Useful Feature Extraction for Host-level Anomaly Detection Using a GAN

Isaiah J. King

Abstract

When building an intrusion detection system, the way data is represented ultimately determines how we define anomalous activity. We assert that to characterize the normalcy of a process, the data representation must capture what portions of the system it edited, the times that these edits occurred, and a characterization of its parent. To this end, we propose an architecture that uses an attention mechanism to embed information about which system entities were edited, and feeds it to a conditional GAN. The GAN's generator learns the distribution of process embeddings, given the executable's filename. Its discriminator learns to detect unusual processes embeddings by comparing real embeddings to generated ones. The detector also makes use of a message passing neural network; this ensures embeddings are propagated through the process tree so even benign looking child processes are marked as anomalous if their parent is. Importantly, this process is entirely inductive; the model trained on one host may be applied to other hosts in the same network to classify processes it has never seen. To evaluate this system we test it on log files of a real-world system undergoing a malware campaign. The results of these tests show that this system can outperform other unsupervised anomalous process detection algorithms on this data set and is a viable method for forensic analysis of compromised systems.

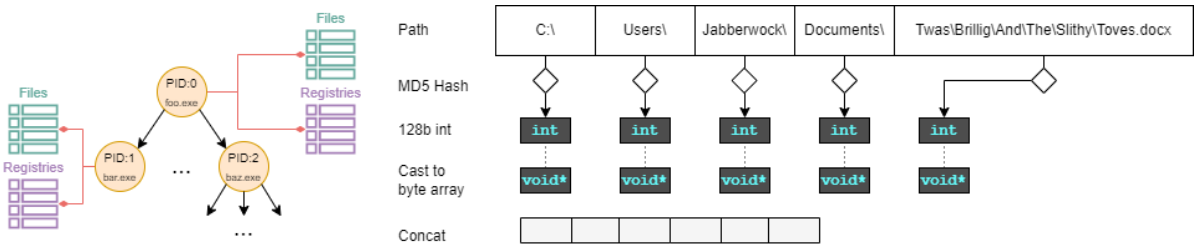
1 Introduction & Background

Auditing log files to identify malware is a perfect task for unsupervised machine learning. Because malware by definition behaves in anomalous ways, detecting it is a matter of defining "normalcy". Here, the problem definition is synonymous to the data representation. When host logs are abstracted as a collection of individual events, as is done by [1, 2], an anomalous process is one with unexpected features. When viewed through the lens of frequency analysis, such as [3], anomalous activity is identified by abnormal rates of activity such as a DDoS, or IP sweep. While sometimes useful, we feel these approaches are deficient, especially to track the *spread* of malware. Samples represent a myopic microcosm within a greater system of activity; the approach ignores relational data informative of malware's movement through a system. To address this, there have been thrusts in the field of graph analytics to study connections within networks [4, 5], or provenance graphs [6, 7, 8]. These techniques fix the issues present in the former two techniques, but they have their own problems. By viewing activity as a singular graph, they neglect to take into account time.

To ignore time is to ignore an entire class of anomalous activity. Events that occur in a strange order, or with unusual frequencies are undetectable with these methods. Though many temporal graph neural network architectures exist [9, 10, 11, 12, 13, 14], to our knowledge, there is only one work which utilizes them for intrusion detection systems [15]. We aim to continue this work and improve upon its shortcomings.

We assert that malicious processes and their children will behave in ways unlike normal processes; ways that manifest in what they edit. This behavior can be observed in numerous advanced persistent threats. For example, the password stealing malware Mimikatz [16] works by first editing a registry key to enable `WDigest`, then redirecting the output of memory dumps to a readable file. To detect threats like these, models must take into account which system entities are edited, in which order, and which parent process spawned them.

To achieve this, we propose a novel intrusion detection system architecture. By incorporating the work of [12] and [17] for continuous temporal graph representation learning as well as the findings



(a) Graph schema

(b) File and registry name vectorization

Figure 1: Host graph data structure. Nodes are processes, and edges represent children, creating a tree structure. Additionally, a list of all files and registries edited by each process is maintained for later processing. This list of strings is processed into a $8\ell \times F_i$ matrix by hashing ℓ levels of the path.

of [18, 19] which show GAN models produce useful negative samples for unsupervised anomaly detection, we hope to build a system capable of tracking the spread of malware in a real-world system. To do this, we represent hosts as continuous temporal graphs of processes, where nodes are individual processes, and directed edges are their children. The inherent tree structure of process graphs will preserve temporal dependencies during GNN message passing [20], however, to fully leverage these networks, we need to produce useful node embeddings, representative of each process' state at a given time.

The contributions of this work are as follows: we will demonstrate that combining the GAN anomaly detection systems proposed by [18] and [19] with continuous inductive graph embedding techniques inspired by [12] yields highly informative spatio-temporal node representations. Further, we show that, when processed, these embeddings can be used for high precision anomaly detection on a real-world cybersecurity data set.

2 Method

In this section, we will discuss how we construct the temporal graph representation of the data, and the model architecture.

2.1 Graph Construction

We define the host graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ as a set of nodes $\mathcal{V}$ representing processes, and a set of edges $\mathcal{E} = \{(u, v) \mid u, v \in \mathcal{V}\}$, representing the parent-child relation between processes. As processes may only have one parent, this ensures that the graph is a tree. In addition to the parent-child relation, we also maintain the name of the executable the process is running and lists of each file and registry key that each process edited during its life, and at what times. The full graph schema is shown in Figure 1a

To represent the objects accessed, we must convert from string to vector. Rather than using an encoding scheme like TF-IDF or a skip-gram based embedder like Word2Vec [21], we prefer an approach that allows for complete inductive representation. As such, we use a memoryless lookup table in the form of a hash function performed at ℓ levels of the input path. The path to the file or registry is split into up to ℓ levels, proceeding forward for files, where earlier levels are more telling, and in reverse for registries, where the opposite is true. These levels are then hashed using MD5 [22], and the 128b output is cast to a byte array which is used as the feature vector for each object accessed by the

process. This process is illustrated in figure 1b. This technique ensures that objects near each other in the storage hierarchy have similar representations. For example, two files saved in the same directory will have identical vectors up to the final 8 indices. Additionally, we store a one-hot encoding of what type of access has occurred (create, delete, modify, etc.) so creating and deleting the same entity are represented in a similar, but discernible way. In addition to these vectors, we also store the times the objects were accessed with respect to the time the process started.

The final data structure is a collection of graphs, each representing the process activity on a single host over one day of the malware campaign. We store 25 host-graphs that were unaffected by the attack to train on, and one compromised host to test on.

2.2 Model Architecture

The model consists of three main components: the process *embedder*, the negative sample *generator*, and the anomaly-detecting *discriminator*. All together, these three structures form a GAN [23].

The objective of the process embedder is to generate informative, inductive embeddings to represent individual processes. Following the work of [12], it uses key, query, value self-attention in conjunction with continuous temporal encoding. We use Time2Vec [17] as the temporal encoding function, which is defined as

$$\phi(t;\theta) = [w_0 t + b_0, \cos(w_1 t + b_1), ..., \cos(w_d t + b_k)] \tag{1}$$

where k is a user-specified hyperparameter, and $\mathbf{w}$ is a trainable parameter. As was proven in [12], this equation approximates a kernel $K(t_1, t_2) = \phi(t_1) \cdot \phi(t_2) \approx |t_1 - t_2|$. In this way, there is a component of the input vector representing temporal similarity, distance between samples in time. This aids in capturing anomalous activity as defined by frequency in addition to the set of accesses.

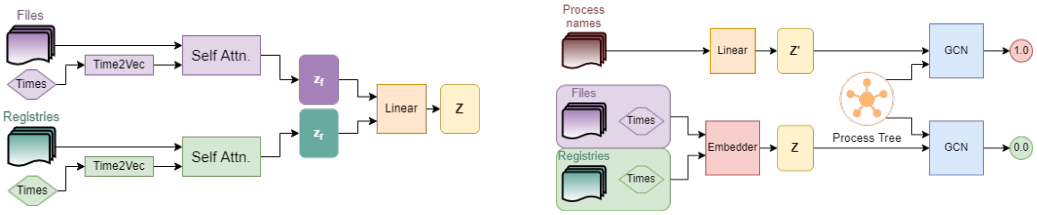
The Time2Vec encoding is concatenated with the sequences of file and registry accesses before they are passed to the attention mechanism. One layer of self-attention is defined as

$$\mathbf{h}^{(\ell)} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d}}\right)\mathbf{V} \tag{2}$$

where $\mathbf{K} = \mathbf{h}^{(\ell-1)}\mathbf{W}_K^{(\ell)} + \mathbf{b}_K^{(\ell)}$, $\mathbf{Q} = \mathbf{h}^{(\ell-1)}\mathbf{W}_Q^{(\ell)} + \mathbf{b}_Q^{(\ell)}$, $\mathbf{V} = \mathbf{h}^{(\ell-1)}\mathbf{W}_V^{(\ell)} + \mathbf{b}_V^{(\ell)}$ are feed forward neural networks (FFNN) with trainable parameters $\mathbf{W}$ and $\mathbf{b}$, and output dimension d. As we are interested in summarizing the collection of accesses in aggregate, the final outputs of the attention mechanism are averaged together to produce a single vector.

In practice, some processes edit upwards of 8,000 objects during their lifetime, while others edit none. To avoid producing enormous matrices during the $\mathbf{Q}\mathbf{K}^\intercal$ matrix multiplication, we randomly sample a constrained number of files and registries each epoch. We found 50 works well. The lists of files and registries are passed to independent self-attention mechanisms; the outputs are concatenated and passed through a final FFNN to project the embeddings into the desired dimension.

Now that we have a mechanism to represent the processes, we need some way of measuring their utility. Ideally, embeddings will be drawn from some distribution of normal process behavior, and unusual processes will have representations that deviate from this norm. To measure this, we use another network whose objective is to assign a normalcy score to these embeddings. To achieve this, we use a simple graph convolutional network (GCN) [20] to further process the embeddings and incorporate the structure of the process tree. GCN's are defined in terms of the symmetrically normalized adjacency



(a) Embedder

(b) Adversarial Network

Figure 2: The training pipeline. The embedder in Subfigure 2a aims to convert a sequence of file and registry encodings into useful embeddings. Meanwhile, and adversarial generator aims to learn the distribution of embeddings a given executable could have. These real embeddings, and generated embeddings are then passed into a graph convolutional network, shown in Subfigure 2b, which aims to distinguish between them, and in the process act as an anomaly detector.

matrix $\mathbf{A}$ and the input matrix $\mathbf{H}$ as

$$\mathbf{H}^{(\ell)} = \sigma(\mathbf{W}\mathbf{H}^{(\ell-1)}\mathbf{A}) \tag{3}$$

The last layer of the GCN will output a 1-dimensional vector tending toward 0 if the sample appears normal, and 1 otherwise. Keeping with the parlance of [23], we refer to this module as the *discriminator*.

To train the embedder and the discriminator requires more than normal samples, however. It is necessary to show the model examples of abnormal processes. As our objective is to find an unsupervised method of anomaly detection, using actual compromised process embeddings is off the table. Additionally, there are so few of them compared to the overwhelmingly normal activity that occurs on hosts. This is where the GAN structure is so powerful.

Inspired by the work of [18] and [19] we train an additional network, the conditional *generator* to produce negative samples. This network only has the encoded names of executables, and random noise as input. From here, it must learn the probable distribution of how processes tend to behave and produce embeddings similar to those the embedder would. This is implemented as a simple FFNN.

2.3 Training

A traditional GAN has the loss function for discriminator $D(x)$ and generator $G(z)$

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(z))] \tag{4}$$

This requires minimal changes when instead of sampling x from a real distribution of data, it is sampled from an embedding of the data, $E(x)$. The objective is now updated as

$$\min_G \max_{D,E} \mathbb{E}_{z\sim E(x\sim X_o)}[\log D(E(x))] + \mathbb{E}_{x\sim X_p}[\log(1 - D(x))] \tag{5}$$

Where X_o are objects (files and registries) and X_p are process names.

Additionally, two optimizations found by [24] were highly effective. One-sided label smoothing, which changes the discriminator's objective from $\log(D(E(x)))$ to $\log(0.1 + D(E(x)))$, such that positive samples are scored as 0.9 improved performance. Feature matching, which updates the generator's

objective to match some intermediate activation in the discriminator on positive samples, rather than its output fits quite well into the encoder-discriminator framework, and considerably improved training stability. If $\mathsf{E}(x)$ is thought of as an intermediate activation in a greater discriminator defined by the composition $\mathsf{D}\circ\mathsf{E}$, then using $\|\mathsf{G}(x_p^{(n)})-\mathsf{E}(x_o^{(n)})\|_2$ as the generator's loss function fits into this paradigm. Finally, we found that also training the discriminator to detect noise, maximizing $\log(1-\mathsf{D}(x\sim\mathcal{N}(0,1)))$ prevented the model from growing overfit as the generator improved.

All together, the loss functions are defined as

$$\begin{aligned}\mathcal{L}_D &= -\log(\mathsf{D}(\mathsf{E}(x_o))+0.1)-\log(1-\mathsf{D}(\mathsf{G}(x_p)))-\log(1-\mathsf{D}(x\sim\mathcal{N}(0,1)))\\ \mathcal{L}_G &= \|\mathsf{G}(x_p)-\mathsf{E}(x_o)\|_2\\ \mathcal{L}_E &= -\log(\mathsf{D}(\mathsf{E}(x_o))+0.1)\end{aligned} \tag{6}$$

3 Experiments

To evaluate the performance of our proposed architecture, we test it on the DARPA OpTC data set [25]. This data set consists of 500 hosts in a network undergoing 5 days of normal activity, followed by 3 days of redteam activity. The data set consists of host logs of activity across the entire network with information about activity on files, processes, the network, user logins, etc..

As we are only interested in a subset of this activity, we parse out only actions related to processes, files, and registries. Table 1 contains a full account of the actions tracked for each object. These objects are then organized into graphs in the manner described in Section 2.1: nodes are processes, edges are process create events, and nodes hold two lists, representing which files and registries it edited during its lifetime.

Object	Actions
Process	CREATE
File	CREATE, DELETE, MODIFY, RENAME, WRITE
Registry	ADD, EDIT, REMOVE

Table 1: OpTC Events Captured

It is difficult to adequately compare our model to prior works. For example, ANUBIS [8] and DeepTaskAPT [26] solve similar problems on this data set but use a supervised learning approach. While these approaches have higher precision and recall than unsupervised techniques, they are unable to detect zero-day vulnerabilities, or other attacks that they have not been explicitly trained for [27]. The only work we could find using a purely unsupervised technique was [28]. Unfortunately, this work reports the results of pure classification (precision and recall values at a specific threshold) rather than the quality of their anomaly scores (average precision, or AUC). In Table 2 We report AUC and AP in addition to the precision and recall if the top 200 most anomalous processes were flagged as anomalous.

Unfortunately, in the time allotted for this project, we could not find a good way of validating the model as it ran. As a result, the evaluation metrics were wildly inconsistent. In future tests where we have more data, the best solution seems to be validation on a known anomalous host graph. However, in the time allotted we could only create labels for one. For our tests, we selected the model with the lowest encoder loss, as this seemed most correlated to the evaluation results, but even this was inconsistent. As a result, what's reported is the best scoring model after 5 independent runs.

Method	AUC	AP	Pr.	Re.
Our Method	**0.9520**	**0.6745**	**0.8300**	**0.7313**
Our Method no t2v	0.8424	0.4933	0.6700	0.5903
Our Method No GNN	0.4949	0.0748	0.0250	0.0220
Our Method No t2v+No GNN	0.4887	0.0704	0.0350	0.0308
Prior Work [28]	–	–	0.8000	0.6600

Table 2: Comparison of Our Method to Prior Work

From the results, it is apparent that our model out-performs the prior work, with slightly higher precision, and much higher recall. This means that the anomaly scores assigned to processes by our model had as low or better false positive scores than the prior work, but also could detect anomalous activity that the prior work could not, as evidenced by the higher recall score. We suspect that this is because our model takes into account time, which is the main difference between the inputs to our model and the prior work's.

Additionally, we ran two ablation studies to examine the value of using the graph structure and the temporal encoding. We found that clearly both of these components add a great deal of value, in particular the GNN. Without it the anomaly detector performs no better than random, as evidenced by the AUC score being close to 0.5. Without the Time2Vec module, the model performs respectably, but still not as well as the prior work.

4 Conclusion

In this project, we found that by leveraging information about which files and registries processes edit, and at which times, it is possible to create a highly precise anomaly detection system. However, the model could be improved further. There are many features in the OpTC data that we did not account for: inter-process communication, modules used, and network traffic all seem ripe for deeper exploration in future work. The results of this project show that this is a promising research direction, and support our hypothesis: processes can be categorized as anomalous inductively by which files and registries they edit, and when they do it in time.

References

[1] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," *arXiv:1802.09089 [cs]*, May 2018. arXiv: 1802.09089.

[2] A. Golczynski and J. A. Emanuello, "End-To-End Anomaly Detection for Identifying Malicious Cyber Behavior through NLP-Based Log Embeddings," *arXiv:2108.12276 [cs]*, Aug. 2021. arXiv: 2108.12276.

[3] S. Bhatia, B. Hooi, M. Yoon, K. Shin, and C. Faloutsos, "Midas: Microcluster-Based Detector of Anomalies in Edge Streams," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3242–3249, Apr. 2020. Number: 04.

[4] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting Lateral Movement in Enterprise Computer Networks with Unsupervised Graph {AI}," pp. 257–268, 2020.

[5] Z. Li, X. Cheng, L. Sun, J. Zhang, and B. Chen, "A Hierarchical Approach for Advanced Persistent Threat Detection with Attention-Based Graph Neural Networks," *Security and Communication Networks*, vol. 2021, p. e9961342, May 2021. Publisher: Hindawi.

[6] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, (New York, NY, USA), pp. 1795–1812, Association for Computing Machinery, Nov. 2019.

[7] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1137–1152, May 2019. ISSN: 2375-1207.

[8] M. M. Anjum, S. Iqbal, and B. Hamelin, "ANUBIS: A Provenance Graph-Based Framework for Advanced Persistent Threat Detection," *arXiv:2112.11032 [cs]*, Dec. 2021. arXiv: 2112.11032.

[9] E. Hajiramezanali, A. Hasanzadeh, N. Duffield, K. R. Narayanan, M. Zhou, and X. Qian, "Variational Graph Recurrent Neural Networks," *arXiv:1908.09710 [cs, stat]*, Apr. 2020. arXiv: 1908.09710 version: 3.

[10] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A Temporal Graph ConvolutionalNetwork for Traffic Prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, pp. 3848–3858, Sept. 2020. arXiv: 1811.05320.

[11] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-Time Dynamic Network Embeddings," in *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*, (Lyon, France), pp. 969–976, ACM Press, 2018.

[12] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive Representation Learning on Temporal Graphs," *ICLR*, Feb. 2020. arXiv: 2002.07962.

[13] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing Network Dynamics using Dynamic Graph Representation Learning," *Knowledge-Based Systems*, vol. 187, p. 104816, Jan. 2020. arXiv: 1809.02657.

[14] A. Divakaran and A. Mohan, "Temporal Link Prediction: A Survey," *New Generation Computing*, vol. 38, pp. 213–258, Mar. 2020.

[15] I. J. King and H. H. Huang, "Euler: Detecting network lateral movement via scalable temporal graph link prediction," in *Network and Distributed Systems Security Symposium*, NDSS, (San Diego, CA), The Internet Society, 2022.

[16] J. Mulder and M. Stingley, "Mimikatz overview, defenses and detection," *SANS Institute, retrieved from the Internet, https://www. sans. org/reading-room/whitepapers/detection/mimikatz-overview-defenses-detection-36780 Feb*, vol. 18, 2016.

[17] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker, "Time2Vec: Learning a Vector Representation of Time," *arXiv:1907.05321 [cs]*, July 2019. arXiv: 1907.05321.

[18] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient GAN-Based Anomaly Detection," *ICLR*, May 2019. arXiv: 1802.06222.

[19] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially Learned Anomaly Detection," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 727–736, Nov. 2018. ISSN: 2374-8486.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations (ICLR)*, 2017.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[22] R. Rivest and S. Dusse, "The md5 message-digest algorithm," 1992.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[24] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[25] DARPA, "Transparent computing engagement 5." `https://github.com/FiveDirections/OpTC-data`, 2019. (Accessed on 03/15/2022).

[26] M. Mamun and K. Shi, "Deeptaskapt: Insider apt detection using task-tree based deep learning," *arXiv preprint arXiv:2108.13989*, 2021.

[27] V. Jyothsna, R. Prasad, and K. M. Prasad, "A review of anomaly based intrusion detection systems," *International Journal of Computer Applications*, vol. 28, no. 7, pp. 26–35, 2011.

[28] A. Golczynski and J. A. Emanuello, "End-to-end anomaly detection for identifying malicious cyber behavior through nlp-based log embeddings," *arXiv preprint arXiv:2108.12276*, 2021.