NLP-based Entity Behavior Analytics for Malware Detection

Pejman Najafi
Hasso Plattner Institute
Potsdam, Germany
pejman.najafi@hpi.de

Daniel Koehler
Hasso Plattner Institute
Potsdam, Germany
daniel.koehler@hpi.de

Feng Cheng
Hasso Plattner Institute
Potsdam, Germany
feng.cheng@hpi.de

Christoph Meinel
Hasso Plattner Institute
Potsdam, Germany
christoph.meinel@hpi.de

***Abstract*—In this research, we formulate malware detection as a large-scale data-mining problem within Security Information and Event Management (SIEM) systems. We hypothesize that behavioral analysis of executable/process activities, such as file reads/writes, process creations, network connections, or registry modifications, enables the detection of advanced stealthy malware. To achieve this detection, we model processes behaviors as a set of directed acyclic graph streams and identify outliers in the set of graph streams. We enable this detection by conversion of the behavioral graph streams into documents, embedding using state-of-the-art Natural Language Processing model, and eventually performing novel outlier detection on the high dimensional vector representation of the documents. We evaluate our approach in a real-world setting, next to the SIEM system of a large-scale international enterprise (over 3TB of EDR logs). The proposed method has shown the capability to detect previously unknown threats.**

***Index Terms*—Malware detection, behavior analytics, data mining, SIEM, NLP, graph-based anomaly detection.**

I. Introduction

Within today's organizations, a Security Information and Event Management (SIEM) system is the centralized repository that is expected to hold all security-relevant data, supporting threat detection, hunting, and investigation. Thereby, SIEM systems are used to detect threats that might have bypassed a company's perimeters of defense.

One of the most valuable data sources ingested into SIEM systems is Endpoint Detection and Response (EDR) logs. EDR and similar tools (e.g., Next-Gen Antivirus, Endpoint Protection Platform, Sysmon) monitor and log API calls of the operating system processes while running an executable. This information can include: the executable's hash, file name, file path, command line, parent process, sub-process creation, file writes/reads/deletes, libraries loaded, registry keys touched, network connections made.

In the last few years, there has been an increasing interest in the analysis of such data both in academia and industry. The industry is taking a more heuristic-based approach [1]. For instance, defining certain patterns and rules such as; Alert if Microsoft Word spawned Command-line. In comparison, the research community is evaluating statistical, machine learning as well as data and graph mining-based approaches [4], [8], [13], [25], [26].

While there is a large number of features one can use to train classifiers for maliciousness detection (e.g. file hash and path prevalence), or command-line analysis, we argue such local features may fail to identify more advanced and stealthy malware. For instance, living off the land [14], code injection [6], [17], script-based applications misuse (PowerShell, Python, .NET) or exploited vulnerabilities. In all these scenarios, the executable and process would appear harmless and reputable, yet carry out sophisticated attacks. Furthermore, usually blacklisting such software across the enterprise is infeasible, thus making them very popular to carry out malicious intents.

While detecting such stealthy malware via their local features can be challenging, the behavioral analysis presents better chances. Behavior-based malware detection is a well-studied topic in which one attempts to identify differentiation from normal behavior, fueled by the knowledge and study of *normal* [11]. Classic behavior-based malware detections solutions such as sandboxes often face scalability issues for large-scale enterprises, i.e., not every executable can be assessed prior to execution. However, as previously described, most modern SIEM systems hold event logs that can be used to model process behavior (e.g. sysmon-like events) [12]. This enables behavior-based malware detection at scale using process activity logs.

The main contributions of this paper are summarized below:

- Modeling process/executable behaviors as a set of Directed Acyclic Graphs, and discussing the importance of each behavior in the context of threat detection.
- Propose a Natural Language Processing (NLP) -based approach to graph-based outlier detection. Particularly, proposing the conversion of graph streams to documents, utilizing a state-of-the-art NLP model to embed documents in high dimensional vector space, and run a novel outlier detection algorithm to isolate anomalous documents, hence detect outlier graph streams (i.e., malicious process behavior).
- Evaluate our hypothesis and approach on a large international enterprise landscape, demonstrating the feasibility and effectiveness for advanced threat detection in a real-world setting.

978-1-6654-4331-9/21/$31.00 ©2021 IEEE

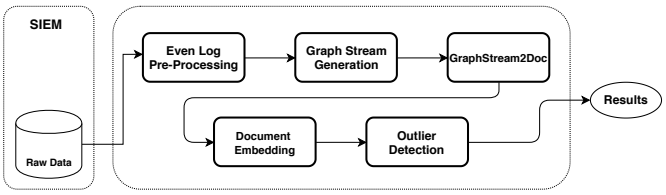

Fig. 1: Proposed approach workflow.

II. Background

A. Dynamic Behavior-Based Malware Detection

In dynamic behavior-based malware analysis, it is essential to identify the criteria for detecting anomalous behavior. Here we shortly expand on the intuition behind each behavior of interest which will be later used to generate the graph streams.

1) Parent and Sub Processes: Sophisticated malware today is no longer a standalone process executing malicious actions on a system. It is rather a system of multiple files, processes and resources that work together to establish an intrusion that goes undetected by anti-malware engines [7], [19]. We therefore aim to extract process-to-process relationships, in particular *process-spawns-process* from the underlying data.

2) File Access (read, write, delete, load): This set of behaviors emphasizes the actions a process takes while touching file system. Independent of the malware, almost all require some modification to the file system. That access might be creating a new file for a later process to execute with higher privileges, encrypting all available files (Ransomware), or reading files for data exfiltration [15], [20].

3) Registry Access: Registry entries can be used in various ways, examples of which include network hooks, auto-start entries, user credentials, and security properties, therefore malware has ever-since been abusing them [2]. According to many research [2], [23], malware can be successfully detected using its access to the registry, furthermore, many anti-malware sandboxing environments use access to registry entries as an identifier for malicious behavior [24], [27]. We, therefore, integrate this behavior into our approach.

4) Network Connections: Network connections are crucial to many modern malware. In particular, in cases such as data exfiltration or botnet Command & Control activities, they are required by the malware's core functionality. In the MITRE ATT&CK Framework[1], a set of use cases enabled by network connections is defined. Anomalous DNS-lookups could also be used as indicators of maliciousness e.g., spear phishing scenarios [10], domain shadowing, or domain generating algorithms [22].

III. Proposed Approach

A. Overview

Figure 1 shows the overview of the proposed approach, which consists of the five stages: *event logs pre-processing*, *graph stream generation*, *graph to document converter*, *document embedding*, and *outlier detection*. In this section, we briefly discuss the main aspects of each stage.

[1]https://attack.mitre.org/matrices/enterprise

B. Behavior Graph-stream Creation

Given a repository of system event logs (e.g., Sysmon, EDR logs), first we create a knowledge graph $G = (\mathcal{V}, \mathcal{E})$, with an object/entity type mapping function $\tau : \mathcal{V} \rightarrow \mathcal{A}$ and link/relationship type mapping function $\varphi : \mathcal{E} \rightarrow \mathcal{R}$. Each object $v \in \mathcal{V}$ is a system entity, globally identifiable by certain attributes (e.g. file hash, domain name, IP and port) and belongs to one particular object type $\tau(v) \in \mathcal{A}$. Each relationship $e \in \mathcal{E}$ is a timestamped system event that belongs to a particular relation $\varphi(e) \in \mathcal{R}$, e.g., fileWrites.

Next, for every given process p identified by its execution context id, and executable image where $p \in \mathcal{V}, \tau(p) = Process$, we generate $s(p, k)$, the graph stream constructed around p with at most k hops (Ordered k-hop Breadth First Traversal).

Lastly, we aggregate the set of graph streams according to their focus process executable, $S(p, k) = \{s(p, k)_1, s(p, k)_2, ..., s(p, k)_n\}$. For instance, all execution of powershell.exe and their corresponding OkBFT graph streams.

C. Graph Embedding

1) Graph to Document Converter: With the recent advancements in the field of natural language processing, today's state-of-the-art NLP models are not only capable of learning the co-occurrences of words but also semantic, context, and order considerations [21].

Therefore, as long as we can meaningfully represent the graph streams as documents, treating nodes and relationships as sequences of words forming sentences, we can rely on the model to maintain the correlation among words (nodes).

In this regard, we convert each graph stream from our set of behavior graph streams, $S(p, k)$ to a document. This is achieved by treating every node in the graph stream as a noun, and each edge as a verb. Note that, each node is globally identifiable by certain attributes, e.g., a file is identified by its name and hash, domain by its fully qualified domain name, IP destination by its IP address and port, (these attributes are highlighted in Figure 2). For instance, if the graph stream shows *powershell.exe* making a network connection to the host *127.0.0.1* on port *80*, the following sentence is generated *"powershell.exe POWERSHELLSHA256 makesNetworkConnectionTo 127.0.0.1:80"*. To clean the sentences, we follow standard NLP document pre-processing steps, such as tokenization and normalization.

2) Document Embedding: While we investigated several embedding techniques for sentences/docs such as Doc2vec [16], in this work we decided to utilize Universal Sentence Encoder (USE) [3]. The reasons for this choice are as follows: First, we were interested in exploring more advanced models which better takes the context and semantics into consideration. Second, a pre-trained model that we could rely on its transfer learning capability with slight fitting. Lastly, USE

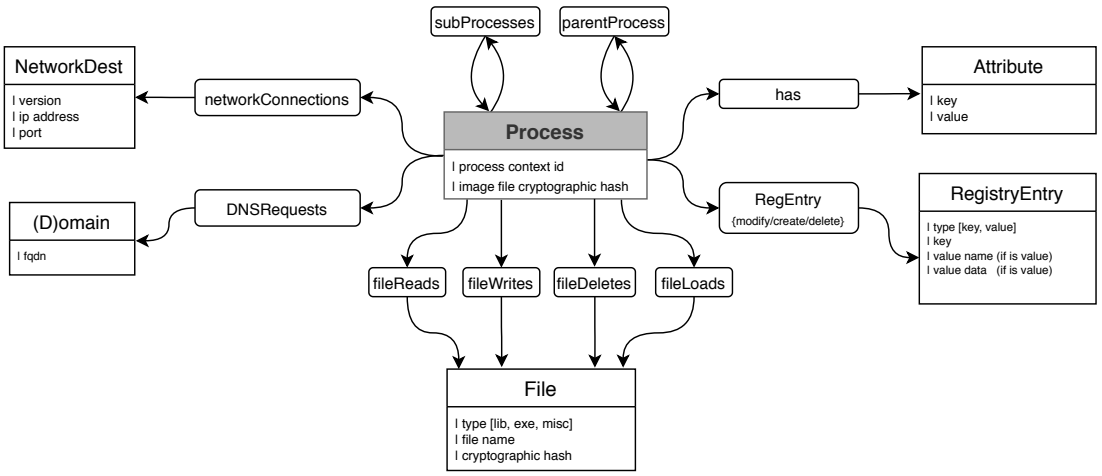

Fig. 2: The schema used to generate graph streams.

offers a model that could be scaled and parallelized when it came to documents embedding.

D. Outlier Detection

Finally, after having, sets of graph streams, converted to documents and embedded into high dimensional vectors, our task is to find the outlier documents, i.e., the malicious graph streams $s(p,k)_x$ within the context of p's executable image.

There are numerous techniques for outlier detection, such as distribution-based, distance-based, or density-based. Each poses some challenges in certain setups, e.g., failure to capture anomalies in uneven distributions, relaying on pre-selection of specific parameters, inability to scale, working with high dimensional data, or the ability to distinguish local and global outliers [5]. All of these are challenges we faced as well. Therefore, we decided to take advantage of the Isolation Forest [18]. Isolation Forest is fundamentally different from other techniques as it does not try to model normality to identify outliers. It rather explicitly isolates anomalies. Most importantly, isolation forest has proven to have the capacity to scale up/out to handle extremely large data size, high-dimensional problems with a large number of irrelevant attributes, and uneven distributions.

IV. Experiment

A. Experiment Setup and Data

Due to scalability requirements, we decided to utilize Apache Spark, which allows us to scale not only our data processing pipeline, but also the graph generation, document embedding, and outlier detection. In addition, to ensure compliance with the company's policies (data processing within a compliant environment), we decided to build our system on Azure Databricks, configured with 8 "Standard_D32s_v3" workers. Thus, having a big data platform backed up by Apache Spark with a total of; 1024-GiB Memory, 246 vCPU Cores, 2048 GiB temp SSD storage).

For this research, we had access to 1 day of process activity logs collected by the SIEM System of a large international enterprise, spanning over 3 TB. These logs were generated from a commercial tool similar to Sysmon. These logs resulted in 900 million graph streams. Throughout our experiments, we kept k equals 1, i.e., the immediate actions of a process. We leave this to our future work to explore the effect of different OkBFTs.

Fig 3 shows the focus file frequency distribution, illustrating there are handful of applications that are widely observed across the Enterprise landscape such as *cmd.exe*, *reg.exe, svchost.exe, WmiPrvSE.exe,* and *powershell.exe*.

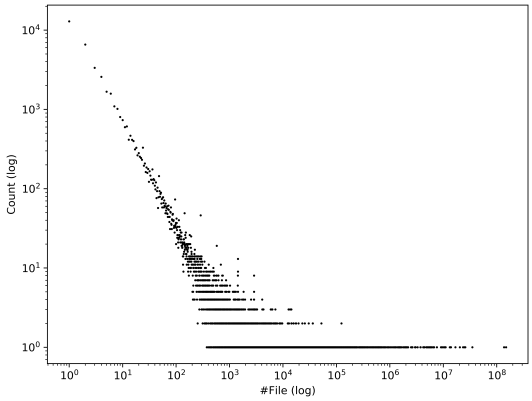

Fig. 3: File count frequency distribution, illustrating how there is only a hand full of executables/files that are heavily observed across the landscape.

B. Evaluation

During the evaluation, we seek to answer whether our proposed approach could be effective to detect malicious events. In this regard, throughout our data set, we were aware of two particular cases of incidents. We knew these two incidents were raised because of malicious `powershell.exe` and `explorer.exe` behavior.

We isolated the graph streams related to those alerts. We then randomly sampled 100 graph streams whose focus process was `explorer.exe` and 100 for `powershell.exe`.

In other words, we assumed 1% contamination (outliers) to test whether our approach could identify these events correctly.

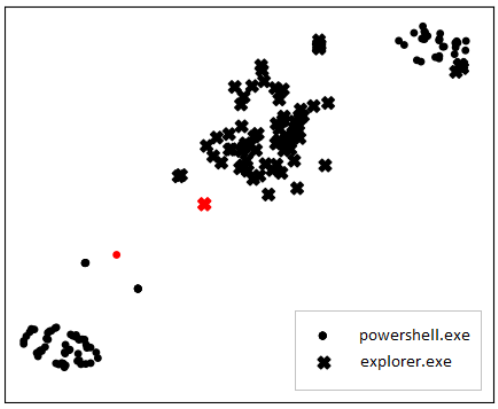

Fig. 4: The t-SNE plot for the embedded graph streams.

Figure 4 shows the t-SNE plot for `powershell.exe` and `explorer.exe`. One can observe the outliers highlighted in red within each community.

Next, after running isolation forest with contamination set to 0.01 we could correctly identify the two outliers. Figure 6 shows the outlier scores distribution.

Lastly, we decided to investigate the next 8 outliers. While all were intriguing, 6 could be verified as false positives associated with certain developers' abnormal behavior. However, 2 could be validated as true positive, resulting in two alerts. We repeated our evaluation (above process) 10 times, and on average, we achieved 0.94 accuracy when manually investigating top 10 outliers. Table I shows the details of the experiment results, and Figure 5 shows Receiver Operating Characteristic (ROC) highlighting 0.98 Area Under a Curve (AUC).

TABLE I: The details of the evaluation.

Metric	Count
Total Number of Samples	200
True Positives	4
True Negatives	190
False Positives	6
False Negatives	0

This experiment was carried on only a sample of data to test our hypothesis. The results showcased the ability of this approach to aid in the detection of previously unknown malicious executions, hence potential stealthy/advanced malware. We would like to explore this approach further in our future work.

V. Related Work

A large number of works have proposed to leverage provenance graphs for threat detection, below we discuss the most relevant ones. While we can only provide a short overview, we would like to encourage the reader to read into the mentioned works' for further information.

Wang et al. propose PROVDETECTOR [26], a malware detection system using kernel-level provenance monitoring to capture the behavior of each target process. Similar to our approach, the authors models a program's runtime behaviors such as file read, write, execute, or network connections as a Directed Acyclic Graph (DAG), next the authors utilize Doc2vec as the embedding technique and eventually pass through Local Outlier Factor (LOF) to detect anomalous patterns and thus previously unseen attacks. Motivated by this work, our effort follows a very similar idea. However, while PROVDETECTOR extracts rare paths of variable length from the provenance graph, we consider all immediate k-hop neighbors of a process as we attempt to model process behavior rather than building a provenance graph. Furthermore, we focus on scaling the pipeline using big data architectures, ensuring the parallelization of each stage. Lastly, while PROVDETECTOR utilizes Doc2vec, we explore a more advanced NLP model (Universal Sentence Encoder) which takes context, semantic, and order into closer consideration.

Hassan et al. [9] propose NoDoze, a threat alert triage system aiming at reducing the number of false-positive alerts within the context of threat detection systems by the rareness of causal path in a provenance graph. The authors later propose an improved prototype system named, RapSheet [8] by first matching EDR system logs to MITRE ATT&CK attack pattern, this work distinguishes itself from NoDoze by correlating alerts that are causally related but appear on different ancestry paths. While our underlying schema to model process behavior is very similar to NoDoze and RapSheet, they were originally designed as an alert triage system, whereas our work focuses on threat hunting by mining the event logs, i.e., threats that no alerts have yet been received for.

VI. Conclusion and Future Work

In this work, we explored the feasibility of behavior-based malware detection using system event or EDR logs collected within a typical enterprise SIEM. We proposed modeling process behavior as a set of observed graph streams. These graph streams are converted to documents and embedded using Universal Sentence Encoder. Once brought into high dimensional vector space, running Isolation Forest enables the detection of anomalous documents, hence potential malicious events of interest.

We carried out our experiment in a real-world setting, next to the SIEM system of a large enterprise showing the feasibility of the approach. We were successfully capable of finding two new incidents, thus showing the effectiveness of this approach with an average accuracy of 0.94.

While our approach has shown the ability to identify outliers successfully, we acknowledge high numbers of false positives, as presented in section IV. This is because anomalous behavior is not necessarily malicious behavior.

Within our future work, we would like to explore the comparison between different embedding and outlier detection algorithms. Furthermore, we aim at experimenting with a larger dataset and performing a thorough analysis of the

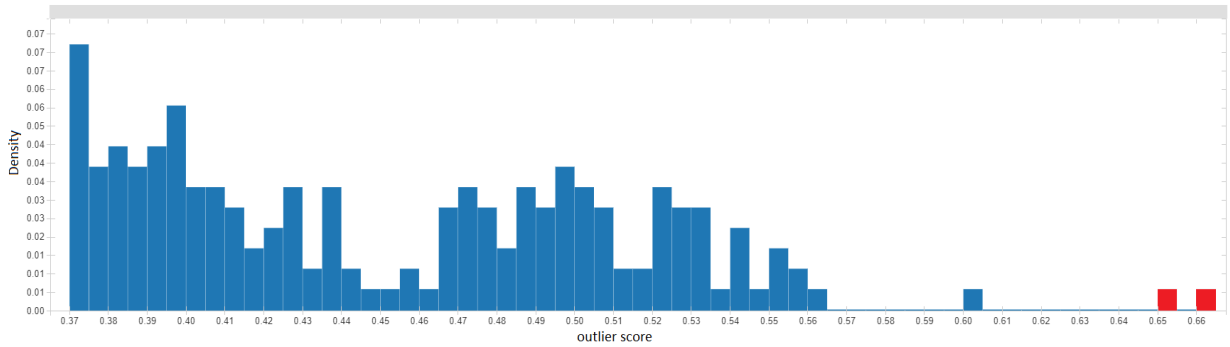

Fig. 5: Outlier scores distribution for the experiment.

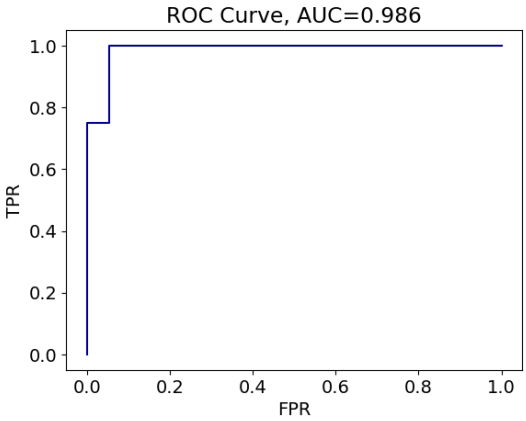

Fig. 6: Receiver operating characteristic curve highlighting the performance of the proposed approach when thresholding.

behavioral features and the choice of k, and their impact on the results with the aim to reduce the false-positive rate.

References

[1] R. Anthony, "Detecting security incidents using windows workstation event logs," *SANS Institute, InfoSec Reading Room Paper*, 2013.

[2] F. Apap, A. Honig, S. Hershkop, E. Eskin, and S. Stolfo, "Detecting malicious software by monitoring anomalous windows registry accesses," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2002, pp. 36–53.

[3] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[4] D. H. P. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-scale graph mining and inference for malware detection," in *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 2011, pp. 131–142.

[5] Z. Cheng, C. Zou, and J. Dong, "Outlier detection using isolation forest and local outlier factor," in *Proceedings of the conference on research in adaptive and convergent systems*, 2019, pp. 161–168.

[6] S. Fewer, "Reflective dll injection," 2008.

[7] G. Hăjmăşan, A. Mondoc, R. Portase, and O. Creţ, "Evasive Malware Detection Using Groups of Processes," in *ICT Systems Security and Privacy Protection*, ser. IFIP Advances in Information and Communication Technology, S. De Capitani di Vimercati and F. Martinelli, Eds. Cham: Springer International Publishing, 2017, pp. 32–45.

[8] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2020.

[9] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *Network and Distributed Systems Security Symposium*, 2019.

[10] J. Hong, "The state of phishing attacks," *Communications of the ACM*, vol. 55, no. 1, pp. 74–81, 2012.

[11] N. Idika and A. P. Mathur, "A Survey of Malware Detection Techniques," *Purdue University*, p. 48, 2007.

[12] Y. Kawakoya, E. Shioji, M. Iwamura, and J. Miyoshi, "Api chaser: Taint-assisted sandbox for evasive malware analysis," *Journal of Information Processing*, vol. 27, pp. 297–314, 2019.

[13] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[14] S. Kumar *et al.*, "An emerging threat fileless malware: a survey and research challenges," *Cybersecurity*, vol. 3, no. 1, pp. 1–12, 2020.

[15] B. J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitraş, "The dropper effect: Insights into malware distribution with downloader graph analytics," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1118–1129.

[16] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, 2014, pp. 1188–1196.

[17] J. Leitch, "Process hollowing," 2013.

[18] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[19] J. A. Morales, M. Main, W. Luo, S. Xu, and R. Sandhu, "Building malware infection trees," in *2011 6th International Conference on Malicious and Unwanted Software*, Oct. 2011, pp. 50–57.

[20] P. OKane, S. Sezer, and K. McLaughlin, "Obfuscation: The hidden malware," *IEEE Security & Privacy*, vol. 9, no. 5, pp. 41–47, 2011.

[21] S. Palachy, "Document embedding techniques," Sep 2019. [Online]. Available: https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d

[22] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A comprehensive measurement study of domain generating malware," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 263–278.

[23] A. Tajoddin and M. Abadi, "Ramd: registry-based anomaly malware detection using one-class ensemble classifiers," *Applied Intelligence*, vol. 49, no. 7, pp. 2641–2658, 2019.

[24] ——, "RAMD: Registry-based anomaly malware detection using one-class ensemble classifiers," *Applied Intelligence*, vol. 49, no. 7, pp. 2641–2658, Jul. 2019. [Online]. Available: https://doi.org/10.1007/s10489-018-01405-0

[25] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[26] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. Gunter *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Symposium on Network and Distributed System Security (NDSS)*, 2020.

[27] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 32–39, 2007.