Anomaly Detection Using AutoEncoders: the Advanced Persistent Threats Case

Ngoc Nhu Hoang

Advised by: Prof. Talal Rahwan. Department of Computer Science, New York University Abu Dhabi

Introduction

Advanced persistent threats (APTs) are stealthy and covert cyber-attacks designed to penetrate specific computer networks through exfiltration, corruption of data, or damaging critical systems. One method for identifying APTs is to detect anomalous OS-level activities, which are not easily detectable using traditional security measures as APTs stay dormant over a long time, mimic normal operations, and constitute only a minuscule part of all operations.

This paper presents and evaluates the **AutoEncoder** and **Adversarial AutoEncoder** models trained to reconstruct normal data faithfully, highlighting anomalous data whose reconstruction errors are high. The two models show good performances on par with or surpassing classical anomaly detection algorithms. The ability to isolate suspicious activities in OS-level data is of significant importance to detecting APTs early and mitigating impacts on the system.

Methodology

The following methods use deep neural networks trained on normal data to learn some meaningful compressed representation of regularities that can be used to reconstruct the data as faithfully as possible and further distinguish anomalies.

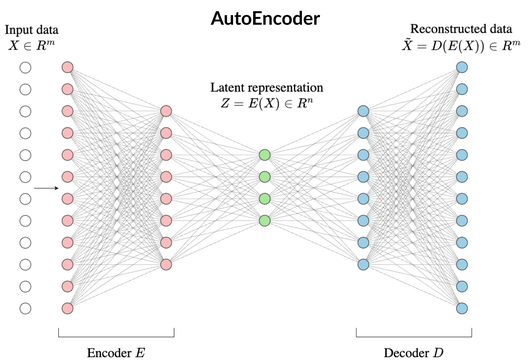

Figure 1. General architecture of the AutoEncoder model.

The AutoEncoder learns to capture the most important compressed features of the normal data that can be used to reconstruct the data as accurately as possible and will fail to reconstruct data with abnormal patterns.

Loss function: $L_{AE} = |X - \tilde{X}| = |X - D(E(X))|$

Anomaly score: $A(x) = |x - \tilde{x}| = |x - D(E(x))|$

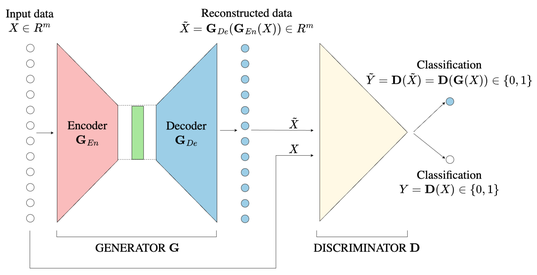

Figure 2. General architecture of the Adversarial AutoEncoder model.

Inspired by the Generative Adversarial Network [3] architecture, a deep neural network acts as the discriminator attempting to separate original data from data reconstructed by the AutoEncoder (the generator). The AutoEncoder is further motivated to reconstruct data that resemble the original data as closely as possible to fool the discriminator.

Loss functions:

- Discriminator: $L_D = |\overrightarrow{1} - \mathbf{D}(X)| + |\overrightarrow{0} - \mathbf{D}(\mathbf{G}(X))|$
- Generator: $L_G = |X - \mathbf{G}(X)| - \lambda \times L_D$

Anomaly score: $A(x) = |x - \tilde{x}| = |x - \mathbf{G}(x)| = |x - \mathbf{G}_{De}(\mathbf{G}_{En}(x))|$

Data

Data from DARPA's Transparent Computing program [1] includes system operations recorded while APT-style attacks are carried out on 4 OS: **Android, Linux, BSD, Windows**. Each system has two scenarios: **Pandex** and **Bovia**. The final data includes Boolean-valued datasets called contexts, each representing an aspect of process behavior. Each OS/scenario pair has 5 contexts: `Event`, `Exec`, `Parent`, `Netflow`, `All`. In most datasets, anomalous processes constitute under 1% of all processes.

Object ID	EVENT_CLONE	EVENT_CHECK_FILE_ATTRIBUTES	EVENT_OTHER	EVENT_MPROTECT	EVENT_CLOSE	EVENT_CREATE_OBJECT	EVENT_LSEEK	EVENT_UNLINK	EVENT_WAIT	EVENT_MODIFY_PROCESS	EVENT_RECVFROM	EVENT_MODIFY_FILE_ATTRIBUTES	EVENT_WRITE	EVENT_BIND	EVENT_READ	EVENT_RENAME	EVENT_OPEN	EVENT_LOADLIBRARY	EVENT_CONNECT	EVENT_SENDTO	EVENT_SENDMSG
285d5fed-06dc-32ae-a04a-13cc9426616b	0	1	1	0	1	1	0	1	0	0	0	1	1	0	1	1	1	1	0	0	0
1e3548c0-b030-3591-97ac-71b67bbcb305	0	1	1	0	1	1	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0
b4f1724e-0ba1-316b-973f-69e5d5e3490c	0	1	1	0	1	1	0	1	0	0	0	1	1	0	1	1	1	0	0	0	0
e2a4e818-3ce2-3626-8e22-134b542d1d77	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
.....																					

Figure 3. A sample of processes from one dataset.

Evaluation method

Normalized discounted cumulative gain

The models prioritize assigning higher scores to anomalous data points and producing a ranking of the processes based on how anomalous they are. The metric chosen is the **normalized discounted cumulative gain (nDCG)** [4] typically used in Information Retrieval to measure the ability to retrieve relevant entities. nDCG score ranges from 0 to 1, with 1 being the best score i.e. when all anomalies are ranked at the top.

$$DCG = \sum_{i=1}^{N} \frac{rel_i}{\log_2(i+1)}; nDCG = \frac{DCG}{iDCG}$$

rel_i: relevance score of the i^{th} entry. iDCG: best achievable DCG.

Baseline algorithms

Performances of the two models are evaluated against 4 unsupervised algorithms for anomaly detection previously evaluated on the same datasets [2]: **Attribute Value Frequency, Frequent Pattern Outlier Factor, Outlier Degree, One-Class Classification by Compression**.

Results

	Process		All	Event	Exec	Parent	Netflow
Cadets (BSD)	Pandex	AE	0.8215	0.5791	0.8165	0.5358	0.1165
		AAE	0.7268	0.6394	0.6276	0.5822	0.1175
	Bovia	AE	0.8077	0.4823	0.8168	0.7992	0.1519
		AAE	0.9079	0.4848	0.8524	0.8035	0.0760
5dir (Windows)	Pandex	AE	0.5931	0.7086	0.2272	0.1970	0.6289
		AAE	0.5989	0.6676	0.2882	0.1871	0.6495
	Bovia	AE	0.4072	0.2639	0.3256	0.4303	0.0973
		AAE	0.3952	0.2514	0.3218	0.4200	0.1298
Trace (Linux)	Pandex	AE	0.6064	0.4204	0.3013	0.2397	0.3953
		AAE	0.7711	0.4916	0.4080	0.2278	0.4006
	Bovia	AE	0.7054	0.4761	0.4871	0.2564	0.3725
		AAE	0.6494	0.4234	0.4796	0.2850	0.3633
Clearscope (Android)	Pandex	AE	0.7815	0.6708	0.4033	NA	0.6014
		AAE	0.7857	0.5484	0.5885	NA	0.6284
	Bovia	AE	0.2712	0.6357	0.5212	NA	0.3904
		AAE	0.2949	0.2631	0.3952	NA	0.1681

Table 1. nDCG scores of the two architectures on all available contexts. NA: data not available.

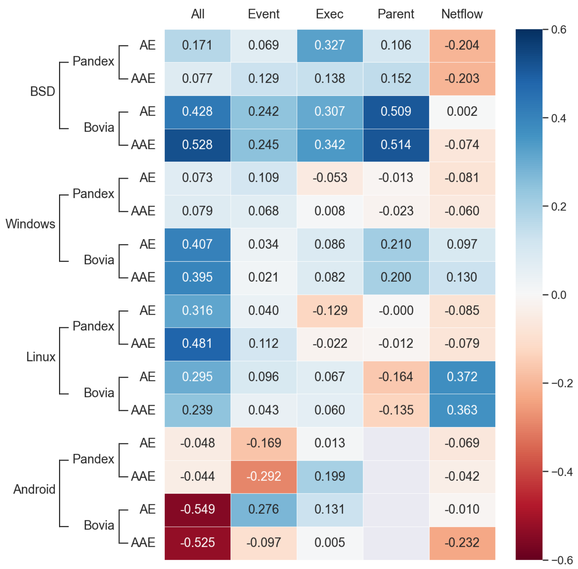

Figure 4. Differences in nDCG scores achieved by the models and the best-performing baseline algorithms.

Figure 4 compares the performances of the two AutoEncoder-based methods against the baseline algorithms. Each cell visualizes:

$$\Delta = nDCG_{\text{AE/AAE}} - \max(nDCG_{\{\text{AVF, OC3, OD, FPOF}\}})$$

The red cells mostly concentrate in the Android system and the `Netflow` context. These datasets either contain few processes (Android) or processes with very high dimensions (`Netflow`). In 26 out of the 38 datasets (68.4%), the best nDCG score is achieved by either the AutoEncoder or the Adversarial AutoEncoder model.

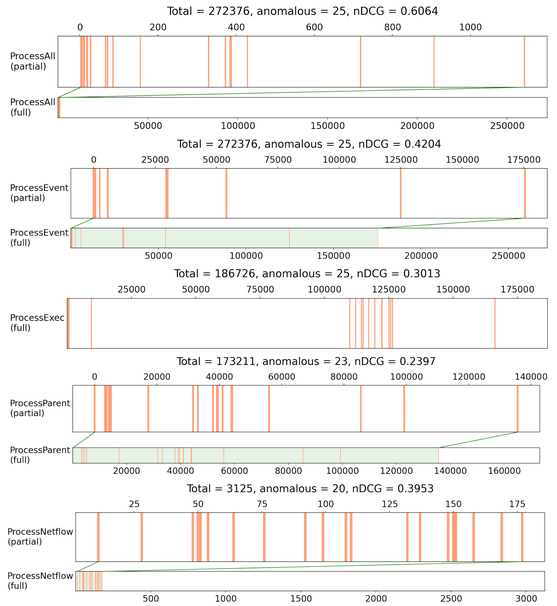

Figure 5. AutoEncoder rankings of anomalies on five contexts of the Linux system, Pandex scenario.

Conclusion and future works

This project presents two AutoEncoder-based methods to detect APTs using OS-level data. The models are trained exclusively on normal to assign anomaly scores to the processes. The two models show good performances on par with or surpassing classical anomaly detection algorithms, especially in cases where the baseline algorithms fail to finish within a reasonable time. The experiments and results from this paper can contribute to the development of efficient methods for the early detection of APTs in provenance data, potentially mitigating the damages of such attacks on the system.

In the future, different types of neural networks can be used and the layers should accommodate the varying dimensions of data. The Adversarial AutoEncoder's discriminator can be modified to better balance the workloads. Further research is needed to establish a reliable threshold for abnormality to further isolate anomalous processes.

Acknowledgement

The author thanks Professor Talal Rahwan and Sidahmed Benabderrahmane for their guidance and feedback on this project.

References

[1] Transparent computing.

[2] Ghita Berrada, Sidahmed Benabderrahmane, James Cheney, William Maxwell, Himan Mookherjee, Alec Theriault, and Ryan Wright.
A baseline for unsupervised advanced persistent threat detection in system-level provenance.
CoRR, abs/1906.06940, 2019.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.
Generative adversarial nets.
In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[4] Kalervo Järvelin and Jaana Kekäläinen.
Cumulated gain-based evaluation of IR techniques.
ACM Transactions on Information Systems, 20(4):422–446, October 2002.