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

Isaiah J. King
The George Washington University, School of Engineering and Applied Sciences

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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 out-perform other unsupervised anomalous process detection algorithms on this data set and is a viable method for forensic analysis of compromised systems.

Introduction

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, an anomalous process is one with unexpected features. When viewed through the lens of frequency analysis, 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.

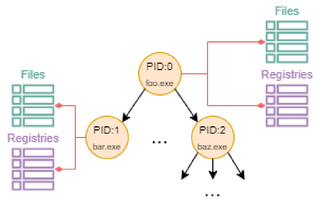
We assert that malicious processes and their children will behave in ways unlike normal processes; ways that manifest in what they edit. This behavior can be observed in numerous advanced persistent threats. For example, the password stealing malware Mimikatz works by first editing a registry key to 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 [1] and [2] for continuous temporal graph representation learning as well as the findings of [3,4] 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, however, to fully leverage these networks, we need to produce useful node embeddings, representative of each process' state at a given time

Methodology

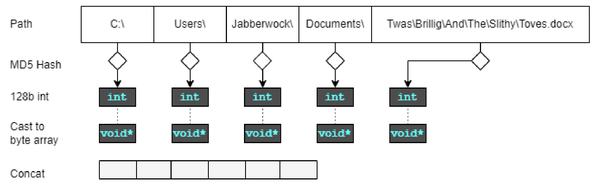
Graph Construction
We define the host graph $\mathcal{G}=\{\mathcal{V},\mathcal{E}\}$ as a set of nodes $\mathcal{V}$ representing processes, and a set of edges $\mathcal{E} = \{(u,v) \mid u,v \in \mathcal{V}\}$ representing the parent-child relation between processes. As processes may only have one 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.

To represent the objects accessed, we must convert from string to vector. 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



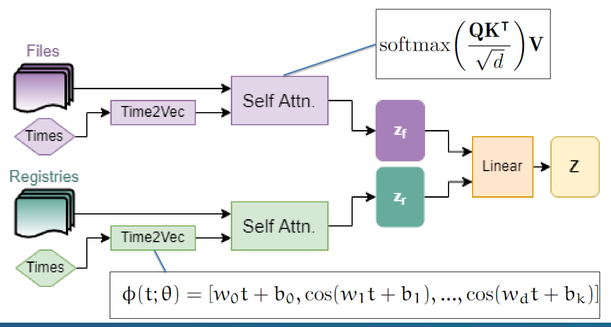
of the input path. The path to the file or registry is split into up to ℓ levels, proceeding forward for files, where earlier levels are more telling, and in reverse for registries, where the opposite is true. These levels are then hashed using MD5, and the 128b output is cast to a byte array which is used as the feature vector for each object accessed by the process.

This 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.

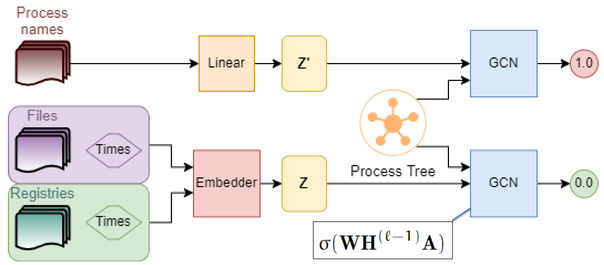


Model Architecture
The model consists of three main components: the process *embedder*, the negative sample *generator*, and the anomaly-detecting *discriminator*. All together, these three structures form a GAN [5]

The objective of the process embedder is to generate informative, inductive embeddings to represent individual processes. Following the work of [1], it uses key, query, value self-attention in conjunction with continuous temporal encoding. We use Time2Vec [2] as the temporal encoding function



Inspired by the work of [3] and [4] 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. 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, the *discriminator*, whose objective is to assign a normalcy score to these embeddings. To achieve this, we use a simple graph convolutional network (GCN) [6] to further process the embeddings and incorporate the structure of the process tree.



Training Details
We used Binary Cross Entropy loss on the discriminator and embedder, and additionally had the discriminator classify noise as negative to prevent overfitting. For the generator we used feature matching loss as described in [7].

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

Experimental Setup
To evaluate the performance of our proposed architecture, we test it on the DARPA OpTC data set [8]. 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.

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

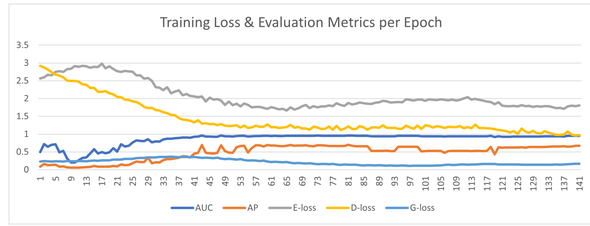
Embedder	t2v dim	8-None
	Attn. Hidden	64-ReLU-drop 0.25
	Layers	4
	Attn. Heads	8
	Attn Output	32-ReLU
	Final Output	64-Sigmoid
Generator	Noise	8
	Hidden	128-RReLU
	Output	64-Sigmoid
Discriminator	GCN hidden	64-tanh-drop 0.25
	Layers	2
	Output	1-Sigmoid

Results

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

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.



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] Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive Representation Learning on Temporal Graphs," *International Conference on Learning Representations* (ICLR), Feb. 2020.
[2] 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
[3] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient GAN-Based Anomaly Detection," *International Conference on Learning Representations* (ICLR), May 2019.
[4] 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.
[5] 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.
[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations* (ICLR), 2017.
[7] 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
[8] DARPA, "Transparent computing engagement 5." https://github.com/FiveDirections/ OpTC-data, 2019. (Accessed on 03/15/2022).