequently, cybersecurity measures to continuously monitor and analyze device critical activity, identify suspicious behavior, and proactively mitigate security risks are highly desirable. In this article, a concept of behavioral profiling is described to distinguish between benign and malicious software by observing a system's internal resource usage on Windows devices. We rely on the Windows built-in event tracing mechanism to log processes' critical interactions for a given amount of time that are converted into structured data using a graph data structure. After that, we extract features from the generated graphs to analyze a process behavior using a deep neural network. Finally, we evaluate our prototype on a collected dataset that contains one thousand benign and malicious samples each and achieve an accuracy of $\approx 90\%$.**

***Index Terms*—Autonomous protection, intrusion detection, behavioral profiling**

I. Introduction

The new malware and malware variants pose a constant threat to the security and safety of users computing devices and data. Studies have shown that traditional malware detection tools (e.g., signature-based detection, check-summing, and application whitelisting) are not highly effective to protect our digital assets [1], [2]. CryptoLocker, ILOVEYOU, Koobface, Code Red, Melissa, Wannacry, Petya, Mirai, SpyEye, and Stuxnet are just a few malware that notoriously crippled millions of devices and caused colossal financial losses in recent years [3].

By and large, both the static and dynamic analysis approaches have been heavily investigated for malware detection [4], [5]. Static analysis involves examining the code or structure of a file without executing it. It can be useful for identifying known malware patterns and identifying code similarities with known malware families. However, it can be less effective against malware that employs code obfuscation techniques, as these techniques can make it more difficult for static analysis tools to recognize the malware's malicious intent. Additionally, static analysis can result in false positives, as benign code may be incorrectly flagged as malicious.

Dynamic analysis involves executing a file in a controlled environment and analyzing its behavior. It can be effective at detecting new and unknown malware, as it can identify behaviors that are characteristic of malware. However, dynamic analysis can be more resource-intensive and time-consuming than static analysis, as it requires the creation of a virtual environment to execute the file safely. Dynamic analysis can also be vulnerable to sandbox detection, as malware may be programmed to detect when it is running in a sandbox environment and change its behavior accordingly. Furthermore, fileless attacks can be difficult to detect using both static and dynamic analysis methods, as they do not involve writing files to disk. This is a technique used by some malware to evade detection by traditional anti-malware systems.

Nevertheless, considering the significant rise in remote work trends and interconnectivity of computers in recent years, near real-time device behavioral monitoring, anomaly detection, and threat prevention mechanisms are highly desirable. Autonomous protection can be a promising solution to detect and respond to potential cyber-attacks that can operate independently of human intervention or control. Inherently, autonomous protection employs behavioral analysis and artificial intelligence to continuously monitor and analyze device critical activity, identify suspicious behavior, and proactively mitigate security risks.

In this article, we introduce a concept of behavioral profiling by using system logs and traces in Windows OS-based systems for autonomous protection from malware. Event logs that are the inventory of essential activities on a system can be a primary source of digital evidence for forensic investigation [6]. We rely on the Windows built-in instrumentation-free Event Tracing for Windows (ETW) for log collection. We consider events associated with an operating system, applications running on devices, and network traffic with connected devices as part of the trust computing base [7].

A. Contributions

- We design and develop a Behavioral Profiling Prototype (BPP) for autonomous protection from malware. The design of each module, i.e., *Logs Collection Module*, *Graphs Generation Module*, *Features Extraction Module*, and *Behavioral Analysis Module* in BPP is delineated in-depth.
- We demonstrate the transformation of unstructured data to structured data using a graph data structure.
- We elucidate mechanisms for feature generation from Windows events using a built-in ETW logging facility for Windows-based systems.
- We present a step-wise implementation of Deep Neural Network (DNN) for behavioral analysis.

- We collect a total of two thousand benign and malicious samples for evaluating the prototype.

B. Article Structure

The rest of the article is structured as follows: *Section II* presents the work related to behavioral-based malware detection approaches. *Section III* describes the behavioral profiling prototype. *Section IV* presents a detailed high-level design of a behavioral profiling prototype. *Section V* provides the experimental details including data generation, results, and discussions. *Section VI* concludes the article together with some insight about future work.

II. Related Work

Behavioral-based approaches leverage various dynamic properties like file system activities, terminal commands, network communication, and system calls to understand the run-time behavior of malware during the execution [8]. Berlin et al. [9] investigated the potential of Windows audit logs to augment existing Windows enterprise management tools. The authors presented the argument that audit logs can provide an effective, low-cost alternative to deploying additional expensive agent-based breach detection systems in many government and industrial settings. The system used a linear classification model and tested the model using a small subset of audit log features, i.e., q continuous time-ordered sets of events that detected 83% percent of malware samples with a 0.1% false positive rate. Aharoni et al. [10] proposed a mechanism that extracts behavioral and statistical information from the events to build a predictive model based on supervised learning. The mechanism ranks the events for predicting malicious processes, which was tested using VirusTotal and was able to detect more than two-thirds of the malicious events with less than a 2% false positive rate.

Rana et al. [11] proposed a framework for ETW to capture kernel-level events and transform the logs in human-readable JSON format. The authors also integrated the agent into the cuckoo sandbox. The data captured by the Cuckoo included signatures, networks, processes, files system, registry, API calls, and some static information. And, the data captured by their agent included process, files system, registry, page faults, ALPC, DLLs imported, and some static information. Overall, 104 features are extracted from the data collected by Cuckoo, and 58 features are extracted from the data collected by their agent for behavior analysis. K-Nearest Neighbor, Multi-layer Perceptron, Decision Tree, and Random Forest are used to design a bimodal classification model. Random Forest classifier performs the best on the combined (Cuckoo+ETW) data with an accuracy of 99.68% and a false positive rate of 0.45%.

Ahmed et al. [12] proposed a ransomware detection system that they called profiling kernel-Level events to detect ransomware (PEELER). The authors explain that the ransomware typically disables real-time monitoring, archive scanning, or even deletes its binary from the disk to achieve stealthiness and remain undetected. Subsequently, they launch the attack by encrypting files or locking the device screens and displaying ransom payment notes. The PEELER system tracks ransomware activities from a ransomware binary execution to a ransom note display on the victim's device, thus, based on the device's behavioral characteristics Peeler continuously monitors a target system's kernel events and detects ransomware attacks on the system.

III. Behavioral Profiling Prototype

The basic concept that we exploit to design our Behavioral Profiling Prototype is inspired by a criminal profiling ideology, i.e., "*a creation of a psychological, behavioral, and demographics profile of the type of person likely to have committed the crime*" [13].

A. Definition

Behavioral profiling can be described as an act or process of gathering information based on observed characteristics or known patterns gathered from/about a *system* or *device* yielding vital information that could aid to detect potential threats.

B. Pilot Study

Our pilot study targets Windows devices given the widespread use of Microsoft Windows operating systems which is about 30% of the global OS market share [14]. Studies have reported that over 95% of all new malware threats surfaced in 2022 targeted Windows devices [15]. Some of the malware that Microsoft listed include Coin miners, Exploits and exploit kits, Macro malware, Phishing, Ransomware, Rootkits, Supply chain attacks, Tech support scams, Trojans, Unwanted software, and Worms [16].

We create a Windows 10 virtual machine (VM) using VirtualBox[1] that is freely available as Open Source Software under the terms of the GNU General Public License (GPL). We use a virtual machine (VM) to emulate the behavior of an executable code, JavaScript, website content, and other potential ways that can be used to attack a system and collect events log to record Windows events using a built-in ETW logging facility for any Windows-based system.

The log collection process for generating benign and malicious samples is explained in Section IV-A. ETW Logs can be described as semi-structured texts that comprise a constant part, i.e., log event, and a variable part, i.e., log parameter. We preprocess the raw ETW logs by correlating their process IDs and timestamps. Each sample consists of several log files containing information about events (*listed in Table I*) for three minutes triggered by various activities running on the VM. Each set of log files is converted into directed acyclic graphs that are described in Section IV-B for further analysis. Section IV-C describes the features that we extracted from each sample for behavioral analysis. Section IV-D explains the methods investigated for the samples behavior analysis using DNN.

[1]https://www.virtualbox.org

IV. High Level Design

We design and develop BPP targeting the devices running on the windows operating system. Figure 1 illustrates the block diagram of BPP consisting of four core modules, i.e., *Logs Collector*, *Graphs Generator*, *Features Extractor*, and *Behavior Analyzer*.

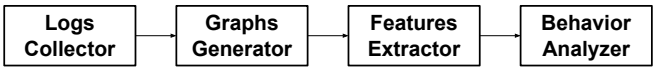

Fig. 1. The block diagram of Behavioral Profiling Prototype

A. Log Collection Module

Event Tracing for Windows provides a framework for tracing and logging events that are raised by user-mode applications and kernel-mode drivers [17]. We use APIs *TraceLoggingRegister* and *TraceLoggingWrite* to register and write events triggered for a given period, which is easily configured. The logging mechanism utilizes per-processor buffers that are written to a file by an asynchronous writer thread. ETW events are rich metadata, localizable message strings, and schematized data payloads for easy consumption of event data. High-level events and kernel events logged information related to process and thread creation events, file events, network events, and registry.

Table I lists the events that we registered to monitor a device's behavior [11]. File input/output event class can be exploited for getting events related to the file system. Process event class can be used to capture events related to processes. The thread event class can be used to capture events related to threads. Registry event class can be used to capture events related to the registry. And, network event class can be used to capture network-related events.

TABLE I
Events for monitoring a device's behavior

Class	Event/Identifier	Description
File Input/Output	File Create ($FILE\_CREATE$)	A file is created
File Input/Output	File Delete ($FILE\_DELETE$)	A file is deleted.
Process	Process Start ($PROCESS\_START$)	A process is started.
Thread	Thread Start ($THREAT\_START$)	A thread is started.
Registry	Registry Set Value ($RIGISTRY\_SET$)	A key is set.
Network	TCP/IP Connect IPV4/IPV6 ($TCP\_CONNECT$)	A TCP connection is establish.
Network	UDP/IP Connect IPV4/IPV6 ($UDP\_CONNECT$)	A UDP connection is establish.

B. Graphs Generation Module

All the event logs collected for a pre-configured time interval are converted to directed acyclic graphs (DAGs) using networkx library[2] (refer Figure 2). We generate a DAG for each process, where a root node is represented by the process UID (unique identifier) and the children are represented by $UID.EVENT\_ID$, specifying the events (*listed in Table I*) triggered by the root process or the spawned processes.

[2]https://networkx.org

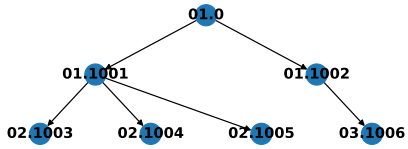

Fig. 2. A typical DAG of a process

C. Features Extraction Module

Let $G = (V, E, D)$ denote a DAG, where $V = \{v_i | i \in \mathbb{N}\}$ represents a set of $\mathbb{N}$ nodes and $E = \{e_{v_i v_j} | v_i, v_j \in V\}$ refers to a set of directed edges connecting node v_i to node v_j, and $D = d_{v_i} | v_i \in V$ denotes a set of dictionaries embed in each node. The graph-level features are described below.

- *Nodes Count*: The total number of nodes in a graph.
- *Leaf Nodes Count*: The total number of nodes with a child.
- *Graph Depth*: The total number of edges from the root node to a leaf node in the longest path.
- *Node Degree Statistics*: Univariate statistical information, such as minimum, maximum, mean, and standard deviation is computed from all the nodes' degrees in the given graph. Node degree is the total number of edges ingress and egress a node.
- *Events Count*: The total count of each type of event described in Table I.

D. Behavior Analysis Module

We implemented a densely connected feed-forward Neural Network exploiting the TensorFlow library. The parameters for the DNN architecture are empirically fine-tuned to obtain optimal results. An $input\_shape = (12,)$ is passed as an argument to the first layer that instructs Keras to insert an input layer before it. Here, 12 is the total number of graph-level features per observation described in Section IV-C. We rely on ReLU as an activation function for the first and second layers that can tackle the vanishing gradient problem and Sigmoid as an activation function for the output layer that is effective for classification problems.

However, we use different output units for the first and second layers for optimizing the model. We compile the model using $binary\_crossentropy$ as a loss function, $adam$ as an optimizer, and $accuracy$ as performance metrics. Binary cross entropy compares each of the predicted probabilities to the actual class output, i.e., 0 (malicious) or 1 (benign). The loss functions compute the quantity that a model seeks to minimize during training. Adam optimization is a stochastic gradient descent method that applies adaptive estimation of first-order and second-order moments to update network weights. Finally, accuracy makes the $evaluate$ function callable with the signature $result = fn(y\_true, y\_pred)$.

V. Experimental Details

A. Dataset

A total of two thousand graphs are generated containing an equal number of benign and malicious samples from the event logs acquired from emulating benign and malicious activities

on Windows 10 VM. Live malware samples can be downloaded from websites listed in Table II as per the website guidelines.

TABLE II
WEBSITES TO GET MALWARE SAMPLES

1.	http://www.vxvault.net/ViriList.php	5.	https://urlhaus.abuse.ch
2.	https://support.clean-mx.com/clean-mx/index.php	6.	https://tria.ge
3.	https://virussign.com	7.	https://malshare.com
4.	https://vx-underground.org/malware.html	8.	https://bazaar.abuse.ch

B. Results

We divide the dataset with graph-level features only into three disjoint and non-overlapping subsets, i.e., a training set, a validation set, and a test set. The $test\_size$ parameter can be between 0.0 and 1.0 defining the split ratio for the dataset. We use 50% of the dataset for the training model and 25% each for validation and testing of the model. The result is reported in terms of Accuracy, that is the ratio of truly accepted testing samples and the total number of testing samples, i.e. $\frac{TrueBenign+TrueMalicious}{TrueBenign+TrueMalicious+FalseBenign+FalseMalicious}$.

We use the Keras fit API to train the model by slicing the training samples into batches of size $batch\_size$ and repeatedly iterating over the entire dataset for a given number of $epochs$. The initialize the model with default batch size, i.e., 32. The selection of epochs depends on the inherent perplexity of data, however, it is recommended to use three times the number of features available in the dataset as an initial value for epochs that can be fine-tuned by increasing or decreasing the epochs based on the validation results. Table III presents the results obtained using $evaluate$ API using the testing dataset in ten separate runs using different hyper-parameters values for tuning the model.

TABLE III
MODEL RESULTS

	Hyper-parameters Tuning			Results
#	Epochs	Batch Size	Output Units in Layers	Model's Accuracy (%) in 10 runs
1.	36	32	12	80.2, 78.0, 73.6, 74.6, 78.4, 78.0, 80.4, 77.8, 81.2, 77.4
2.	72	32	12	78.6, 80.0, 79.4, 80.0, 84.0, 74.8, 80.6, 77.6, 78.6, 79.8
3.	108	32	12	82.4, 80.4, 81.0, 80.0, 83.2, 86.4, 82.0, 81.80, 84.6, 84.2
4.	108	64	12	78.0, 77.8, 77.2, 80.8, 77.0, 80.2, 74.6, 80.0, 85.0, 78.0
5.	108	16	12	82.2, 76.4, 82.8, 87.4, 79.0, 79.2, 83.6, 76.4, 73.4, 82.8
6.	108	32	120	81.8, 83.8, 83.6, 90.6, 89.6, 82.4, 88.2, 85.4, 84.6, 87.2
7.	108	32	120, 240	89.8, 88.6, 88.8, 89.6, 90.4, 88.2, 86.6, 89.6, 87.4, 88.0

C. Discussions

Figure 3 depicts the model's accuracy in ten individual runs arranged in ascending with different hyper-parameter, i.e., *Epochs, Batch Size, and Output Units* values that we fine-tune to optimize the model. The blue line presents the accuracy obtained in the ten runs with Epochs = 108, Batch Size = 32, and Units values (120, 240) showing a consistent result after tuning the hyper-parameter.

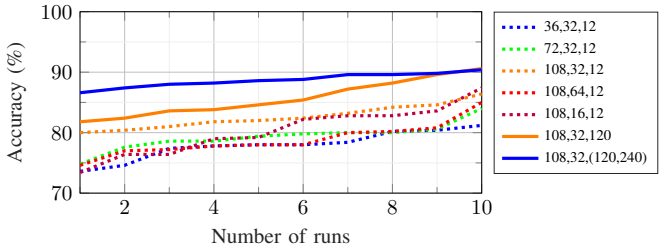

Fig. 3. Model's Accuracy in ascending order with different hyper-parameter (*Epochs, Batch Size, Output Units*) values

The training and validation accuracy plot for 108 epochs can be observed in Figure 4. It is imperative to select the correct batch size and epoch together with the suitable number of training and validation data, which can heuristically optimize the internal variables at each layer, to obtain the best-fit model. A model can be prevented from over-fitting with the help of different Regularizers per-layer basis penalizing the model from obtaining large weights. L1 (Lasso) and L2 (Ridge) are two popular Keras regularization parameters. Alternatively, the dropout method can be applied for addressing DNN over-fitting.

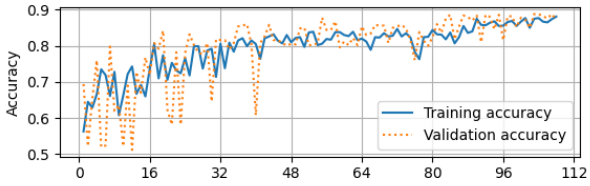

Fig. 4. Training and validation accuracy

Overall, BPP aims to solve the problems encompassing the representation and determination of applications or processes behavior to detect previously seen or unseen threats in a continuous manner.

VI. CONCLUSIONS AND FUTURE WORK

This article presented a behavioral profiling prototype for autonomous protection against emerging threats, reducing the risk of human error, and minimizing the impact of security breaches. The BPP converts unstructured data (i.e, ETW logs) into structured data (i.e., directed acyclic graphs) and extracts graph-level features to distinguish between benign and malicious activities on a Windows-based system using DNN. The BPP obtains an accuracy of $\approx 90\%$ on a collected dataset of two thousand benign and malicious samples. In the future, we will extend our malware dataset and include other OS as well for behavioral profiling.

ACKNOWLEDGMENT

The work is partly funded by the European Union under Horizon Europe Programme - Grant Agreement 101070537 — CrossCon. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] C. Beaman, A. Barkworth, T. D. Akande, S. Hakak, and M. K. Khan, "Ransomware: Recent advances, analysis, challenges and future research directions," *Computers & Security*, vol. 111, p. 102490, 2021.

[2] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–48, 2019.

[3] P. I. Security, "List of computer viruses." https://personalinfosecurity.com/list-of-computer-viruses/, *(Accessed on 10-01-2023)*. online web resource.

[4] F. A. Aboaoja, A. Zainal, F. A. Ghaleb, B. A. S. Al-rimy, T. A. E. Eisa, and A. A. H. Elnour, "Malware detection issues, challenges, and future directions: A survey," *Applied Sciences*, vol. 12, no. 17, p. 8482, 2022.

[5] A. Schranko de Oliveira and R. J. Sassi, "Behavioral malware detection using deep graph convolutional neural networks," *International Journal of Computer Applications*, pp. 1–8, 2021.

[6] H. Studiawan, F. Sohel, and C. Payne, "A survey on forensic investigation of operating system logs," *Digital Investigation*, vol. 29, pp. 1–20, 2019.

[7] M. Ficco, "Malware analysis by combining multiple detectors and observation windows," *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1276–1290, 2021.

[8] S. Saxena and S. Mancoridis, "Malware detection using behavioral whitelisting of computer systems," in *Proceedings of the International Symposium on Technologies for Homeland Security (HST)*, pp. 1–6, IEEE, 2019.

[9] K. Berlin, D. Slater, and J. Saxe, "Malicious behavior detection using windows audit logs," in *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pp. 35–44, 2015.

[10] E. Aharoni, R. Peleg, S. Regev, and T. Salman, "Identifying malicious activities from system execution traces," *IBM Journal of Research and Development*, vol. 60, no. 4, pp. 5–1, 2016.

[11] S. Rana, N. Kumar, A. Handa, and S. K. Shukla, "Automated windows behavioral tracing for malware analysis," *Security and Privacy*, p. e253, 2022.

[12] M. E. Ahmed, H. Kim, S. Camtepe, and S. Nepal, "Peeler: Profiling kernel-level events to detect ransomware," in *Proceedings of the European Symposium on Research in Computer Security*, pp. 240–260, Springer, 2021.

[13] N. Al Mutawa, J. Bryce, V. N. Franqueira, A. Marrington, and J. C. Read, "Behavioural digital forensics model: Embedding behavioural evidence analysis into the investigation of digital crimes," *Digital Investigation*, vol. 28, pp. 70–82, 2019.

[14] L. Luo, C. Zou, S. Narain, and X. Fu, "On teaching malware analysis on latest windows," *Journal of the Colloquium for Information Systems Security Education*, vol. 9, no. 1, pp. 7–7, 2022.

[15] R. C, "Over 95% of all new malware threats discovered in 2022 are aimed at windows." https://atlasvpn.com/blog/over-95-of-all-new-malware-threats-discovered-in-2022-are-aimed-at-windows, *(Accessed on 10-01-2023)*. online web resource.

[16] Microsoft, "Understanding malware and other threats." https://learn.microsoft.com/en-us/microsoft-365/security/intelligence/understanding-malware?view=o365-worldwide, *(Accessed on 10-01-2023)*. online web resource.

[17] Microsoft, "Tracelogging." https://learn.microsoft.com/en-us/windows/win32/etw/about-event-tracing, *(Accessed on 10-01-2023)*. online web resource.